TREINAMENTOS

Lógica de Programação

Lógica de Programação

16 de setembro de 2013
As apostilas atualizadas estão disponíveis em **www.k19.com.br**

Esta apostila contém:

- 238 exercícios de fixação.
- 82 exercícios complementares.
- 4 desafios.
- 61 questões de prova.

Sobre a K19

A K19 é uma empresa especializada na capacitação de desenvolvedores de software. Sua equipe é composta por profissionais formados em Ciência da Computação pela Universidade de São Paulo (USP) e que possuem vasta experiência em treinamento de profissionais para área de TI.

O principal objetivo da K19 é oferecer treinamentos de máxima qualidade e relacionados às principais tecnologias utilizadas pelas empresas. Através desses treinamentos, seus alunos tornam-se capacitados para atuar no mercado de trabalho.

Visando a máxima qualidade, a K19 mantém as suas apostilas em constante renovação e melhoria, oferece instalações físicas apropriadas para o ensino e seus instrutores estão sempre atualizados didática e tecnicamente.

Seguro Treinamento

Na K19 o aluno faz o curso quantas vezes quiser!

Comprometida com o aprendizado e com a satisfação dos seus alunos, a K19 é a única que possui o Seguro Treinamento. Ao contratar um curso, o aluno poderá refazê-lo quantas vezes desejar mediante a disponibilidade de vagas e pagamento da franquia do Seguro Treinamento.

As vagas não preenchidas até um dia antes do início de uma turma da K19 serão destinadas ao alunos que desejam utilizar o Seguro Treinamento. O valor da franquia para utilizar o Seguro Treinamento é 10% do valor total do curso.

Termo de Uso

Termo de Uso

Todo o conteúdo desta apostila é propriedade da K19 Treinamentos. A apostila pode ser utilizada livremente para estudo pessoal . Além disso, este material didático pode ser utilizado como material de apoio em cursos de ensino superior desde que a instituição correspondente seja reconhecida pelo MEC (Ministério da Educação) e que a K19 seja citada explicitamente como proprietária do material.

É proibida qualquer utilização desse material que não se enquadre nas condições acima sem o prévio consentimento formal, por escrito, da K19 Treinamentos. O uso indevido está sujeito às medidas legais cabíveis.

Conheça os nossos cursos

K01- Lógica de Programação

K02 - Desenvolvimento Web com HTML, CSS e JavaScript

K03 - SQL e Modelo Relacional

K11 - Orientação a Objetos em Java

K12 - Desenvolvimento Web com JSF2 e JPA2

K21 - Persistência com JPA2 e Hibernate

K22 - Desenvolvimento Web Avançado com JFS2, EJB3.1 e CDI

K23 - Integração de Sistemas com Webservices, JMS e EJB

K41 - Desenvolvimento Mobile com Android

K51 - Design Patterns em Java

K52 - Desenvolvimento Web com Struts

K31 - C# e Orientação a Objetos

K32 - Desenvolvimento Web com ASP.NET MVC

www.k19.com.br/cursos

CAPÍTULO 1

INTRODUÇÃO

O que é um Computador?

Atualmente, os computadores estão presentes no cotidiano da maioria das pessoas. Você, provavelmente, já está acostumado a utilizar computadores no seu dia a dia. Mas, será que você conhece o funcionamento básico de um computador? A seguir, listaremos os principais elementos de um computador e suas respectivas funções.

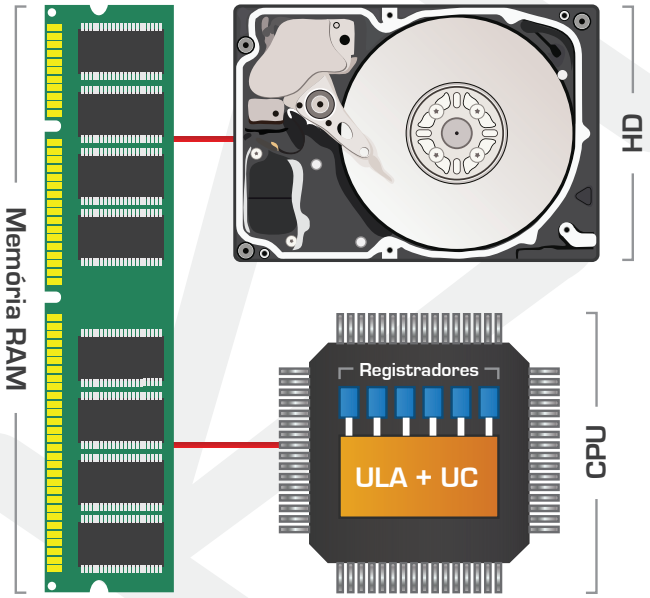

Figura 1.1: Principais elementos de um computador

CPU (Unidade Central de Processamento - Processador): A CPU é o "cérebro" que controla o funcionamento dos outros componentes do computador e realiza todo o processamento necessário. Esse processamento consiste basicamente na realização de operações matemáticas e operações de armazenamento ou recuperação de dados.

Registradores: Os registradores armazenam os dados que estão sendo processados pela CPU. O acesso ao conteúdo dos registradores é extremamente rápido. Por outro lado, eles não possuem muito espaço. Dessa forma, não é possível armazenar uma grande quantidade de informação

Memória RAM: Os dados utilizados pelos programas que estão abertos são armazenados na memória RAM. O acesso ao conteúdo da memória RAM é rápido porém mais lento do que o acesso ao conteúdo dos registradores. Por outro lado, o espaço da memória RAM é bem maior do que o espaço dos registradores.

Disco Rígido: Os dados armazenados nos registradores e na memória RAM são descartados quando o computador é desligado. O conteúdo que não pode ser descartado ao desligar o computador deve ser armazenado no disco rígido. O acesso ao disco rígido é bem mais lento do que o acesso a memória RAM mas, em geral, o espaço é bem maior.

Os computadores são capazes se comunicar com dispositivos periféricos como teclado, mouse, monitor, caixa de som, impressoras, projetores, entre outros. Eles também são capazes de se comunicar com outros computadores. Essa comunicação é realizada através das diversas portas físicas que os computadores possuem. A seguir listaremos algumas portas físicas e as suas respectivas funções.

Ethernet: Utilizada para conectar um computador a uma rede local de computadores. Através dessa porta, um computador pode enviar e receber dados de outros computadores.

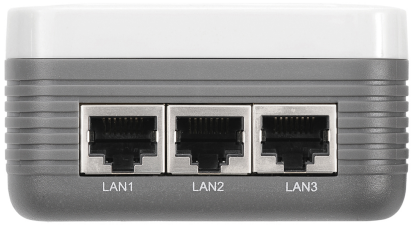

Figura 1.2: Porta Ethernet

Paralela: Essa porta foi criada conectar um computador a uma impressora. Hoje, é utilizada também para conectar computadores a scanners, câmeras de vídeo, entre outros dispositivos.

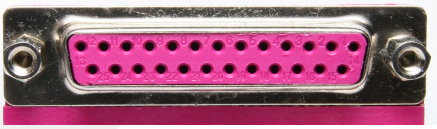

Figura 1.3: Porta Paralela

PS/2: Teclados e mouses antigos são conectados aos computadores através dessa porta.

Figura 1.4: Porta PS/2

USB: Atualmente, é a porta mais utilizada. Diversos dispositivos são conectados aos computadores através das portas USB. Por exemplo, teclados, mouses, impressoras, celulares, HDs externos, entre outros.

Figura 1.5: Porta USB

HDMI: Essa porta é utilizada para transmissão digital de áudio e vídeo.

Figura 1.6: Porta HDMI

Para resumir, podemos dizer que um computador é uma máquina que executa comandos matemáticos e armazena dados. Você deve estar se perguntando, como os computadores conseguem realizar tarefas tão sofisticadas se eles apenas executam comandos matemáticos e armazenam dados? A resposta é simples. Os computadores são programados por pessoas e essas pessoas conseguem criar programas que realizam tarefas sofisticadas a partir dos recursos básicos oferecidos pelos computadores. Da mesma forma que pessoas são capazes de produzir pinturas sofisticadas utilizando apenas tinta, pincel e quadro.

Analogia

Figura 1.7: Analogia entre programar e pintar

Formato Binário

Os computadores são capazes de receber, armazenar e enviar dados. Contudo, esses dados são definidos em um formato que não é comum no dia a dia das pessoas. Eles são definidos em formato binário. Por exemplo, as pessoas estão acostumadas a lidar com os números em formato decimal. Por outro lado, os computadores trabalham com números em formato binário. Veja a seguir, a representação decimal e binária de alguns números.

Decimal	Binário
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Decimal	Binário
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100
21	10101

Decimal	Binário
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101
30	11110
31	11111
32	100000

Tabela 1.1: Representação decimal e binária

Os textos também são manipulados pelos computadores em formato binário. Normalmente, cada caractere de um texto corresponde a uma sequência de oito 0s e 1s. Os caracteres são mapeados para formato binário através de padrões como **ASCII** (http://pt.wikipedia.org/wiki/ASCII) e **Unicode** (http://pt.wikipedia.org/wiki/Unicode). Veja a seguir, a representação binária de alguns caracteres de acordo com o padrão ASCII.

Caractere	Binário
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011

Caractere	Binário
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110

Caractere	Binário
W	01010111
X	01011000
Y	01011001
Z	01011010
a	01100001
b	01100010
c	01100011
d	01100100
e	01100101
f	01100110
g	01100111

Tabela 1.2: Representação binária de caracteres seguindo o padrão ASCII

Como vimos, os números e os caracteres de um texto são facilmente representados em formato binário. Contudo, os computadores também são capazes de manipular imagens, áudio e vídeo. Para esses tipos de dados, a transformação para formato binário é bem mais complicada. Pesquise por PNG, MP3 e AVI que são formatos binários de imagens, áudios e vídeos através dos seguinte endereços:

- PNG - http://www.w3.org/TR/PNG/
- MP3 - http://en.wikipedia.org/wiki/MP3
- AVI - http://en.wikipedia.org/wiki/Audio_Video_Interleave

Unidades

É importante ser capaz de mensurar a quantidade de dados que um computador pode armazenar ou transmitir. Essa mensuração pode ser realizada com ajuda das unidades de medida. A unidade de medida básica é o **Bit**. Cada 0 ou 1 que um computador armazena ou transmite é um Bit. Além dessa unidade básica, existem várias outras. Veja a seguir, algumas delas.

Byte (B)

- 8 Bits

Quilobyte (kB)

- 1024 B
- 8192 Bits

Megabyte (MB)

- 1024 kB
- 1048576 B
- 8388608 Bits

Gigabyte (GB)

- 1024 MB
- 1048576 kB
- 1073741824 B
- 8589934592 Bits

Terabyte (TB)

- 1024 GB
- 1048576 MB
- 1073741824 kB
- 1099511627776 B
- 8796093022208 Bits

Arquiteturas de Processadores

Os comandos que os processadores dos computadores executam são definidos em formato binário. Considere o exemplo fictício a seguir com algumas instruções para um determinado processador.

Figura 1.8: Instruções de processador

A primeira instrução indica ao processador que o valor 19 deve ser armazenado no *registrador 1*. A segunda instrução indica que o valor 11 deve ser armazenado no *registrador 2*. Já a terceira instrução determina a realização da soma dos valores anteriormente armazenados nos *registradores 1 e 2* além de indicar que o resultado seja armazenado no *registrador 3*. Por último, a quarta instrução determina ao processador que o valor do *registrador 3* deve ser exibido na tela.

Não há um padrão universal para o formato das instruções que os processadores podem executar. Consequentemente, as mesmas operações podem ser definidas de formas diferentes em dois processadores distintos. Considere o exemplo fictício a seguir com algumas instruções para dois processadores de tipos diferentes.

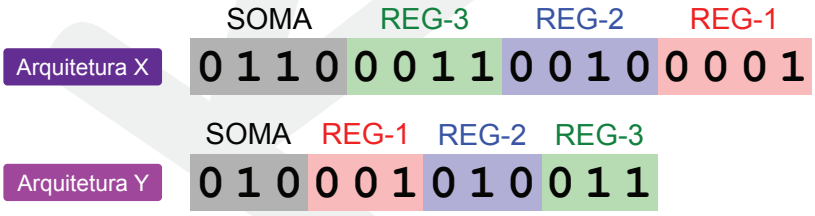

Figura 1.9: Instruções de processadores diferentes

Observe que as duas instruções indicam aos processadores que o valor do registrador 1 deve ser somado ao valor do registrador 2 e o resultado deve ser armazenado no registrador 3. Contudo, as sequências binárias dessas instruções são diferentes porque os processadores são de arquiteturas diferentes.

As instruções que um processador pode executar são definidas pela arquitetura do seu processador. As principais arquiteturas de processadores são:

- x86
- x86_64
- ARM

O que é um Programa?

Os comandos que os processadores dos computadores oferecem são extremamente básicos. Normalmente, são comandos para realizar operações aritméticas como soma, subtração, divisão e multiplicação. Qualquer tarefa mais complexa deve ser resolvida através de uma sequência desses comandos básicos.

Por exemplo, para calcular a média aritmética dos números 10, 15 e 20, devemos obter o valor da somatória desses números através de operações básicas de soma. Depois, podemos utilizar uma operação básica de divisão para dividir o valor dessa somatória por 3 e assim obter o valor 15 que é a média dos números.

Uma sequência de comandos básicos de processador que resolve uma determinada tarefa recebe o nome de **programa**. Os programas são armazenados em arquivos comumente chamados de **executáveis**.

Na prática, os programas são formados por instruções que dependem da arquitetura do processador e do sistema operacional. Consequentemente, um mesmo programa pode não funcionar em sistemas operacionais diferentes ou em processadores de arquiteturas diferentes.

Linguagem de Máquina

Como vimos anteriormente, o formato dos comandos que um computador pode executar depende basicamente da arquitetura do seu processador. Como existem diversas arquiteturas diferentes, um mesmo comando pode funcionar em um computador e não funcionar em outro. O formato dos comandos que funcionam em um determinado processador define a **linguagem de máquina** ou **código de máquina** desse processador.

Comandos definidos em linguagem de máquina são perfeitos para os computadores processarem. Por outro lado, eles são extremamente complexos para as pessoas entenderem. Podemos verificar essa complexidade, observando o trecho de comandos em linguagem de máquina da figura abaixo:

```
00000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000100000000000100000000
00000000000011100001111110111010000111000000000101101000000100
11100110100100001101110000000000101001100110011010010000101010101
00011010000110100101110011001000000111000001110010011011110 1100
11101110010011000010110110100100000011000110110000101101110011 0
11100110111101110100001000000011000100110010100100000011100100 11
```

Figura 1.10: Trecho do programa Chrome em binário

Linguagem de Programação

Devido a complexidade, escrever um programa em linguagem de máquina é inviável. Para tornar viável o desenvolvimento de programas, existem as **linguagens de programação** que tentam se aproximar das linguagens humanas. Confira um trecho de código escrito com a linguagem de pro-

gramação Java:

```
while (lineMeasurer.getPosition() < paragraphEnd) {
  TextLayout layout = lineMeasurer.nextLayout(formatWidth);
  drawPosY += layout.getAscent();
  float drawPosX;

  if (layout.isLeftToRight()) {
    drawPosX = 0;
  } else {
    drawPosX = formatWidth - layout.getAdvance();
  }
}
```

Código Java 1.1: Exemplo de código em Java

Por enquanto, você não precisa se preocupar em entender o que está escrito no código acima. Apenas, observe que um programa escrito em linguagem de programação é bem mais fácil de ser lido do que um programa escrito em linguagem de máquina.

Mais Sobre

A maioria das linguagens de programação são **case sensitive**. Isso significa que elas diferenciam as letras maiúsculas das minúsculas. Portanto, ao escrever o código de um programa, devemos tomar cuidado para não trocar uma letra maiúscula por uma letra minúscula ou vice-versa.

Compilador

Vimos que os computadores são capazes de processar o código escrito em linguagem de máquina. Também vimos que é inviável desenvolver um programa em linguagem de máquina. Por isso, existem as linguagens de programação. Daí surge uma pergunta: se os computadores entendem apenas comandos em linguagem de máquina, como eles podem executar código escrito em linguagem de programação?

Na verdade, os computadores não executam código escrito em linguagem de programação. Esse código que é denominado **código fonte** deve ser traduzido para código em linguagem de máquina. Essa tradução é realizada por programas especiais chamados **compiladores**.

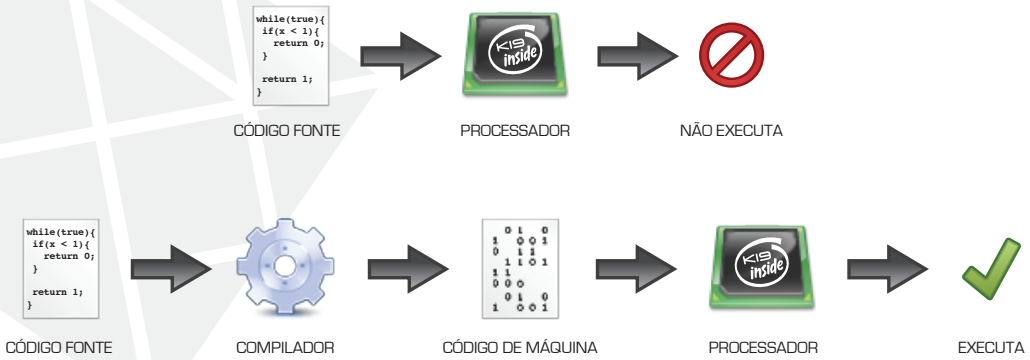

Figura 1.11: Processo de compilação e execução de um programa

Máquinas Virtuais

Como vimos anteriormente, o código fonte de um programa deve ser compilado para que esse programa possa ser executado por um computador. Além disso, vimos que os compiladores geram executáveis específicos para um determinado sistema operacional e uma determinada arquitetura de processador. Qual é o impacto disso para quem desenvolve sistemas para múltiplas plataformas?

A empresa que deseja ter uma aplicação disponível para diversos sistemas operacionais (Windows, Linux, Mac OS X, etc) e arquiteturas de processadores (Intel, ARM, PowerPC, etc) deverá desenvolver e manter um código fonte para cada plataforma (sistema operacional + arquitetura de processador). Consequentemente, os custos dessa empresa seriam altos.

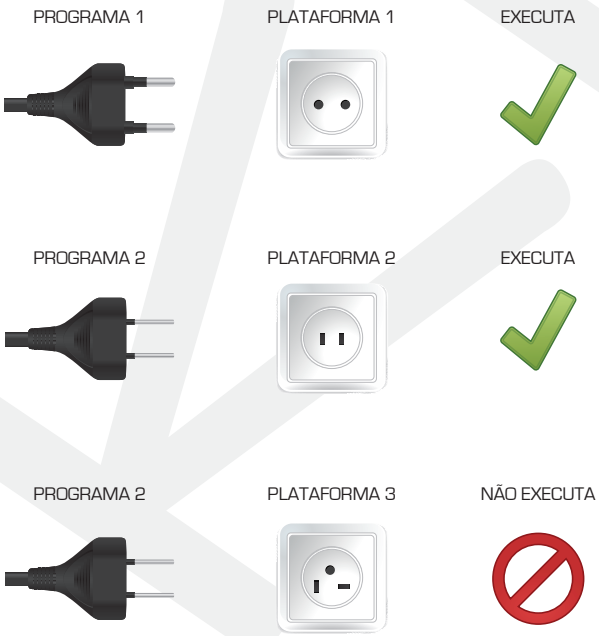

Figura 1.12: Ilustração mostrando que cada plataforma necessita de um executável específico.

Para diminuir os custos e aumentar a produtividade, podemos utilizar **máquinas virtuais**. Em um ambiente que utiliza máquina virtual, quando o código fonte é compilado, ele é traduzido para um código escrito na linguagem da máquina virtual. A linguagem da máquina virtual também pode ser considerada uma linguagem de máquina. Na execução, a máquina virtual traduz os comandos em linguagem de máquina virtual para comandos em linguagem de máquina correspondente à plataforma utilizada.

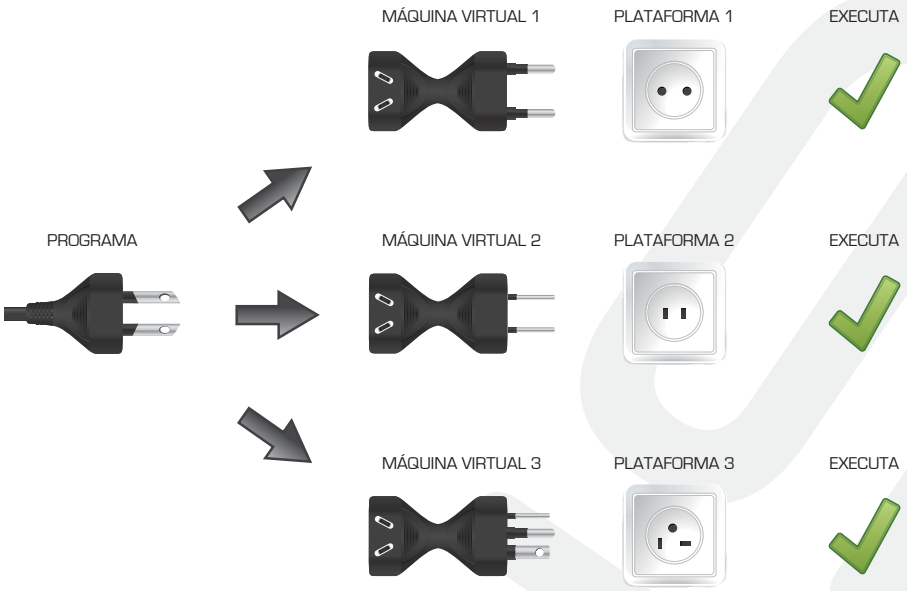

Figura 1.13: Ilustração do funcionamento da máquina virtual.

Tudo parece estar perfeito agora, porém, olhando atentamente a figura acima, percebemos que existe a necessidade de uma máquina virtual para cada plataforma. Alguém poderia dizer que, de fato, não resolvemos o problema.

A diferença é que implementar a máquina virtual não é tarefa dos programadores que desenvolvem as aplicações que serão executados nas máquinas virtuais. A implementação da máquina virtual é responsabilidade de terceiros que geralmente são empresas bem conceituadas ou projetos de código aberto que envolvem programadores do mundo inteiro. Os maiores exemplos são a Oracle JVM (Java Virtual Machine), OpenJDK JVM, Microsoft CLR (Common Language Runtime) e Mono CLR.

A máquina virtual não funciona apenas como um mero adaptador. Ela normalmente traz recursos como o gerenciamento de memória, otimização do código em tempo de execução entre outros.

Hello World em Java

Vamos escrever o nosso primeiro programa para entendermos como funciona o processo de escrita de código fonte, compilação e execução de um programa.

Importante
Antes de compilar e executar um programa escrito em Java, é necessário que você tenha instalado e configurado em seu computador o JDK (Java Development Kit). Consulte o artigo da K19, http://www.k19.com.br/artigos/instalando-o-jdk-java-development-kit/.

Dentro de um editor de texto, escreva o seguinte código e salve o arquivo com o nome **HelloWorld.java**.

```
1 class HelloWorld {
2   public static void main(String[] args) {
3     System.out.println("Hello World");
4   }
5 }
```

Código Java 1.2: HelloWorld.java

Em seguida abra um terminal ou, no caso do Windows, o Prompt de Comando e entre na pasta em que você salvou o arquivo **HelloWorld.java**. Feito isso, digite o seguinte comando no terminal:

```
k19$ javac HelloWorld.java
```

Terminal 1.1: Compilando o arquivo HelloWorld.java

Esse comando compilará o arquivo **HelloWorld.java**. O programa **javac** é o compilador do Java. Após compilarmos o arquivo **HelloWorld.java**, nosso programa já estará pronto para ser executado. Porém, antes de executá-lo, digite no terminal o comando **ls** ou o comando **dir** no Prompt de Comando. Um arquivo chamado **HelloWorld.class** deverá aparecer na listagem de arquivos. Esse arquivo contém o código em linguagem de máquina virtual Java.

```
k19$ ls
HelloWorld.class    HelloWorld.java
```

Terminal 1.2: Listagem do diretório

Agora vamos executar o nosso programa através do comando **java**:

```
k19$ java HelloWorld
Hello World
```

Terminal 1.3: Executando o programa HelloWorld

Para executar o conteúdo do arquivo **HelloWorld.class**, a extensão **.class** não deve ser utilizada. Seguindo os passos acima, você terá um resultado semelhante ao mostrado abaixo:

```
k19$ javac HelloWorld.java
k19$ ls
HelloWorld.class    HelloWorld.java
k19$ java HelloWorld
Hello World
```

Terminal 1.4: Compilação e execução do programa HelloWorld

Hello World em C#

Agora, vamos utilizar outra linguagem de programação para criar o programa semelhante ao visto anteriormente.

Importante
Para compilar um programa escrito em C# é necessário ter o .NET Framework instalado em seu computador. As versões mais recentes do sistema operacional Windows já vêm com o framework instalado.

Se você utiliza os sistemas operacionais Linux ou Mac OS X, pode compilar e executar programas em C# utilizando a plataforma Mono (http://www.mono-project.com/).

Dentro de um editor de texto, escreva o seguinte código e salve o arquivo com o nome **HelloWorld.cs**.

```
1 class HelloWorld
2 {
3   static void Main()
4   {
5     System.Console.WriteLine("Hello World");
6   }
7 }
```

Código C# 1.1: HelloWorld.cs

Em seguida abra o Prompt de Comando do Windows e entre na pasta em que você salvou o arquivo **HelloWorld.cs**. Feito isso, digite o seguinte comando no Prompt de Comando:

```
C:\Users\K19\Desktop\logica-de-programacao>csc HelloWorld.cs
```

Terminal 1.5: Compilando o programa HelloWorld

Esse comando compilará o arquivo **HelloWorld.cs**. O programa **csc** é o compilador do C#. Após compilarmos o arquivo **HelloWorld.cs**, o programa estará pronto para ser executado. Porém, antes de executá-lo, digite no Prompt de Comando o comando **dir**. Um arquivo chamado **HelloWorld.exe** deverá aparecer na listagem de arquivos. Esse arquivo é o executável gerado pelo compilador do C#.

```
C:\Users\K19\Desktop\logica-de-programacao>dir
 O volume na unidade C não tem nome.
 O Número de Série do Volume é 40EF-8653

 Pasta de C:\Users\K19\Desktop\logica-de-programacao

02/03/2013  21:07    <DIR>          .
02/03/2013  21:07    <DIR>          ..
02/03/2013  20:58                90 HelloWorld.cs
02/03/2013  21:07             3.584 HelloWorld.exe
               2 arquivo(s)          3.674 bytes
               2 pasta(s)   22.508.589.056 bytes disponíveis
```

Terminal 1.6: Listagem do diretório

Agora vamos executar o nosso programa:

```
C:\Users\K19\Desktop\logica-de-programacao>HelloWorld.exe
Hello World
```

Terminal 1.7: Executando o programa HelloWorld

Seguindo os passos acima, você terá um resultado semelhante ao mostrado abaixo:

```
Microsoft Windows [versão 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\K19>cd Desktop\logica-de-programacao

C:\Users\K19\Desktop\logica-de-programacao>csc HelloWorld.cs
Microsoft (R) Visual C# Compiler versão 4.0.30319.17929
 para Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. Todos os direitos reservados.
```

```
C:\Users\K19\Desktop\logica-de-programacao>dir
 O volume na unidade C não tem nome.
 O Número de Série do Volume é 40EF-8653

 Pasta de C:\Users\K19\Desktop\logica-de-programacao

03/04/2013  16:50    <DIR>          .
03/04/2013  16:50    <DIR>          ..
02/04/2013  20:58                90 HelloWorld.cs
03/04/2013  16:50             3.584 HelloWorld.exe
               2 arquivo(s)          3.674 bytes
               2 pasta(s)   22.362.529.792 bytes disponíveis

C:\Users\K19\Desktop\logica-de-programacao>HelloWorld.exe
Hello World

C:\Users\K19\Desktop\logica-de-programacao>
```

Terminal 1.8: Compilação e execução do programa HelloWorld

Método Main

Como vimos anteriormente, um programa é basicamente uma sequência de instruções. As instruções de um programa escrito em Java devem ser definidas dentro do método **main**.

```
1 class Programa {
2   public static void main(String[] args) {
3     PRIMEIRA INSTRUÇÃO
4     SEGUNDA INSTRUÇÃO
5     TERCEIRA INSTRUÇÃO
6     ...
7   }
8 }
```

Código Java 1.3: Método main

Podemos dizer que o "ponto de partida" de um programa em Java é a primeira instrução do método **main**. As demais instruções são executas na mesma ordem que estão definidas no código. Eventualmente, durante a execução das instruções, algum erro pode ocorrer e interromper o fluxo do processamento.

Analogamente, as instruções de um programa escrito em C# também devem ser definidas dentro do método **Main**. Contudo, a estrutura da linguagem Java e um pouco diferente da estrutura da linguagem C#.

```
1  class Programa
2  {
3    static void Main()
4    {
5      PRIMEIRA INSTRUÇÃO
6      SEGUNDA INSTRUÇÃO
7      TERCEIRA INSTRUÇÃO
8      ...
9    }
10 }
```

Código C# 1.2: Método Main

Exercícios de Fixação Com Java

1 Abra um terminal e crie uma pasta com o seu nome. Você deve salvar os seus exercícios nessa pasta.

```
K19$ mkdir rafael

K19$ cd rafael

K19/rafael$
```

Terminal 1.9: Criando a pasta de exercícios

Mais Sobre

Nos exercícios com Java, vamos assumir a utilização de um sistema operacional da família Unix. Em sistemas dessa família, o comando **mkdir (make directory)** é utilizado para criar pastas no terminal, o comando **cd (change directory)** é utilizado para trocar a pasta atual do terminal e o comando **ls (list)** é utilizado para listar os arquivos e diretórios da pasta atual do terminal.

Se você estiver utilizando o sistema operacional Windows, os comandos correspondentes ao **mkdir** e **ls** são **md** e **dir** respectivamente. O comando **cd** possui a mesma função em ambiente Unix ou Windows.

```
C:\Users\K19> md rafael

C:\Users\K19> cd rafael

C:\Users\K19\rafael>
```

Terminal 1.10: Criando a pasta de exercícios

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao1.zip

2 Dentro da sua pasta de exercícios, crie uma pasta para os arquivos desenvolvidos nesse capítulo chamada **introducao**.

```
K19/rafael$ mkdir introducao

K19/rafael$ cd introducao

K19/rafael/introducao$
```

Terminal 1.11: Criando a pasta dos exercícios desse capítulo no Linux

```
C:\Users\K19\rafael> md introducao

C:\Users\K19\rafael> cd introducao

C:\Users\K19\rafael\introducao>
```

Terminal 1.12: Criando a pasta dos exercícios desse capítulo no Windows

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao2.zip

3 Utilize um editor de texto e implemente um programa utilizando a linguagem programação Java. Crie um arquivo chamado **HelloWorld.java** na pasta **introducao**.

```
1 class HelloWorld {
2   public static void main(String[] args) {
3     System.out.println("Hello World");
4   }
5 }
```

Código Java 1.4: HelloWorld.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao3.zip

4 Através do terminal, entre na pasta **introducao**; compile o arquivo **HelloWorld.java**; execute o programa.

```
K19/rafael/introducao$ ls
HelloWorld.java

K19/rafael/introducao$ javac HelloWorld.java

K19/rafael/introducao$ ls
HelloWorld.class HelloWorld.java

K19/rafael/introducao$ java HelloWorld
Hello World
```

Terminal 1.13: Compilando e Executando

Exibindo mensagens

Geralmente, as linguagens de programação possuem comandos para exibir mensagens para os usuários. Veja a seguir, a instrução da linguagem Java que permite exibir mensagens.

```
1 System.out.println("MENSAGEM");
```

Agora, na linguagem C#, o comando para exibir mensagens é:

```
1 System.Console.WriteLine("MENSAGEM");
```

Podemos utilizar as chamadas "sequências de escape" para indicar quebras de linhas ou tabulações nas mensagens. Uma quebra de linha é indicada com a sequência de escape "**\n**". Uma tabulação com a sequência de escape "**\t**".

```
1 System.out.println("Linha1\nLinha2\nLinha3");
2 System.out.println("\tTexto Tabulado");
```

```
1 System.Console.WriteLine("Linha1\nLinha2\nLinha3");
2 System.Console.WriteLine("\tTexto Tabulado");
```

Os comandos **System.out.println** e **System.Console.WriteLine** adicionam uma quebra de linha no final da mensagem exibida. Para exibir mensagens sem quebra de linha, podemos utilizar os comandos **System.out.print** e **System.Console.Write** nas linguagens Java e C# respectivamente.

```
System.out.print("MENSAGEM SEM QUEBRA DE LINHA");
```

```
System.Console.Write("MENSAGEM SEM QUEBRA DE LINHA");
```

Comentários

Podemos acrescentar comentários no código fonte. Geralmente, eles são utilizados para explicar a lógica do programa. Os compiladores ignoram os comentários inseridos no código fonte. Portanto, no código de máquina gerado pela compilação do código fonte, os comentários não são inseridos.

Em Java ou C# para comentar uma linha, devemos utilizar a marcação **//**.

```
System.out.println("K19");
// comentário de linha
System.out.println("Rafael Cosentino");
```

```
System.Console.WriteLine("K19");
// comentário de linha
System.Console.WriteLine("Rafael Cosentino");
```

Em Java e C#, também é possível comentar um bloco com os marcadores **/*** e ***/**.

```
System.out.println("K19");
/* comentário de bloco
todo esse trecho
está comentado */
System.out.println("Rafael Cosentino");
```

```
System.Console.WriteLine("K19");
/* comentário de bloco
todo esse trecho
está comentado */
System.Console.WriteLine("Rafael Cosentino");
```

Indentação

A organização do código fonte é fundamental para o entendimento da lógica de um programa. Cada linguagem de programação possui os seus próprios padrões de organização. Observe a organização padrão do código fonte nas linguagens Java e C#.

```
class Programa {
  public static void main(String[] args) {
    // instruções
  }
}
```

Código Java 1.10: Programa.java

```
class Programa
{
  static void Main()
```

```
4   {
5     // instruções
6   }
7 }
```

Código C# 1.8: Programa.cs

Para destacar a hierarquia dos blocos, o conteúdo de cada bloco deve ser indentado com tabulações ou espaços. Um código corretamente indentado é mais fácil de ler. Consequentemente, a manutenção das aplicações torna-se mais simples.

Engenharia Reversa (Conteúdo Extra)

Provavelmente, você já desmontou um brinquedo ou algum aparelho eletrônico para tentar descobrir como ele funciona. Ao fazer isso, mesmo sem saber, você praticou engenharia reversa.

Muitas empresas praticam engenharia reversa para entender o funcionamento dos produtos dos concorrentes. Países também utilizam esse tipo de abordagem para avaliar a capacidade militar dos outros países.

A engenharia reversa também é aplicada na área de software. As instruções do código de máquina de um programa podem ser traduzidas para alguma linguagem de programação através de programas especiais que são chamados de **decompiladores**.

Normalmente, o código em linguagem de programação gerado a partir da decompilação do código de máquina de um programa não é fácil de entender. Geralmente, é possível, apesar de normalmente ser muito difícil, modificar o funcionamento de um programa para qualquer que seja o propósito utilizando a abordagem da engenharia reversa.

Ofuscadores (Conteúdo Extra)

Para dificultar o processo de engenharia reversa, podemos utilizar ferramentas que modificam o código fonte ou o código compilado com o intuito de prejudicar o processo de decompilação. Essas ferramentas são chamadas de **Ofuscadores**.

Na maior parte dos casos, a utilização de ofuscadores torna inviável ou muito custosa a aplicação de engenharia reversa com intuito de "copiar" ou "piratear" um software.

Exercícios de Fixação Com Java

5 Crie um arquivo na pasta **introducao** chamado **Triangulo.java** com o seguinte conteúdo.

```
1 class Triangulo {
2   public static void main(String[] args) {
3     System.out.println("*");
4     System.out.println("**");
5     System.out.println("***");
6     System.out.println("****");
```

```
7       System.out.println("*****");
8     }
9   }
```

Código Java 1.11: Triangulo.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao5.zip

6 Através do terminal, entre na pasta **introducao**; compile o arquivo **Triangulo.java**; execute o programa.

```
K19/rafael/introducao$ javac Triangulo.java

K19/rafael/introducao$ java Triangulo
*
**
***
****
*****
```

Terminal 1.14: Compilando e Executando

7 Crie um arquivo na pasta **introducao** chamado **TrianguloComBarraN.java** com o seguinte conteúdo.

```
1  class TrianguloComBarraN {
2    public static void main(String[] args) {
3      System.out.println("*\n**\n***\n****\n*****");
4    }
5  }
```

Código Java 1.12: TrianguloComBarraN.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao7.zip

8 Através do terminal, entre na pasta **introducao**; compile o arquivo **TrianguloComBarraN.java**; execute o programa.

```
K19/rafael/introducao$ javac TrianguloComBarraN.java

K19/rafael/introducao$ java TrianguloComBarraN
*
**
***
****
*****
```

Terminal 1.15: Compilando e Executando

Erro: Não Fechar os Blocos

Um erro de compilação comum em Java ou C# é esquecer de fechar os blocos com chave. Observe, nos exemplos a seguir, esse erro ocorrendo.

```
1  class Programa {
2    public static void main(String[] args) {
```

```
3     // instruções
4 }
```

Código Java 1.13: Programa.java

```
1 class Programa {
2   public static void main(String[] args) {
3     // instruções
```

Código Java 1.14: Programa.java

```
1 class Programa
2 {
3   static void Main()
4   {
5     // instruções
6 }
```

Código C# 1.9: Programa.cs

```
1 class Programa
2 {
3   static void Main()
4   {
5     // instruções
```

Código C# 1.10: Programa.cs

Veja a mensagem que o compilador do Java exibe quando um bloco não é fechado corretamente.

```
Programa.java:2: error: reached end of file while parsing
  public static void main(String[]  args) {
                                           ^
1 error
```

Terminal 1.16: Erro de compilação

Analogamente, veja a mensagem que o compilador do C# exibe quando um bloco não é fechado corretamente.

```
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.

Programa.cs(4,3): error CS1513: } expected
Programa.cs(4,3): error CS1513: } expected
```

Terminal 1.17: Erro de compilação

Erro: Trocar Maiúsculas e Minúsculas

Um erro de compilação comum em Java ou C# é utilizar letras maiúsculas onde deveriam ser utilizadas letras minúsculas ou vice-versa. Nos exemplos a seguir, o identificador **System** foi escrito com "s", porém o correto é com "S".

```
1 class HelloWorld {
2   public static void main(String[] args) {
3     system.out.println("Hello World");
4   }
```

```
5 }
```

Código Java 1.15: HelloWorld.java

```
1 class HelloWorld
2 {
3   static void Main()
4   {
5     system.Console.WriteLine("Hello World");
6   }
7 }
```

Código C# 1.11: HelloWorld.cs

Veja as mensagens de erro do compilador do Java e do C# respectivamente.

```
HelloWorld.java:3: error: package system does not exist
    system.out.println("Hello World");
           ^
1 error
```

Terminal 1.18: Erro de Compilação

```
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.

HelloWorld.cs(5,3): error CS0103: The name 'system' does not exist in the current context
```

Terminal 1.19: Erro de Compilação

Erro: Esquecer o Ponto e Vírgula

Para encerrar uma instrução, devemos utilizar o caractere "**;**". Não inserir esse caractere no final das instruções gera erro de compilação. Veja, nos dois exemplos abaixo, esse erro ocorrendo.

```
1 class HelloWorld {
2   public static void main(String[] args) {
3     System.out.println("Hello World")
4   }
5 }
```

Código Java 1.16: HelloWorld.java

```
1 class HelloWorld
2 {
3   static void Main()
4   {
5     System.Console.WriteLine("Hello World")
6   }
7 }
```

Código C# 1.12: HelloWorld.cs

Veja as mensagens de erro do compilador do Java e do C# respectivamente.

```
HelloWorld.java:3: error: ';' expected
    System.out.println("Hello World")
                                     ^
1 error
```

Terminal 1.20: Erro de Compilação

```
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.

HelloWorld.cs(5,42): error CS1002: ; expected
```

Terminal 1.21: Erro de Compilação

Erro: Esquecer o Main

Todo programa deve ter um "ponto de partida". Em Java ou C#, todo programa precisa do método **main**. Nessas duas linguagens, se você esquecer de definir o método main obterá um erro. Em Java, o erro será de execução. Já em C#, o erro será de compilação. Veja o exemplo a seguir.

```
1 class HelloWorld {
2   public static void Main(String[] args) {
3     System.out.println("Hello World");
4   }
5 }
```

Código Java 1.17: HelloWorld.java

Observe que no código Java acima, o método main foi definido com letra maiúscula. Contudo, no Java, o correto é com minúscula. Ao compilar o código, nenhum erro ocorre. Mas, ao executar, o seguinte erro é exibido.

```
K19/rafael$ javac HelloWorld.java
K19/rafael$ java HelloWorld
Error: Main method not found in class HelloWorld, please define the main method as:
   public static void main(String[] args)
```

Terminal 1.22: Erro de Execução

Agora, considere o exemplo a seguir em C#.

```
1 class HelloWorld
2 {
3   static void main()
4   {
5     System.Console.WriteLine("Hello World");
6   }
7 }
```

Código C# 1.13: HelloWorld.cs

Nesse caso, o método main foi definido com letra minúscula. Contudo, no C#, o correto é com maiúscula. Ao compilar o código, um erro semelhante ao exibido abaixo ocorrerá.

```
Microsoft (R) Visual C# Compiler version 4.0.30319.17929
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.

error CS5001: Program 'c:\Users\cosen\Desktop\Rafael\HelloWorld.exe' does
  not contain a static 'Main' method suitable for an entry point
```

Terminal 1.23: Erro de Compilação

Exercícios de Fixação Com C#

9 Utilize um editor de texto e implemente um programa utilizando a linguagem programação C#. Crie um arquivo chamado **HelloWorld.cs** na pasta **introducao**.

```
1 class HelloWorld
2 {
3   static void Main()
4   {
5     System.Console.WriteLine("Hello World");
6   }
7 }
```

Código C# 1.14: HelloWorld.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao9.zip

10 Através do terminal, entre na pasta **introducao**; compile o arquivo **HelloWorld.cs**; execute o programa.

```
C:\Users\K19\rafael\introducao> csc HelloWorld.cs

C:\Users\K19\rafael\introducao> HelloWorld.exe
Hello World
```

Terminal 1.24: Compilando e Executando

11 Crie um arquivo na pasta **introducao** chamado **Triangulo.cs** com o seguinte conteúdo.

```
1  class Triangulo
2  {
3    static void Main()
4    {
5      System.Console.WriteLine("*");
6      System.Console.WriteLine("**");
7      System.Console.WriteLine("***");
8      System.Console.WriteLine("****");
9      System.Console.WriteLine("*****");
10   }
11 }
```

Código C# 1.15: Triangulo.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao11.zip

12 Através do terminal, entre na pasta **introducao**; compile o arquivo **Triangulo.cs**; execute o programa.

```
C:\Users\K19\rafael\introducao> csc Triangulo.cs

C:\Users\K19\rafael\introducao> Triangulo.exe
*
**
***
****
*****
```

Terminal 1.25: Compilando e Executando

13 Crie um arquivo na pasta **introducao** chamado **TrianguloComBarraN.cs** com o seguinte conteúdo.

```
class TrianguloComBarraN
{
  static void Main()
  {
    System.Console.WriteLine("*\n**\n***\n****\n*****");
  }
}
```

Código Java 1.18: TrianguloComBarraN.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-fixacao13.zip

14 Através do terminal, entre na pasta **introducao**; compile o arquivo **TrianguloComBarraN.cs**; execute o programa.

```
C:\Users\K19\rafael\introducao> csc TrianguloComBarraN.cs

C:\Users\K19\rafael\introducao> TrianguloComBarraN.exe
*
**
***
****
*****
```

Terminal 1.26: Compilando e Executando

Exercícios Complementares

1 Utilize a linguagem Java para implementar um programa que mostre a seguinte saída.

```
Hello World 1
Hello World 2
```

2 Utilize a linguagem C# para implementar um programa que mostre a seguinte saída.

```
Hello World 1
Hello World 2
```

3 Utilize a linguagem Java para implementar um programa que mostre a sua frase preferida.

4. Utilize a linguagem C# para implementar um programa que mostre a sua frase preferida.

5. Utilize a linguagem Java para implementar um programa que mostre a seguinte saída.

```
#    #   #    #####
#   #   ##   #     #
#  #   # #   #     #
###      #    ######
#  #     #         #
#   #    #   #     #
#    # #####  #####
```

6. Utilize a linguagem C# para implementar um programa que mostre a seguinte saída.

```
#    #   #    #####
#   #   ##   #     #
#  #   # #   #     #
###      #    ######
#  #     #         #
#   #    #   #     #
#    # #####  #####
```

Desafios

1. Crie um programa em Java ou C# para exibir o seu nome de forma semelhante ao mostrado abaixo.

```
______        __             _      _____                        _   _
| ___ \      / _|           | |    /  __ \                      | | (_)
| |_/ /__ _ | |_ __ _   ___ | |    | /  \/ ___  ___  ___ _ __  | |_ _ _ __    ___
|    // _` |  _/ _` |/ _ \ | |    | |    / _ \/ __|/ _ \ '_ \| __| | '_ \ / _ \
| |\ \ (_| | || (_| |  __/ |    | \__/\ (_) \__ \  __/ | | | |_| | | | | (_) |
\_| \_\__,_|_| \__,_|\___|_|     \____/\___/|___/\___|_| |_|\__|_|_| |_|\___/
```

2. Pesquise na internet sobre ASCII Art. Crie programas para exibir imagens em ASCII Art.

Resumo do Capítulo

1. Os principais elementos de um computador são: **CPU**, **memória RAM** e **disco rígido**.

2. A **CPU** é o responsável por **executar** as **instruções** dos programas.

3. Os dados dos programas abertos no computador são armazenados na **memória RAM**.

4. Os arquivos dos usuários e dados dos programas instalados no computador são armazenados no **disco rígido**.

5 Os dados armazenados na **memória RAM** são perdidos se o computador for desligado.

6 Os dados armazenados no **disco rígido não** são perdidos se o computador for desligado.

7 Os computadores se comunicam com outros computadores ou com dispositivos periféricos através das suas **portas físicas**.

8 Os dados manipulados por um computador são definidos em **formato binário**.

9 As principais unidades de medida para dados binários são: **Bit**, **Byte(B)**, **Quilobyte(kB)**, **Megabyte(MB)**, **Gigabyte(GB)** e **Terabyte(TB)**.

10 Os **comandos** que um processador pode executar são definidos pela sua **arquitetura**.

11 Processadores de **arquiteturas diferentes** entendem **comandos diferentes**.

12 Um **programa** é uma **sequência de instruções** que resolve uma determinada tarefa.

13 As **linguagens de programação** são mais fáceis para pessoas entenderem do que as **linguagens de máquina**.

14 Os **programas** são definidos em **linguagem de programação**.

15 As principais linguagens de programação são **case sensitive**.

16 Os **compiladores** traduzem o **código fonte** de um programa para **código de máquina**.

17 As **máquinas virtuais** permitem a criação de programas **portáveis**.

18 Todo programa necessita de um "**ponto de partida**". O ponto de partida dos programas escritos em Java ou C# é a primeira instrução do método **main**.

19 No código fonte de um programa em Java ou C#, **comentários** são inseridos com os marcadores **//**, **/*** e ***/**.

20 A **indentação** melhora a legibilidade do código fonte.

21 Código escrito em linguagem Java deve ser armazenado em arquivos **.java**. Código escrito em linguagem C# deve ser armazenado em arquivos **.cs**

22 Os comandos **System.out.println** e **System.Console.WriteLine** são utilizados para exibir mensagens com quebra de linha nas linguagens Java e C# respectivamente.

23 Os comandos **System.out.print** e **System.Console.Write** são utilizados para exibir mensagens **sem** quebra de linha nas linguagens Java e C# respectivamente.

Prova

1 Qual alternativa está correta?

a) A principal função dos processadores é armazenar dados.

b) Os dados armazenados no disco rígido são perdidos quando o computador é desligado.

c) O acesso aos dados armazenados na memória RAM é mais rápido do que o acesso aos dados armazenados nos registradores do processador.

d) Geralmente, o espaço de armazenamento da memória RAM é menor do que o espaço de armazenamento do disco rígido.

e) A memória RAM é a principal responsável pela execução das instruções de um programa.

2 Qual alternativa está correta?

a) Um computador não pode transmitir dados para outro computador.

b) A conexão entre os teclados e os computadores é realizada através da porta Ethernet.

c) As portas USB são a única forma de estabelecer a comunicação dos computadores com os dispositivos periféricos.

d) Atualmente, diversos dispositivos podem ser conectados aos computadores através das portas USB.

e) Os mouses atuais são conectados aos computadores através das portas HDMI.

3 Qual é a representação binária do número 19?

a) 00019

b) 10011

c) 10101

d) 11001

e) 01101

4 Quantos Bits ocupa um arquivo de 19 kB?

a) 19000

b) 19

c) 8192

d) 1048576

e) 155648

5 Qual alternativa está errada?

a) x86 e x86_64 são arquiteturas de processador.

b) Os comandos que um processador pode executar dependem da arquitetura desse processador.

c) Processadores de arquiteturas diferentes executam comandos iguais.

d) As instruções que os processadores executam são definidas em binário.

e) Processadores executam código em linguagem de máquina.

6 Qual alternativa está correta?

a) Java e C# são linguagens de programação.

b) Java e C# são linguagens de máquina.

c) Java é uma linguagem de programação e C# é uma linguagem de máquina.

d) C# é uma linguagem de programação e Java é uma linguagem de máquina.

7 Qual é a função dos compiladores?

a) Traduzir código de máquina para código fonte.

b) Executar código de máquina.

c) Executar código fonte.

d) Armazenar os dados do computador.

e) Traduzir código fonte para código de máquina.

8 Qual é a vantagem das linguagens de programação que utilizam máquinas virtuais?

a) A criação de programas "portáveis".

b) Essas linguagens possuem mais comandos.

c) Os programas desenvolvidos com essas linguagens são mais rápidos.

d) A criação de programas específicos para um sistema operacional.

e) Os programas desenvolvidos com essas linguagens consomem menos memória.

9 Qual alternativa possui apenas nomes válidos para arquivos de código fonte Java?

a) "K19.java" e "Treinamentos.Java".

b) "K19.java" e "Treinamentos.java".

c) "K19.JAVA" e "Treinamentos.java".

d) "K19.JAVA" e "Treinamentos.JAVA".

e) "K19.Java" e "Treinamentos.Java".

10 Qual alternativa possui apenas nomes válidos para arquivos de código fonte C#?

a) "K19.cs" e "Treinamentos.cs".

b) "K19.csharp" e "Treinamentos.csharp".

c) "K19.cs" e "Treinamentos.csharp".

d) "K19.CS" e "Treinamentos.cs".

e) "K19.Csharp" e "Treinamentos.Cs".

11 Qual alternativa declara corretamente o método main em Java?

a) public static main(String[] args).

b) public static void Main(String[] args).

c) void main(String[] args).

d) static void Main().

e) public static void main(String[] args).

12 Qual alternativa declara corretamente o método main em C#?

a) public static main(String[] args).

b) public static void main().

c) void Main(String[] args).

d) static void Main().

e) public static void main(String[] args).

13 Qual código pode ser utilizado para exibir a mensagem “K19” em Java?

a) system.out.println("K19").

b) System.out.println("K19").

c) System.Console.WriteLine("K19").

d) system.console.writeLine("K19").

e) print("K19").

14 Qual código pode ser utilizado para exibir a mensagem “K19” em C#?

a) system.out.println("K19").

b) System.out.println("K19").

c) System.Console.WriteLine("K19").

d) system.console.writeLine("K19").

e) print("K19").

15 Quais são os marcadores utilizados para inserir comentários em Java ou C#?

a) “//” e “%”.

b) “#”, “//”, “/*” e “*/”.

c) “//”, “/*” e “*/”.

d) “<!- -” e “- ->”.

e) “%” e “#”.

Minha Pontuação [ ] Pontuação Mínima: [12] Pontuação Máxima: [15]

CAPÍTULO 2

ALGORITMOS

O que é um Algoritmo?

Um **algoritmo** é uma sequência de instruções que resolve uma determinada tarefa. Essas instruções podem ser executadas por um computador ou até mesmo por um ser humano. Um algoritmo pode ser comparado a uma receita de bolo, onde cada passo da preparação do bolo corresponde a uma instrução do algoritmo.

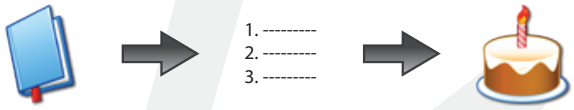

Figura 2.1: Preparação de um bolo seguindo os passos de uma receita (algoritmo)

Normalmente, desenvolver algoritmos eficientes não é uma tarefa simples. No meio acadêmico, diversas técnicas para o desenvolvimento de algoritmos mais eficientes são estudadas pela Ciência da Computação.

Como um algoritmo pode ser representado?

Nós podemos representar um algoritmo da maneira que acharmos melhor, desde que tal representação seja bem estruturada e organizada. Porém, as representações mais utilizadas são a de **Fluxograma** e de **Pseudocódigo**.

Fluxograma

O **fluxograma** é um dos métodos mais utilizados para se representar um algoritmo. Trata-se de uma espécie de diagrama e é utilizado para documentar processos (simples ou complexos). Tal tipo de diagrama ajuda o leitor a visualizar um processo, compreendê-lo mais facilmente e encontrar falhas ou problemas de eficiência.

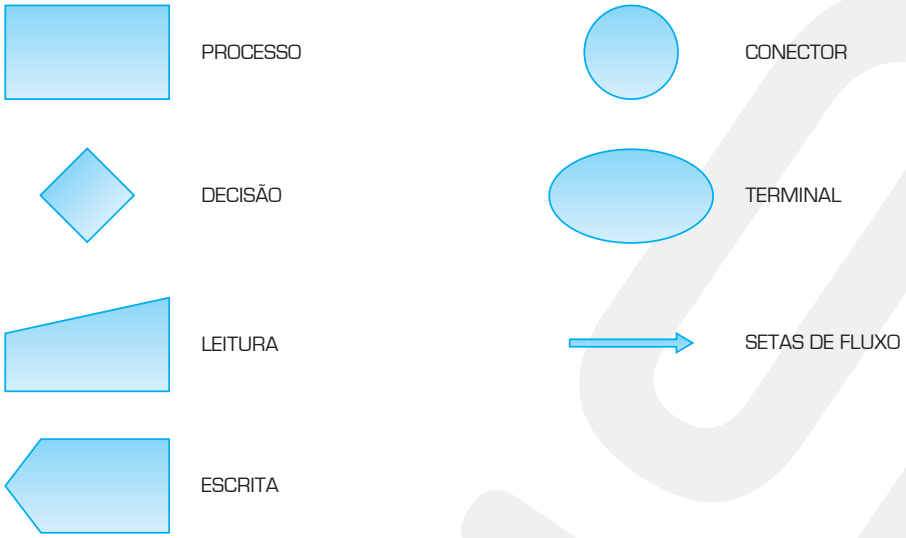

Figura 2.2: Símbolos utilizados em um fluxograma

Vamos supor que seja necessário criar um algoritmo para sacar uma determinada quantia de dinheiro de um caixa eletrônico de um banco. Como ficaria o fluxograma desse algoritmo?

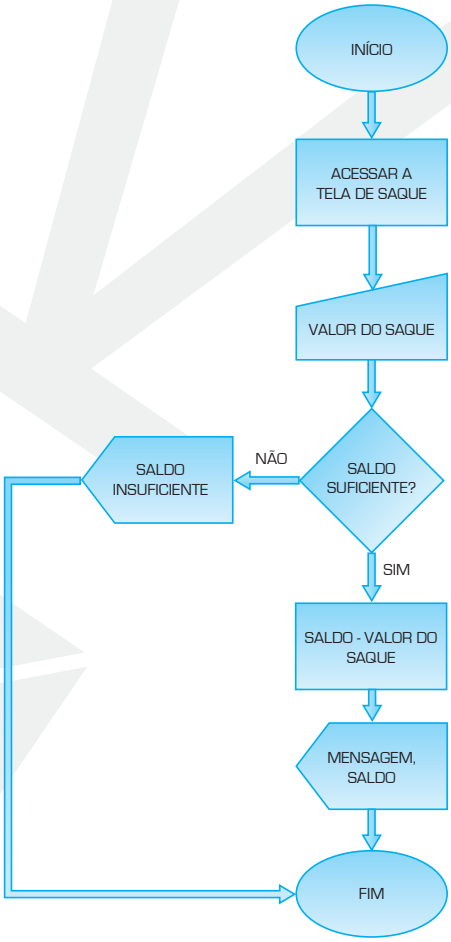

Figura 2.3: Exemplo de fluxograma para a operação de saque em um caixa eletrônico de um banco

Para entender o algoritmo que um fluxograma representa, é necessário conhecer o significado de cada símbolo.

Pseudocódigo

Escrever um algoritmo em **pseudocódigo** é outra forma muito utilizada por autores de livros que tratam de algoritmos, pois dessa forma o leitor não precisa ter o conhecimento prévio de nenhuma linguagem de programação. Nos países cujo idioma principal é o português, muitos se referem ao pseudocódigo como **portugol**. Vamos ver como ficaria o exemplo anterior escrito em pseudocódigo:

```
1 INICIO
2    LER(ValorDoSaque)
3    SE ValorDoSaque > 0 E ValorDoSaque <= Saldo ENTÃO
4       Saldo = Saldo - ValorDoSaque;
5       ESCREVER("Saque efetuado com sucesso. Saldo atual: ", Saldo);
6    SENÃO
7       ESCREVER("Saldo Insuficiente.");
8    FIM SE
9 FIM
```

Pseudocódigo 2.1: Exemplo de pseudocódigo para a operação de saque em um caixa eletrônico.

A representação em pseudocódigo é bem simples e na maioria dos casos é suficiente para se explicar um algoritmo. Existem alguns interpretadores de portugol como o **VisuAlg** e, no caso da língua inglesa, temos algumas linguagens como **Pascal** e **BASIC** cuja sintaxe se assemelha muito com a sintaxe de um pseudocódigo em inglês.

Exercícios de Fixação

1 Escreva, utilizando um fluxograma, um algoritmo para a operação de depósito em um caixa eletrônico de um banco.

2 Escreva, utilizando um fluxograma, um algoritmo para calcular o desconto obtido por um aluno da K19 através do Programa Indicação Premiada (veja as regras no site).

Dica: faça com que o aluno que está indicando receba de início 5% de desconto.

Desafios

1 Escreva, utilizando um fluxograma, um possível algoritmo para o jogo Travessia do Rio disponível online em diversos sites (ex: http://www.aulavaga.com.br/jogos/raciocinio/travessia-do-rio/).

O jogo consiste em atravessar todos os personagens de uma margem à outra do rio seguindo as seguintes regras:

1. Somente o pai, a mãe e o policial sabem pilotar o barco;
2. A mãe não pode ficar sozinha com os filhos;

3. O pai não pode ficar sozinho com as filhas;
4. O prisioneiro não pode ficar com nenhum membro da família sem o policial;
5. O barco pode transportar, no máximo, duas pessoas por vez;
6. Você pode fazer quantas viagens desejar.

CAPÍTULO 3

VARIÁVEIS

O que é uma Variável?

Os dados manipulados por um programa são armazenados em **variáveis**. Normalmente, uma variável é associada a uma posição da memória RAM. Nas variáveis é possível armazenar dados de vários tipos: numéricos, strings (texto), booleanos (verdadeiro ou falso), referências, entre outros.

Toda variável possui um nome (um identificador). Os nomes das variáveis são utilizados para manipular os dados contidos nelas. Como, normalmente, as variáveis são associadas à posições da memória RAM, os identificadores das variáveis funcionam como nomes simbólicos dos endereços da memória RAM.

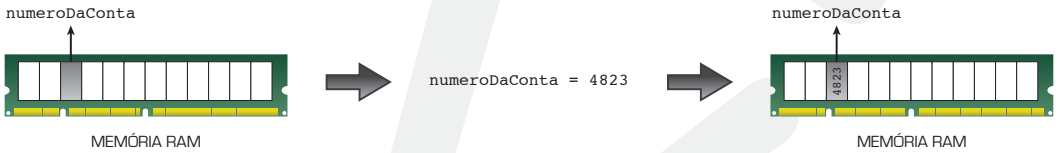

Figura 3.1: Processo de atribuição do valor numérico 4823 à variável numeroDaConta.

Declarando variáveis em Java ou C#

Para criar uma variável em Java ou C#, é necessário declará-la. Nessas duas linguagens de programação, para declarar uma variável é necessário informar o seu tipo e o seu nome (identificador).

```
1 int numeroDaConta;
2 double saldo;
3 boolean contaAtiva;
```

Código Java 3.1: Declaração de variáveis em Java.

```
1 int numeroDaConta;
2 double saldo;
3 bool contaAtiva;
```

Código C# 3.1: Declaração de variáveis em C#.

Mais Sobre

Em Java ou C#, devemos informar, no código fonte, o tipo de dado que uma variável poderá armazenar. Por isso, essas duas linguagens são **estaticamente tipadas**, ou seja, os tipos das variáveis devem ser definidos em tempo de compilação.

Inicialização

Após declararmos uma variável e antes de utilizá-la, devemos inicializá-la para evitarmos um erro de compilação.

```
1 int numeroDaConta;
2 numeroDaConta = 3466;
3
4 boolean contaAtiva = true;
```

Código Java 3.2: Declaração e inicialização de variáveis em Java.

```
1 int numeroDaConta;
2 numeroDaConta = 3466;
3
4 bool contaAtiva = true;
```

Código C# 3.2: Declaração e inicialização de variáveis em C#.

Como podemos observar, a inicialização das variáveis é feita através do operador "=". Note também que podemos, em uma única linha, declarar e inicializar uma variável.

Pare para pensar...
O que aconteceria se o compilador Java ou C# permitisse utilizar uma variável não inicializada?

Um programador C (não C#), responderia essa pergunta facilmente, pois em C é possível utilizar uma variável sem inicializá-la. Quando uma variável é declarada, um espaço na memória ram do computador é reservado. Esse espaço pode ter sido utilizado, anteriormente, por outro programa e pode conter dados não mais utilizados. Dessa forma, se uma variável não inicializada for utilizada, o valor que estava no espaço de memória correspondente a essa variável será utilizado.

Muitos programadores C esquecem de inicializar suas variáveis com os valores adequados. Isso provoca muitos erros de lógica. Em Java e C#, esse problema não existe pois as variáveis devem sempre ser inicializadas antes de serem utilizadas.

Tipos de Básicos

As linguagens Java e C# possuem tipos básicos de variáveis. Esses tipos são os mais utilizados e servem como base para a criação de outros tipos. A seguir, veja os tipos básicos da linguagem Java e da linguagem C#.

Tipo	Descrição	Tamanho ("peso")
byte	Valor inteiro entre -128 e 127 (inclusivo)	1 byte
short	Valor inteiro entre -32.768 e 32.767 (inclusivo)	2 bytes
int	Valor inteiro entre -2.147.483.648 e 2.147.483.647 (inclusivo)	4 bytes
long	Valor inteiro entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807 (inclusivo)	8 bytes

Tipo	Descrição	Tamanho (“peso”)
float	Valor com ponto flutuante entre $1,40129846432481707 \times 10^{-45}$ e $3,40282346638528860 \times 10^{38}$ (positivo ou negativo)	4 bytes
double	Valor com ponto flutuante entre $4,94065645841246544 \times 10^{-324}$ e $1,79769313486231570 \times 10^{308}$ (positivo ou negativo)	8 bytes
boolean	true ou false	1 bit
char	Um único caractere Unicode de 16 bits. Valor inteiro e positivo entre 0 (ou ‘\u0000’) e 65.535 (ou ‘\uffff’)	2 bytes

Tabela 3.1: Tipos de dados básicos em Java.

Tipo	Descrição	Tamanho (“peso”)
sbyte	Valor inteiro entre -128 e 127 (inclusivo)	1 byte
byte	Valor inteiro entre 0 e 255 (inclusivo)	1 byte
short	Valor inteiro entre -32.768 e 32.767 (inclusivo)	2 bytes
ushort	Valor inteiro entre 0 e 65.535 (inclusivo)	2 bytes
int	Valor inteiro entre -2.147.483.648 e 2.147.483.647 (inclusivo)	4 bytes
uint	Valor inteiro entre 0 e 4.294.967.295 (inclusivo)	4 bytes
long	Valor inteiro entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807 (inclusivo)	8 bytes
ulong	Valor inteiro entre 0 e 18.446.744.073.709.551.615 (inclusivo)	8 bytes
float	Valor com ponto flutuante entre $1,40129846432481707 \times 10^{-45}$ e $3,40282346638528860 \times 10^{38}$ (positivo ou negativo)	4 bytes
double	Valor com ponto flutuante entre $4,94065645841246544 \times 10^{-324}$ e $1,79769313486231570 \times 10^{308}$ (positivo ou negativo)	8 bytes
decimal	Valor com ponto flutuante entre $1,0 \times 10^{-28}$ e $7,9 \times 10^{28}$ (positivo ou negativo)	16 bytes
bool	true ou false	1 bit
char	Um único caractere Unicode de 16 bits. Valor inteiro e positivo entre 0 (ou ‘\u0000’) e 65.535 (ou ‘\uffff’)	2 bytes

Tabela 3.2: Tipos de dados básicos em C#.

String

Tanto na linguagem Java quanto na linguagem C#, o tipo **string** é um dos mais importantes e mais utilizados. O tipo string é usado para o armazenamento de texto (sequência de caracteres). Observe, nos exemplos abaixo, que o texto que deve ser armazenado nas variáveis é definido dentro de **aspas duplas**.

```
1 String texto = "K19 Treinamentos";
```

Código Java 3.3: Tipo String em Java

```
1 string texto = "K19 Treinamentos";
```

Código C# 3.3: Tipo string em C#

Os caracteres que podem ser utilizados para formar o conteúdo de uma string são definidos pelo padrão **Unicode**(http://www.unicode.org/). As plataformas Java e .NET utilizam o mapeamento **UTF-16** do Unicode.

O espaço utilizado por uma string depende da quantidade de caracteres que ela possui. Cada caractere ocupa 16 Bits. Portanto, a string "K19 Treinamentos" que possui 16 caracteres (o espaço também deve ser contabilizado) ocupa 256 Bits.

Data e Hora (Conteúdo Extra)

As linguagens Java e C# possuem tipos específicos para armazenar **data e hora**. Em Java, é muito comum, utilizarmos o tipo **Calendar**.

```
1 java.util.Calendar exatamenteAgora = java.util.Calendar.getInstance();
```

Código Java 3.4: Data e Hora Atuais - Calendar

No código acima, a data e hora atuais do computador são armazenadas na variável **exatamente-Agora**. Também podemos definir data e hora específicas.

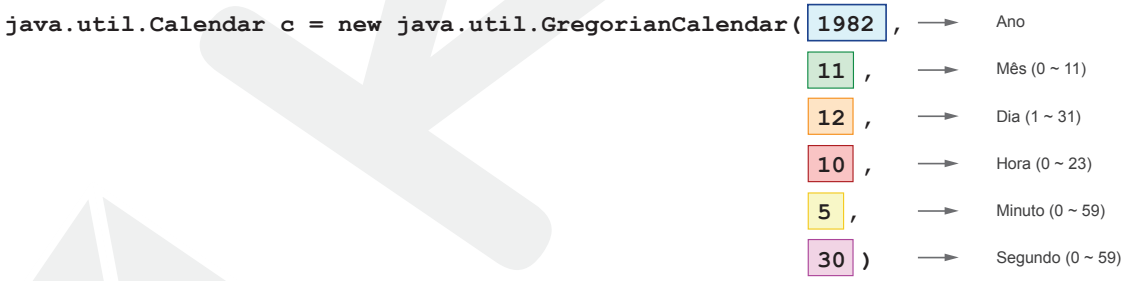

Figura 3.2: Data e Hora Específicas - Calendar

No exemplo acima, o primeiro parâmetro define o ano; o segundo o mês; o terceiro o dia; o quarto a hora; o quinto os minutos; e o sexto os segundos. O mês é definido da seguinte forma: 0 é janeiro, 1 é fevereiro, 2 março e assim por diante. Dessa forma, a data "12 de dezembro de 1982" e hora "10:05:30" foram armazenadas na variável **c**.

Em C#, normalmente, utilizamos o tipo **DateTime**.

```
1 System.DateTime exatamenteAgora = System.DateTime.Now;
```

Código C# 3.4: Data e Hora Atuais - DateTime

No código acima, a data e hora atuais do computador são armazenadas na variável chamada **exatamenteAgora**. Também podemos definir data e hora específicas. Veja os exemplos a seguir.

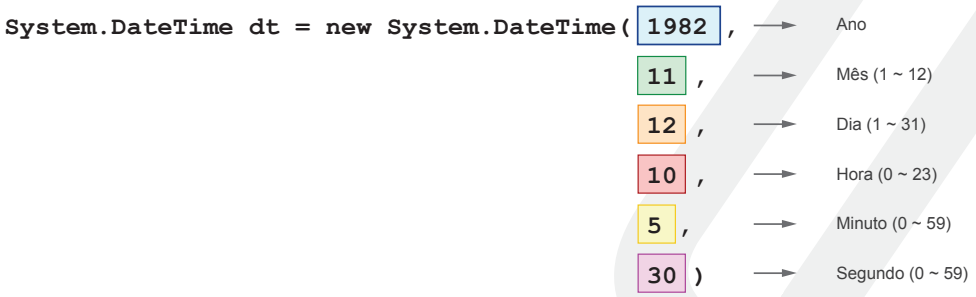

Figura 3.3: Data e Hora Específicas - DateTime

```
1 System.DateTime dt = new System.DateTime(1982, 12, 12, 10, 5, 30);
```

Código C# 3.5: Data e Hora Específicas - DateTime

No exemplo acima, o primeiro parâmetro define o ano; o segundo o mês; o terceiro o dia; o quarto a hora; o quinto os minutos; e o sexto os segundos. Dessa forma, a data "12 de dezembro de 1982" e hora "10:05:30" foram armazenadas na variável **dt**.

Valores Literais

Os valores inseridos diretamente no código fonte são chamados "valores literais".

Null

O valor nulo é representado pelo literal **null**, tanto em Java quanto em C#. Esse valor não pode ser utilizado para os tipos básicos numéricos e booleanos apresentados anteriormente (lembrando que o tipo **char** é um tipo numérico).

```
1 String nome = null;
```

Código Java 3.5: Inicializando uma string com null

```
1 string nome = null;
```

Código C# 3.6: Inicializando uma string com null

Booleanos

Em Java ou C#, o valor "verdadeiro" é representado pelo valor literal **true** e o valor "falso" pelo valor literal **false**.

```
1 boolean a = true;
2
3 boolean b = false;
```

Código Java 3.6: Utilizando valores literais booleanos em Java

```
1 bool a = true;
2
3 bool b = false;
```

Código C# 3.7: Utilizando valores literais booleanos em C#

Inteiros

Em Java, números inteiros podem ser definidos de quatro formas diferentes: binário, octal, decimal e hexadecimal. Para tanto, devemos seguir as seguintes regras:

- Se um número inteiro inicia com **0b** ou **0B** ele é binário;
- Se ele inicia com **0** é octal;
- Se inicia com **0x** ou **0X** é hexadecimal;
- Caso contrário é decimal.

```
1  // 19 em binário
2  int a = 0b10011;
3
4  // 19 em octal
5  int b = 023;
6
7  // 19 em decimal
8  int c = 19;
9
10 // 19 em hexadecimal
11 int d = 0x13;
```

Código Java 3.7: binário | octal | decimal | hexadecimal

Já em C#, esses números podem ser definidos apenas em decimal ou hexadecimal.

- Se um número inteiro inicia com **0x** ou **0X** ele é hexadecimal;
- Caso contrário é decimal.

```
1 // 19 em decimal
2 int c = 10;
3
4 // 19 em hexadecimal
5 int d = 0x13;
```

Código C# 3.8: decimal | hexadecimal

Mais Sobre

Como vimos, variáveis do tipo **int** não armazenam valores maiores do que **2.147.483.647**. Então, considere o valor inteiro **2.147.483.648**. Esse valor não pode ser armazenado em variáveis do tipo **int** pois ultrapassa o limite de **2.147.483.647**.

Por outro lado, o valor **2.147.483.648** pode ser armazenado em variáveis do tipo **long** já que esse tipo de variável aceita valores até **9.223.372.036.854.775.807**.

Em Java, o seguinte código gera erro de compilação.

```
1 // erro de compilação
2 long a = 2147483648;
```

Código Java 3.8: Erro de compilação

Para resolver esse problema, devemos utilizar o sufixo **L** ou **l**.

```
1 // valor literal inteiro do tipo long
2 long a = 2147483648L;
```

*Código Java 3.9: Utilizando o sufixo **L** em Java*

```
1 // valor literal inteiro do tipo long
2 long b = 2147483648l;
```

*Código Java 3.10: Utilizando o sufixo **l** em Java*

Reais

Em Java ou C#, valores literais reais são definidos com o separador de casas decimais "**.**"(ponto). Veja alguns exemplos:

```
1 double a = 19.19;
2
3 double b = 0.19;
4
5 double c = .19;
```

Código Java 3.11: Valores literais reais

```
1 double a = 19.19;
2
3 double b = 0.19;
4
5 double c = .19;
```

Código C# 3.9: Valores literais reais

Mais Sobre

Em Java ou C#, por padrão, independentemente da grandeza, os valores literais reais são tratados como **double**. Por exemplo, considere o valor **19.09**. Esse valor poderia ser tratado como **float** ou **double**. Contudo, por padrão, ele será tratado como **double**. Dessa forma, os códigos a seguir geram erros de compilação.

```
1 float a = 19.09;
```

Código Java 3.12: Erro de compilação

```
1 float a = 19.09;
```

Código C# 3.10: Erro de compilação

Para resolver esse problema, devemos utilizar o sufixo **F** ou **f**. Ao utilizar um desses sufixos, indicamos ao compilador que o valor literal real deve ser tratado como **float**.

```
1 float a = 19.09F;
2
3 float b = 19.09f;
```

*Código Java 3.13: Utilizando o sufixo **F** e **f** em Java*

```
1 float a = 19.09F;
2
3 float b = 19.09f;
```

*Código C# 3.11: Utilizando o sufixo **F** e **f** em C#*

Caracteres

Em Java ou C#, caracteres literais são definidos dentro de **aspas simples**. Veja alguns exemplos.

```
1 char a = 'K';
```

Código Java 3.14: Caracteres literais

```
1 char a = 'K';
```

Código C# 3.12: Caracteres literais

Nas inicializações acima, o valor numérico associado ao caractere **K** é armazenado nas variáveis.

Apenas um caractere pode ser definido dentro de **aspas simples**.

Strings literais

Em Java ou C#, strings literais são definidas dentro de **aspas duplas**. Veja alguns exemplos.

```
1 String a = "K19 Treinamentos";
```

Código Java 3.15: Strings literais

```
1 string a = "K19 Treinamentos";
```

Código C# 3.13: Strings literais

Determinados caracteres são especiais e não podem ser inseridos diretamente dentro das **aspas duplas**. Por exemplo, os códigos a seguir geram um erro de compilação pois utilizam o caractere especial "\".

```
1 String a = "C:\k19\rafael\cosentino";
```

Código Java 3.16: Erro de compilação

```
1 string a = "C:\k19\rafael\cosentino";
```

Código C# 3.14: Erro de compilação

Para solucionar esses erros, devemos utilizar o caractere "\" imediatamente antes dos caracteres especiais.

```
1 String a = "C:\\k19\\rafael\\cosentino";
```

*Código Java 3.17: Tratando os caracteres especiais com *

```
1 string a = "C:\\k19\\rafael\\cosentino";
```

*Código C# 3.15: Tratando os caracteres especiais com *

Em C#, podemos utilizar o caractere **@** no início das strings. Dessa forma, todos os caracteres especiais dentro das **aspas duplas** serão considerados caracteres normais.

```
1 string a = @"C:\k19\rafael\cosentino";
```

Código C# 3.16: Tratando os caracteres especiais com @

Números Aleatórios

Para realizar alguns exercícios ou mostrar alguns exemplos, utilizaremos números aleatórios. Esses números podem ser gerados facilmente com código Java ou C#. No exemplo a seguir, utilizamos a classe **Math** e o método **random()** do Java para gerar números aleatórios do tipo **double** maiores ou iguais a **0** e menores do que **1**.

```
1 double numero = Math.random();
```

Código Java 3.18: Gerando números aleatórios em Java

Podemos adaptar o intervalo dos números gerados com algumas operações matemáticas. Por exemplo, para gerar números maiores ou iguais a -50 e menores do que 50, basta realizar uma multiplicação e uma subtração.

```
1 double numero = Math.random() * 100 - 50;
```

Código Java 3.19: Gerando números aleatórios em Java

Em C#, para gerar números aleatórios do tipo **double** maiores ou iguais a **0** e menores do que **1**, devemos utilizar a classe **System.Random** e o método **NextDouble**().

```
1 System.Random gerador = new System.Random();
2 double numero = gerador.NextDouble();
```

Código C# 3.17: Gerando números aleatórios em C#

Novamente, podemos adaptar o intervalo dos números gerados com algumas operações matemáticas. Por exemplo, para gerar números maiores ou iguais a -25 e menores do que 50, basta realizar uma multiplicação e uma subtração.

```
1 System.Random gerador = new System.Random();
2 double numero = gerador.NextDouble() * 75 - 25;
```

Código C# 3.18: Gerando números aleatórios em C#

Exercícios de Fixação Com Java

1 Abra um terminal; entre na pasta dos seus exercícios e crie uma pasta chamada **variaveis** para os arquivos desenvolvidos nesse capítulo.

```
K19/rafael$ mkdir variaveis

K19/rafael$ cd variaveis

K19/rafael/variaveis$
```

Terminal 3.1: Criando a pasta `variaveis` *no Linux*

```
C:\Users\K19\rafael> md variaveis

C:\Users\K19\rafael> cd variaveis

C:\Users\K19\rafael\variaveis>
```

Terminal 3.2: Criando a pasta `variaveis` *no Windows*

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao1.zip

2 Na pasta **variaveis**, implemente um programa em Java que declare uma variável do tipo **int** chamada **idade**. Essa variável deve ser inicializada com o valor da sua idade. Por fim, exiba o valor da variável.

```
1 class TestaVariavel {
2   public static void main(String[] args) {
3     int idade;
4
5     idade = 27;
6
7     System.out.println(idade);
8   }
9 }
```

Código Java 3.20: TestaVariavel.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao2.zip

3 Compile e execute a classe **TestaVariavel**.

```
K19/rafael/variaveis$ javac TestaVariavel.java

K19/rafael/variaveis$ java TestaVariavel
27
```

Terminal 3.3: Compilando e executando a classe `TestaVariavel`

4 Na pasta **variaveis**, implemente um programa em Java que gere um número real aleatório entre **0** e **100**. Esse número deve ser armazenado em uma variável do tipo **double** chamada **numeroAleatorio**. Por fim, exiba o valor da variável.

```
1 class TestaNumeroAleatorio {
2   public static void main(String[] args) {
3     double numeroAleatorio;
4
5     numeroAleatorio = Math.random() * 100;
6
7     System.out.println(numeroAleatorio);
8   }
9 }
```

Código Java 3.21: TestaNumeroAleatorio.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao4.zip

5 Compile e execute a classe **TestaNumeroAleatorio**.

```
K19/rafael/variaveis$ javac TestaNumeroAleatorio.java

K19/rafael/variaveis$ java TestaNumeroAleatorio
19.775172204803429
```

Terminal 3.4: Compilando e executando a classe `TestaNumeroAleatorio`

6 Na pasta **variaveis**, implemente um programa em Java que declare uma variável do tipo **String** chamada **nome**. Essa variável deve ser inicializada com o seu nome. Por fim, exiba o valor da variável.

```
1 class TestaString {
2   public static void main(String[] args) {
3     String nome;
4
5     nome = "Rafael Cosentino";
6
7     System.out.println(nome);
8   }
9 }
```

Código Java 3.22: TestaString.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao6.zip

7 Compile e execute a classe **TestaString**.

```
K19/rafael/variaveis$ javac TestaString.java

K19/rafael/variaveis$ java TestaString
Rafael Cosentino
```

Terminal 3.5: Compilando e executando a classe `TestaString`

Casting

Considere um valor dentro do intervalo de valores do tipo **int**. Tanto no Java quanto no C#, esse valor pode ser armazenado em uma variável do tipo **long**, pois todos os valores que estão no intervalo do tipo **int** também estão no intervalo do tipo **long**.

Por causa disso, essas duas linguagens de programação permitem que qualquer valor armazenado em uma variável do tipo **int** possa ser copiado para uma variável do tipo **long**. Veja o exemplo a seguir.

```
1 int a = 19;
2 long b = a;
```

Código Java 3.23: Compatibilidade

```
1 int a = 19;
2 long b = a;
```

Código C# 3.19: Compatibilidade

Agora, considere um valor dentro do intervalo de valores do tipo **long**. Não podemos garantir que esse valor possa ser armazenado em uma variável do tipo **int** porque o intervalo do tipo **long** é mais abrangente do que o intervalo do tipo **int**. Por exemplo, o número **2147483648** está no intervalo do tipo **long** mas não está no intervalo do tipo **int**.

Por causa disso, as linguagens Java e C# não permitem que o valor de uma variável do tipo **long** seja copiado para uma variável do tipo **int**. A tentativa de realizar esse tipo de cópia gera erro de compilação mesmo que o valor armazenado na variável do tipo **long** seja compatível com **int**. Veja o exemplo a seguir.

```
1 long a = 19;
2 int b = a;
```

Código Java 3.24: Erro de compilação - Incompatibilidade

```
1 long a = 19;
2 int b = a;
```

Código C# 3.20: Erro de compilação - Incompatibilidade

Nesses casos, podemos aplicar uma operação de **conversão** também chamada de operação de **casting**. Veja como essa operação é aplicada.

```
1 long a = 19;
2 int b = (int)a;
```

Código Java 3.25: Conversão - Casting

```
1 long a = 19;
2 int b = (int)a;
```

Código C# 3.21: Conversão - Casting

Operações de casting podem gerar resultados bem indesejados. Considere que uma variável do tipo **long** armazena o valor **2147483648**. Se uma operação de casting for aplicada para copiar esse valor para uma variável do tipo **int** ocorrerá perda de precisão e o valor obtido na cópia será **-2147483648** tanto em Java quanto em C#.

```
1 long a = 2147483648L;
2 int b = (int)a; // b = -2147483648
```

Código Java 3.26: Casting - Perdendo precisão

```
1 long a = 2147483648L;
2 int b = (int)a; // b = -2147483648
```

Código C# 3.22: Casting - Perdendo precisão

Em geral, quando há o risco de perder precisão, os compiladores exigem a operação de casting. Isso funciona como um alerta para o programador. Contudo, em alguns casos, mesmo com o risco de perder precisão, os compiladores não exigem a operação de casting. Considere os exemplos a seguir.

```
1 long a = 9223372036854775807L;
2 float b = a; // b = 9223372000000000000
```

Código Java 3.27: Não precisa de casting mas tem perda de precisão

```
1 long a = 9223372036854775807L;;
2 float b = a; // b = 9223372000000000000
```

Código C# 3.23: Não precisa de casting mas tem perda de precisão

Nos exemplos acima, a variável do tipo **long** armazena o valor **9223372036854775807**. Ao copiar o conteúdo dessa variável para uma variável do tipo **float**, há uma perda precisão e o valor obtido é **9223372000000000000** tanto no Java quanto no C#.

Conversão de string

Considere uma variável do tipo **string** contendo o valor "19". Não podemos copiar o valor dessa variável para uma variável do tipo **int**, pois um erro de compilação seria gerado.

```
1 String a = "19";
2 int b = a;
```

Código Java 3.28: Erro de compilação - Incompatibilidade

```
1 string a = "19";
2 int b = a;
```

Código C# 3.24: Erro de compilação - Incompatibilidade

Nesses casos, é necessário realizar uma conversão de **string** para **int**. Em Java, essa conversão pode ser realizada da seguinte forma:

```
1 String a = "19";
2 int b = Integer.parseInt(a);
```

*Código Java 3.29: Conversão de **string** para **int***

Em C#, essa conversão pode ser realizada da seguinte forma:

```
1 string a = "19";
2 int b = System.Convert.ToInt32(a);
```

*Código C# 3.25: Conversão de **string** para **int***

A tabela a seguir mostra como as conversões são realizadas em Java e C#.

Java	
byte	Byte.parseByte()
short	Short.parseShort()
int	Integer.parseInt()
long	Long.parseLong()
float	Float.parseFloat()
double	Double.parseDouble()
boolean	Boolean.parseBoolean()

Tabela 3.3: Conversão em Java

C#	
sbyte	System.Convert.ToSByte()
byte	System.Convert.ToByte()
short	System.Convert.ToInt16()
ushort	System.Convert.ToUInt16()
int	System.Convert.ToInt32()
uint	System.Convert.ToUInt32()
long	System.Convert.ToInt64()
ulong	System.Convert.ToUInt64()
float	System.Convert.ToSingle()
double	System.Convert.ToDouble()
decimal	System.Convert.ToDecimal()
bool	System.Convert.ToBoolean()

Tabela 3.4: Conversão em C#

Convenções de nomenclatura

Os nomes das variáveis são fundamentais para o entendimento do código fonte. Considere o exemplo a seguir:

```
1 int j;
2 int f;
3 int m;
```

Você consegue deduzir quais dados serão armazenados nas variáveis **j**, **f** e **m**? Provavelmente, não. Vamos melhorar um pouco os nomes dessas variáveis.

```
1 int jan;
2 int fev;
3 int mar;
```

Agora, talvez, você tenha uma vaga ideia. Vamos melhorar mais um pouco os nomes dessas variáveis.

```
1 int janeiro;
2 int fevereiro;
3 int marco;
```

Agora sim! Você já sabe para que servem essas variáveis? Se você parar para pensar ainda não sabe muita coisa. Então, é importante melhorar mais uma vez o nome dessas variáveis.

```
1 int numeroDePedidosEmJaneiro;
2 int numeroDePedidosEmFevereiro;
3 int numeroDePedidosEmMarco;
```

Finalmente, os nomes das variáveis conseguem expressar melhor a intenção delas. Consequentemente, a leitura e o entendimento do código fonte seria mais fácil.

Geralmente, bons nomes de variáveis são compostos por várias palavras como no exemplo a seguir.

```
1 int numeroDeCandidatosAprovados;
```

Quando o nome de uma variável é composto, é fundamental adotar alguma convenção para identificar o início e o termino das palavras. A separação natural das palavras na língua portuguesa são os espaços. Contudo, os nomes das variáveis em Java ou C# não podem possuir espaços. Não adotar nenhuma convenção de nomenclatura para identificar o início e o termino das palavras é como escrever um texto em português sem espaços entre as palavras. Em alguns casos, o leitor não saberia como separar as palavras. Considere o exemplo abaixo.

```
salamesadia
```

O que está escrito no texto acima? A resposta depende da divisão das palavras. Você pode ler como “sala mesa dia” ou “salame sadia”. Dessa forma, fica claro a necessidade deixar visualmente explícito a divisão das palavras.

Em algumas linguagens de programação, delimitadores são utilizados para separar as palavras que formam o nome de uma variável.

```
numero_de_candidatos_aprovados;
numero-de-candidatos-aprovados;
```

Em outras linguagens de programação, letras maiúsculas e minúsculas são utilizadas para separar as palavras.

```
NumeroDeCandidatosAprovados;
numeroDeCandidatosAprovados;
```

Em Java ou em C#, a convenção de nomenclatura adotada para separar as palavras que formam o nome de uma variável é o **Camel Case**, que consiste em escrever o nome da variável com a primeira letra de cada palavra em maiúscula com exceção da primeira letra da primeira palavra.

```
int numeroDaConta;
int NumeroDaConta; // não segue a convenção
```

Código Java 3.35: Convenção para a escrita dos nomes das variáveis em Java e C#.

Também devemos nos lembrar que as duas linguagens são **Case Sensitive**. Dessa forma, **numeroDaConta** e `NumeroDaConta` são consideradas variáveis diferentes pelo fato do nome da primeira começar com letra minúscula e o da segunda com maiúscula.

Regras de nomenclatura

As linguagens Java e C# possuem regras técnicas muito parecidas a respeito da nomenclatura das variáveis. O nome de uma variável:

1. Não deve começar com um dígito;
2. Não pode ser igual a uma palavra reservada;
3. Não pode conter espaço(s);
4. Pode ser uma palavra de qualquer tamanho;
5. Pode conter letras, dígitos e _ (underscore).
6. Em Java, pode conter também o caractere **$**.

```
// válido
int numeroDaConta;

// inválido pois o nome de uma variável não pode começar com um dígito
int 2outraVariavel;

// inválido pois o nome de uma variável não pode ser igual a uma palavra reservada
double double;

// inválido pois o nome de uma variável não pode conter espaços
double saldo da conta;

// válido
int umaVariavelComUmNomeSuperHiperMegaUltraGigante;

// válido
int numeroDaContaCom8Digitos_semPontos;

// válido somente em Java
int valorDoProdutoEmR$;

// inválido pois o nome de uma variável não pode conter o caractere #
int #telefone;
```

Código Java 3.36: Exemplos de nomes de variáveis válidos e inválidos

As linguagens Java e C# permitem a criação de nomes de variáveis em qualquer idioma, pois elas aceitam qualquer caractere **Unicode UTF-16**. Portanto são válidas as variáveis escritas com as acentuações do português, assim como as variáveis escritas em japonês, por exemplo.

Apesar de ser possível o uso de caracteres especiais, assim como o uso dos caracteres **$** (cifrão) e _ (underscore), não é recomendável utilizá-los. Não utilizar tais caracteres é uma boa prática de programação. Essa prática facilita a leitura do código fonte em qualquer editor de texto.

Keywords

Toda linguagem de programação possui um conjunto de palavras reservadas. Em geral, essas palavras representam os comandos da linguagem. Abaixo você pode visualizar as palavras reservadas do Java e do C#.

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Tabela 3.5: Keywords do Java

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit in	int	interface	
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Tabela 3.6: Keywords do C#

Formatação

Considere uma variável que armazena o preço de um produto. Geralmente, preços possuem casas decimais. Dessa forma, devemos escolher um tipo que permita o armazenamento de números reais. Por exemplo, podemos escolher o tipo **double** no Java ou no C#. Veja os exemplos a seguir.

```
1 double preco = Math.random() * 100;
```

Código Java 3.37: Preço de um produto

```
1 System.Random gerador = new System.Random();
2 double preco = gerador.NextDouble() * 100;
```

Código C# 3.26: Preço de um produto

Nos exemplos anteriores, os preços dos produtos foram gerados aleatoriamente. Com alta probabilidade, esses valores possuirão mais do que duas casas decimais. Contudo, provavelmente, seria mais conveniente exibir os preços apenas com duas casas decimais. Isso pode ser feito facilmente em Java ou C# através das **máscaras de formatação**.

```
1 System.out.printf("%.2f", preco);
```

Código Java 3.38: Exibindo números formatados em Java

```
1 System.Console.WriteLine("{0:F2}", preco);
```

Código C# 3.27: Exibindo números formatados em C#

Podemos inserir diversos parâmetros nas máscaras de formatação. Em Java, cada parâmetro deve ser indicado com o caractere "**%**".

```
1 System.out.printf("%1$s tem %2$d anos e pesa %3$.2f", "Jonas" , 30, 49.459);
```

Código Java 3.39: Parâmetros na máscara de formatação

No exemplo, o trecho "**%1$s**" indica que o primeiro parâmetro da máscara é uma string. Já o trecho "**%2$d**" indica que o segundo parâmetro é um número inteiro. Por fim, o trecho "**%3$.2f**" indica que o terceiro parâmetro é um número real formatado com duas casas decimais.

S ou s: string

D ou d: número inteiro decimal

X ou x: número inteiro decimal

f: número real

Em C#, os parâmetros são definidos com **chaves** ("{}").

```
1 System.Console.WriteLine("{0} tem {1:D} anos e pesa {2:F2}", "Jonas", 30, 49.459);
```

Código C# 3.28: Parâmetros na máscara de formatação

No exemplo, o trecho "**{0}**" indica onde o primeiro parâmetro deve ser inserido. Já o trecho "**{1:D}**" indica que o segundo parâmetro é um número inteiro. Por fim, o trecho "**{2:F2}**" indica que o terceiro parâmetro é um número real formatado com duas casas decimais.

D ou d: número inteiro decimal

X ou x: número inteiro hexadecimal

F ou f: número real

Formatação de Data e Hora (Conteúdo Extra)

Normalmente, o formato padrão para exibir data e hora varia de país para país. ou de região para região. Por exemplo, os brasileiros estão mais acostumados com o formato de data "dia/mês/ano". Por outro lado, os americanos costumam utilizar o formato "mês/dia/ano".

Tanto em Java quanto em C#, podemos formatar data e hora facilmente. No código Java abaixo, a formatação "dia/mês/ano hora:minutos:segundos" está sendo aplicada.

```
1 java.util.Calendar fundacaoK19 =
2   new java.util.GregorianCalendar(2010, 7, 27, 10, 32, 15);
3
4 java.text.SimpleDateFormat sdf =
5   new java.text.SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
6
7 String fundacaoK19Formatada = sdf.format(fundacaoK19.getTime());
```

Código Java 3.40: Aplicando o formado "dia/mês/ano hora:minutos:segundos"

Na máscara de formatação, devemos utilizar os caracteres especiais para definir o formato desejado. Veja o que cada caractere indica.

d: dia

M: mês

y: ano

H: hora

m: minutos

s: segundos

Quando o caractere **d** é utilizado de forma simples na máscara de formatação, os dias de 1 até 9 são formatados com apenas um dígito. Quando utilizamos **dd**, os dias de 1 até 9 são formatados com apenas dois dígitos (01, 02, 03, ..., 09). Analogamente, para o mês, ano, hora, minutos e segundos.

Agora, veremos a formatação de data e hora no C#. Veja o exemplo a seguir.

```
1 System.DateTime fundacaoK19 =
2   new System.DateTime(2010, 7, 27, 10, 32, 15);
3
4 string fundacaoK19Formatada = fundacaoK19.ToString("dd/MM/yyyy HH:mm:ss");
```

Código C# 3.29: Aplicando o formado "dia/mês/ano hora:minutos:segundos"

A máscara de formatação de data e hora do C# funciona de forma muito semelhante a do Java.

Exercícios de Fixação Com Java

8 Na pasta **variaveis**, implemente um programa em Java para exibir os valores formatados de algumas variáveis.

```
1 class TestaFormatacao {
2   public static void main(String[] args) {
3     String nome = "Jonas Hirata";
4     int idade = 30;
5     double peso = 49.7345;
6
7     System.out.printf("O %1$s tem %2$d anos e pesa %3$.2f kg\n", nome, idade, peso);
8   }
9 }
```

Código Java 3.41: TestaFormatacao.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao8.zip

9 Compile e execute a classe **TestaFormatacao**.

```
K19/rafael/variaveis$ javac TestaFormatacao.java

K19/rafael/variaveis$ java TestaFormatacao
O Jonas Hirata tem 30 anos e pesa 49.73
```

Terminal 3.6: Compilando e executando a classe `TestaFormatacao`

10 Na pasta **variaveis**, implemente um programa em Java que realiza uma operação de casting.

```
1 class TestaCasting {
2   public static void main(String[] args) {
3     long a = 2147483648L;
4
5     int b = (int)a;
6
7     System.out.println(a);
8     System.out.println(b);
9   }
10 }
```

Código Java 3.42: TestaCasting.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao10.zip

11 Compile e execute a classe **TestaCasting**.

```
K19/rafael/variaveis$ javac TestaCasting.java

K19/rafael/variaveis$ java TestaCasting
2147483648
-2147483648
```

Terminal 3.7: Compilando e executando a classe `TestaCasting`

12 Na pasta **variaveis**, implemente um programa em Java que realiza uma operação de conversão de string.

```
class TestaConversao {
  public static void main(String[] args) {
    String s = "19.09";

    double d = Double.parseDouble(s);

    System.out.println(d);
  }
}
```

Código Java 3.43: TestaConversao.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao12.zip

13 Compile e execute a classe **TestaConversao**.

```
K19/rafael/variaveis$ javac TestaConversao.java

K19/rafael/variaveis$ java TestaConversao
19.09
```

Terminal 3.8: Compilando e executando a classe `TestaConversao`

Erro: Variáveis com nomes repetidos

Um erro de compilação comum em Java ou C# ocorre quando duas ou mais variáveis são declaradas com nome repetido em um mesmo bloco. Veja um exemplo de programa em Java com esse problema.

```
class Programa {
  public static void main(String[] args) {
    int a = 10;

    double a = 10.7;

    int a = 5;
  }
}
```

Código Java 3.44: Programa.java

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.java:5: error: variable a is already defined in method main(String[])
    double a = 10.7;
           ^
Programa.java:7: error: variable a is already defined in method main(String[])
    int a = 5;
        ^
2 errors
```

Terminal 3.9: Erro de compilação

Agora, veja um exemplo de programa em C# com esse problema.

```
class Programa
```

```
2 {
3   static void Main()
4   {
5     int a = 10;
6
7     double a = 10.7;
8
9     int a = 5;
10  }
11 }
```

Código C# 3.30: Programa.cs

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.cs(7,10): error CS0128: A local variable named 'a' is already defined in this scope
Programa.cs(7,14): error CS0266: Cannot implicitly convert type 'double' to 'int'.
An explicit conversion exists (are you missing a cast?)
Programa.cs(9,7): error CS0128: A local variable named 'a' is already defined in this scope
```

Terminal 3.10: Erro de compilação

Erro: Esquecer a inicialização de uma variável local

Outro erro de compilação comum em Java ou C# ocorre quando utilizamos uma variável local não inicializada. Veja um exemplo de programa em Java com esse problema.

```
1 class Programa {
2   public static void main(String[] args) {
3     int a;
4
5     System.out.println(a);
6   }
7 }
```

Código Java 3.45: Programa.java

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.java:5: error: variable a might not have been initialized
    System.out.println(a);
                       ^
1 error
```

Terminal 3.11: Erro de compilação

Agora, veja um exemplo de programa em C# com esse problema.

```
1 class Programa
2 {
3   static void Main()
4   {
5     int a;
6
7     System.Console.WriteLine(a);
8   }
9 }
```

Código C# 3.31: Programa.cs

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.cs(7,42): error CS0165: Use of unassigned local variable 'a'
```

Terminal 3.12: Erro de compilação

Erro: Trocar aspas simples por aspas duplas ou vice-versa

Mais um erro comum em Java ou C# ocorre quando utilizamos aspas simples onde deveria ser aspas duplas ou vice-versa. Veja um exemplo de programa em Java que utiliza aspas duplas onde deveria ser aspas simples.

```
1 class Programa {
2   public static void main(String[] args) {
3     char c = "A";
4   }
5 }
```

Código Java 3.46: Programa.java

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.java:3: error: incompatible types
    char c = "A";
             ^
  required: char
  found:    String
1 error
```

Terminal 3.13: Erro de compilação

Agora, veja um exemplo de programa em Java que utiliza aspas simples onde deveria ser aspas duplas.

```
1 class Programa {
2   public static void main(String[] args) {
3     String s = 'K19 Treinamentos';
4   }
5 }
```

Código Java 3.47: Programa.java

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.java:3: error: unclosed character literal
    String s = 'K19 Treinamentos';
               ^
Programa.java:3: error: not a statement
    String s = 'K19 Treinamentos';
                ^
Programa.java:3: error: ';' expected
    String s = 'K19 Treinamentos';
                  ^
Programa.java:3: error: unclosed character literal
    String s = 'K19 Treinamentos';
                               ^
Programa.java:3: error: not a statement
    String s = 'K19 Treinamentos';
                  ^
5 errors
```

Terminal 3.14: Erro de compilação

Agora, veja um exemplo de programa em C# que utiliza aspas duplas onde deveria ser aspas simples.

```
1 class Programa
2 {
3   static void Main()
4   {
5     char c = "A";
6   }
7 }
```

Código C# 3.32: Programa.cs

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.cs(5,22): error CS0029: Cannot implicitly convert type ‘string’ to ‘char’
```

Terminal 3.15: Erro de compilação

Agora, veja um exemplo de programa em C# que utiliza aspas simples onde deveria ser aspas duplas.

```
1 class Programa
2 {
3   static void Main()
4   {
5     string s = 'K19 Treinamentos';
6   }
7 }
```

Código C# 3.33: Programa.cs

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.cs(5,30): error CS1012: Too many characters in character literal
```

Terminal 3.16: Erro de compilação

Erro: Utilizar o separador decimal errado

Outro erro de compilação comum em Java ou C# ocorre quando não utilizamos o separador decimal correto. Veja um exemplo de programa em Java com esse problema.

```
1 class Programa {
2   public static void main(String[] args) {
3     double d = 19,09;
4   }
5 }
```

Código Java 3.48: Programa.java

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.java:3: error: <identifier> expected
    double d = 19,09;
               ^
1 error
```

Terminal 3.17: Erro de compilação

Agora, veja um exemplo de programa em C# com esse problema.

```
1 class Programa
2 {
3   static void Main()
4   {
5     double d = 19,09;
6   }
7 }
```

Código C# 3.34: Programa.cs

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.cs(5,17): error CS1001: Identifier Expected
```

Terminal 3.18: Erro de compilação

Erro: Valores incompatíveis com os tipos das variáveis

Também é um erro de compilação comum em Java ou C# atribuir valores incompatíveis com os tipos das variáveis. Veja um exemplo de programa em Java com esse problema.

```
1 class Programa {
2   public static void main(String[] args) {
3     int a = 19.09;
4   }
5 }
```

Código Java 3.49: Programa.java

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.java:3: error: possible loss of precision
    int a = 19.09;
            ^
  required: int
  found:    double
1 error
```

Terminal 3.19: Erro de compilação

Agora, veja um exemplo de programa em C# com esse problema.

```
1 class Programa
2 {
3   static void Main()
4   {
5     int a = 19.09;
6   }
7 }
```

Código C# 3.35: Programa.cs

A mensagem de erro de compilação seria semelhante a apresentada abaixo.

```
Programa.cs(5,11): error CS0266: Cannot implicitly convert type 'double' to 'int'.
An explicit conversion exists (are you missing a cast?)
```

Terminal 3.20: Erro de compilação

Exercícios de Fixação Com C#

14 Na pasta **variaveis**, implemente um programa em C# que declare uma variável do tipo **int** chamada **idade**. Essa variável deve ser inicializada com o valor da sua idade. Por fim, exiba o valor da variável.

```
1  class TestaVariavel
2  {
3    static void Main()
4    {
5      int idade;
6
7      idade = 27;
8
9      System.Console.WriteLine(idade);
10   }
11 }
```

Código C# 3.36: TestaVariavel.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao14.zip

15 Compile e execute a classe **TestaVariavel**.

```
C:\Users\K19\rafael\variaveis> csc TestaVariavel.cs

C:\Users\K19\rafael\variaveis> TestaVariavel.exe
27
```

Terminal 3.21: Compilando e executando a classe `TestaVariavel`

16 Na pasta **variaveis**, implemente um programa em C# que gere um número real aleatório entre **0** e **100**. Esse número deve ser armazenado em uma variável do tipo **double** chamada **numeroAleatorio**. Por fim, exiba o valor da variável.

```
1  class TestaNumeroAleatorio
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      double numeroAleatorio = gerador.NextDouble() * 100;
8
9      System.Console.WriteLine(numeroAleatorio);
10   }
11 }
```

Código C# 3.37: TestaNumeroAleatorio.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao16.zip

17 Compile e execute a classe **TestaNumeroAleatorio**.

```
C:\Users\K19\rafael\variaveis> csc TestaNumeroAleatorio.cs

C:\Users\K19\rafael\variaveis> TestaNumeroAleatorio.exe
19.1009406745904
```

Terminal 3.22: Compilando e executando a classe `TestaVariavel`

18 Na pasta **variaveis**, implemente um programa em C# que declare uma variável do tipo **string** chamada **nome**. Essa variável deve ser inicializada com o seu nome. Por fim, exiba o valor da variável.

```
1  class TestaString
2  {
3    static void Main()
4    {
5      string nome;
6
7      nome = "Rafael Cosentino";
8
9      System.Console.WriteLine(nome);
10   }
11 }
```

Código C# 3.38: TestaString.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao18.zip

19 Compile e execute a classe **TestaString**.

```
C:\Users\K19\rafael\variaveis> csc TestaString.cs

C:\Users\K19\rafael\variaveis> TestaString
Rafael Cosentino
```

Terminal 3.23: Compilando e executando a classe `TestaString`

20 Na pasta **variaveis**, implemente um programa em C# para exibir os valores de algumas variáveis formatados.

```
1  class TestaFormatacao
2  {
3    static void Main()
4    {
5      string nome = "Jonas Hirata";
6      int idade = 30;
7      double peso = 49.7345;
8
9      System.Console.WriteLine("O {0} tem {1} anos e pesa {2:F2} kg", nome,idade,peso);
10   }
11 }
```

Código C# 3.39: TestaFormatacao.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao20.zip

21 Compile e execute a classe **TestaFormatacao**.

```
K19/rafael/variaveis$ javac TestaFormatacao.java

K19/rafael/variaveis$ java TestaFormatacao
O Jonas Hirata tem 30 anos e pesa 49.73
```

Terminal 3.24: Compilando e executando a classe `TestaFormatacao`

22 Na pasta **variaveis**, implemente um programa em C# que realiza uma operação de casting.

```
1  class TestaCasting
2  {
3    static void Main()
4    {
5      long a = 2147483648L;
6
7      int b = (int)a;
8
9      System.Console.WriteLine(a);
10     System.Console.WriteLine(b);
11   }
12 }
```

Código C# 3.40: TestaCasting.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao22.zip

23 Compile e execute a classe **TestaCasting**.

```
C:\Users\K19\rafael\variaveis> csc TestaCasting.cs

C:\Users\K19\rafael\variaveis> TestaCasting.exe
2147483648
-2147483648
```

Terminal 3.25: Compilando e executando a classe `TestaCasting`

24 Na pasta **variaveis**, implemente um programa em C# que realiza uma operação de conversão de string.

```
1  class TestaConversao
2  {
3    static void Main()
4    {
5      string s = "19.09";
6
7      double d = System.Convert.ToDouble(s);
8
9      System.Console.WriteLine(d);
10   }
11 }
```

Código Java 3.50: TestaConversao.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-fixacao24.zip

25 Compile e execute a classe **TestaConversao**.

```
C:\Users\K19\rafael\variaveis> csc TestaConversao.cs
```

```
C:\Users\K19\rafael\variaveis> TestaConversao.exe
19.09
```

Terminal 3.26: Compilando e executando a classe `TestaConversao`

Exercícios Complementares

1 Indique os tipos adequados da linguagem Java e C# para cada valor da lista abaixo.

1. “Bom dia”
2. 3
3. 235.13
4. true
5. -135
6. 256.23F
7. ‘A’
8. 6463275245745L

2 Na pasta **variaveis**, implemente um programa em Java que declare uma variável do tipo **double** chamada **peso**. Essa variável deve ser inicializada com o valor do seu peso. Exiba o valor dessa variável.

3 Na pasta **variaveis**, implemente um programa em Java que declare uma variável de cada um dos tipos básicos vistos nesse capítulo. Essas variáveis devem ser inicializadas com valores adequados. Por fim, exiba os valores dessas variáveis.

4 Na pasta **variaveis**, implemente um programa em Java que declare uma variável do tipo string e inicialize-a com o valor “1571.11”. Depois, com uma conversão, copie esse valor para uma variável do tipo **double**.

5 Na pasta **variaveis**, implemente um programa em Java que declare uma variável do tipo **java.util.Calendar** e inicialize-a com a data e a hora atuais. Depois, declare uma outra variável do tipo **java.util.Calendar** e inicialize-a com a data “27 de Agosto de 2010” e hora “10:32:15”. Por fim, formate e exiba essas datas.

6 Na pasta **variaveis**, implemente um programa em C# que declare uma variável do tipo **double** chamada **peso**. Essa variável deve ser inicializada com o valor do seu peso. Exiba o valor dessa variável.

7 Na pasta **variaveis**, implemente um programa em C# que declare uma variável de cada um dos tipos básicos vistos nesse capítulo. Essas variáveis devem ser inicializadas com valores adequados. Por fim, exiba os valores dessas variáveis.

8 Na pasta **variaveis**, implemente um programa em C# que declare uma variável do tipo string e inicialize-a com o valor "1571.11". Depois, com uma conversão, copie esse valor para uma variável do tipo **double**.

9 Na pasta **variaveis**, implemente um programa em C# que declare uma variável do tipo **System.DateTime** e inicialize-a com a data e a hora atuais. Depois, declare uma outra variável do tipo **System.DateTime** e inicialize-a com a data "27 de Agosto de 2010" e hora "10:32:15". Por fim, formate e exiba essas datas.

10 Considere um sistema de gerenciamento de mercadorias de uma loja. Implemente um programa que declare variáveis para representar os seguintes dados: número do pedido, código do produto, quantidade e valor total da compra. Inicialize essas variáveis com valores apropriados. Por fim, exiba os valores armazenados.

Desafios

1 Olhando para a solução dada nos exercícios complementares, você faria alguma alteração caso estivéssemos desenvolvendo o sistema para uma loja pequena? E se fosse para uma grande rede de lojas? Quais seriam as alterações e quais as implicações?

Resumo do Capítulo

1 A função de uma **variável** é **armazenar** uma informação (dado).

2 Toda variável possui um **nome** (**identificador**).

3 Em Java ou C#, toda variável está associada a um **tipo**.

4 Para armazenar um valor em uma variável, esse valor deve ser **compatível** com o tipo da variável.

5 Em Java ou C#, as variáveis devem ser **inicializadas** antes de serem utilizadas.

6 Uma variável do tipo **string**, pode armazenar uma sequência de caracteres.

7 No Java, as datas e horas podem ser armazenadas em variáveis do tipo **java.util.Calendar**.

8 No C#, as datas e horas podem ser armazenadas em variáveis do tipo **System.DateTime**.

9 O separador de casas decimais em Java ou C# é o "**.**"(ponto).

10 Em Java ou C#, os valores literais booleanos são **true** e **false**.

11 Strings literais são definidas dentro de **aspas duplas** no Java e no C#.

12 Operações de **casting** podem gerar resultados bem diferentes dos desejados.

13 As **convenções de nomenclatura** de variáveis são importantes para melhorar a legibilidade do código.

14 No Java e no C#, as convenções de nomenclatura de variáveis são baseadas em letras **maiúsculas** e **minúsculas**.

15 Podemos gerar números aleatórios em Java com o método **Math.random()**.

16 Podemos gerar números aleatórios em C# com a classe **System.Random**.

Prova

1 Qual é a função das variáveis?

a) Exibir as mensagens dos programas.

b) Gerar números aleatórios.

c) Formatar números com casas decimais.

d) Armazenar dados.

e) Realizar cálculos matemáticos.

2 O que ocorre quando uma variável não inicializada é utilizada?

a) Um erro de compilação.

b) Um erro de execução.

c) A variável é inicializada com 0.

d) A variável é inicializada com um valor aleatório.

e) A variável é inicializada com null.

3 Qual alternativa apresenta os tipos básicos do Java para números reais?

a) byte e double

b) int e float

c) float e double

d) real e long

e) single e double

4 Qual alternativa apresenta os tipos básicos do C# para números reais?

a) byte, long e double

b) float, double e decimal

c) float e double

d) float e real

e) double e decimal

5 Considere as linguagens Java e C#, qual alternativa declara corretamente um caractere literal?

a) 'K'

b) 'KK'

c) "K"

d) "KK"

e) K

f) KK

6 No Java e no C#, quais palavras representam os valores literais booleanos?

a) verdadeiro e falso

b) True e False

c) True e false

d) true ou verdadeiro e false ou falso

e) true e false

7 Em Java, quais são as formas de definir os valores literais numéricos inteiros?

a) decimal e hexadecimal

b) binário e decimal

c) octal e decimal

d) binário, decimal e hexadecimal

e) binário, octal, decimal e hexadecimal

8 Em C#, quais são as formas de definir os valores literais numéricos inteiros?

a) decimal e hexadecimal

b) binário e decimal

c) octal e decimal

d) binário, decimal e hexadecimal

e) binário, octal, decimal e hexadecimal

9 Considere as linguagens Java e C#, como são definidas as strings?

a) Dentro de aspas simples.

b) Dentro de aspas duplas.

c) Dentro de aspas simples ou aspas duplas.

10 Qual afirmação sobre casting está correta?

a) As linguagens Java e C# não permitem operações de casting.

b) Operações de casting são utilizadas para copiar valores entre variáveis do mesmo tipo.

c) Operações de casting são perigosas pois podem gerar valores indesejados.

d) Nas linguagens Java e C#, as operações de casting são utilizadas para transformar strings em números.

e) Nas linguagens Java e C#, as operações de casting são utilizadas para transformar números em strings.

11 Qual nome de variável segue a convenção de nomenclatura do Java e do C#?

a) idadeDoMarcelo

b) idadedomarcelo

c) idade_do_marcelo

d) idade-do-marcelo

e) idade do marcelo

12 Qual nome de variável segue as regras de nomeclatura do Java e do C#?

a) 90pesoMinimoDoMarcelo

b) int

c) pesoDoMarceloEstaAcimaDe90

d) peso.do.marcelo

e) peso do marcelo

Minha Pontuação [ ] Pontuação Mínima: [9] Pontuação Máxima: [12]

CAPÍTULO 4

OPERADORES

Tipos de Operadores

Para manipular as variáveis de uma aplicação, devemos utilizar os operadores oferecidos pela linguagem de programação que estamos utilizando. As linguagens Java e C# possuem diversos operadores. Os principais operadores dessas linguagens são:

- Aritmético: **+ - * / %**
- Atribuição: **= += -= *= /= %= ++ --**
- Relacional: **== != < <= > >=**
- Lógico: **& | ^ && ||**

Operadores Aritméticos

Os operadores aritméticos funcionam de forma muito semelhante aos operadores da matemática. Os operadores aritméticos são:

- Adição **+**
- Subtração **-**
- Multiplicação *****
- Divisão **/**
- Módulo **%**

```
1  int umMaisUm = 1 + 1;
2  // umMaisUm = 2
3
4  int tresVezesDois = 3 * 2;
5  // tresVezesDois = 6
6
7  int quatroDivididoPorDois = 4 / 2;
8  // quatroDivididoPorDois = 2
9
10 int seisModuloCinco = 6 % 5;
11 // seisModuloCinco = 1
12
13 int x = 7;
14
15 x = x + 1 * 2;
16 // x = 9
17
18 x = x - 4;
19 // x = 5
20
```

```
21 x = x / (6 - 2 + (3 * 5)/(16 - 1));
22 // x = 1
```

Código Java 4.1: Exemplo de uso dos operadores aritméticos.

Importante

O módulo de um número x, na matemática, é o valor numérico de x desconsiderando o seu sinal (valor absoluto). Na matemática, expressamos o módulo da seguinte forma: $|-2| = 2$.

Em linguagens de programação, o módulo de um número é o resto da divisão desse número por outro. No exemplo acima, o resto da divisão de 6 por 5 é igual a 1. Além disso, lemos a expressão "6%5" da seguinte forma: seis módulo cinco.

Importante

As operações aritméticas em Java e C# obedecem as mesmas regras da matemática com relação à precedência dos operadores e parênteses. Portanto, o cálculo começa com as operações definidas nos parênteses mais internos até os mais externos. As operações de multiplicação, divisão e módulo são resolvidas antes das operações de subtração e adição.

Mais Sobre

As operações de potenciação e raiz quadrada podem ser realizadas através dos métodos **Math.pow** e **Math.sqrt** em Java ou através dos métodos **Math.Pow** e **Math.Sqrt** em C#. Veja alguns exemplos.

```
1 double a = Math.pow(3, 5);
2 // a = 243
3
4 double b = Math.sqrt(9);
5 // b = 3
```

Código Java 4.2: Potenciação e raiz quadrada

```
1 double a = Math.Pow(3, 5);
2 // a = 243
3
4 double b = Math.Sqrt(9);
5 // b = 3
```

Código C# 4.1: Potenciação e raiz quadrada

Divisão Inteira

Considere uma operação de divisão entre valores inteiros. Por exemplo, uma divisão entre valores do tipo básico **int**.

```
1 int a = 5;
2 int b = 2;
```

```
3 System.out.println(a / b);
```

Código Java 4.3: Divisão inteira

```
1 int a = 5;
2 int b = 2;
3 System.Console.WriteLine(a / b);
```

Código C# 4.2: Divisão inteira

Matematicamente, o resultado da operação "5 / 2" é "2.5". Contudo, nos exemplos acima, o valor obtido na divisão "a / b" é **2**. Em Java ou C#, quando ocorre uma divisão entres dois valores inteiros, a parte fracionária é descartada.

Podemos, explicitamente, converter um dos valores envolvidos na divisão ou até mesmo os dois para algum tipo numérico real. Dessa forma, a divisão não seria inteira e a parte fracionária não seria descartada. Essas conversões podem ser realizadas com operações de casting. Nos exemplos a seguir, o resultado das operações de divisão é **2.5**.

```
1  int a = 5;
2  int b = 2;
3
4  // convertendo apenas o "a"
5  System.out.println((double)a / b);
6
7  // convertendo apenas o "b"
8  System.out.println(a / (double)b);
9
10 // convertendo apenas o "a" e o "b"
11 System.out.println((double)a / (double)b);
```

Código Java 4.4: Castings

```
1  int a = 5;
2  int b = 2;
3
4  // convertendo apenas o "a"
5  System.Console.WriteLine((double)a / b);
6
7  // convertendo apenas o "b"
8  System.Console.WriteLine((double)a / (double)b);
9
10 // convertendo apenas o "a" e o "b"
11 System.Console.WriteLine((double)a / (double)b);
```

Código C# 4.3: Castings

Pare para pensar...
Considerando o que foi discutido anteriormente a respeito de divisão inteira e casting. Qual é o resultado da operação do exemplo a seguir?

```
1 double d = (double)(5 / 2);
```

Concatenação de Strings

Como vimos anteriormente, o operador + é utilizado para realizar soma aritmética. Mas, ele também pode ser utilizado para concatenar strings tanto no Java quanto no C#. Veja alguns exemplos.

```
String s1 = "Marcelo";
String s2 = " ";
String s3 = "Martins";

// "Marcelo Martins"
String s4 = s1 + s2 + s3;
```

```
string s1 = "Marcelo";
string s2 = " ";
string s3 = "Martins";

// "Marcelo Martins"
string s4 = s1 + s2 + s3;
```

Considere os exemplos a seguir.

```
String s1 = "Idade: ";
int idade = 30;

// "Idade: 30"
String s2 = s1 + idade;
```

```
string s1 = "Idade: ";
int idade = 30;

// "Idade: 30"
string s2 = s1 + idade;
```

Observe que o operador + foi aplicado a valores do tipo **int** e do tipo **string**. Nesses casos, os valores do tipo **int** são, automaticamente, convertidos para **string** e a concatenação é realizada. Analogamente, essa conversão ocorrerá toda vez que o operador + for aplicado a valores não **string** com valores do tipo **string**.

Pare para pensar...

As expressões são avaliadas da esquerda para a direita. Dessa forma, considere os seguintes exemplos:

```
System.out.println(1 + 2 + 3 + " testando");
System.out.println("testando" + 1 + 2 + 3);
```

```
System.Console.WriteLine(1 + 2 + 3 + " testando");
System.Console.WriteLine("testando" + 1 + 2 + 3);
```

O que seria exibido nesses exemplos?

Exercícios de Fixação Com Java

1 Abra um terminal; Entre na pasta com o seu nome e crie uma pasta chamada **operadores** para os arquivos desenvolvidos nesse capítulo.

```
K19/rafael$ mkdir operadores

K19/rafael$ cd operadores

K19/rafael/operadores$
```

Terminal 4.1: Criando a pasta `operadores`

```
C:\Users\K19\rafael> md operadores

C:\Users\K19\rafael> cd operadores

C:\Users\K19\rafael\operadores>
```

Terminal 4.2: Criando a pasta `variaveis` *no Windows*

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao1.zip

2 Na pasta **operadores**, implemente um programa em Java que utilize os operadores aritméticos.

```
1  class TestaOperadoresAritmeticos {
2    public static void main(String[] args) {
3      int a = 1 + 1;
4      int b = 10 - 2;
5      int c = 2 * 3;
6      int d = 25 / 5;
7      int e = 10 % 4;
8
9      System.out.println(a);
10     System.out.println(b);
11     System.out.println(c);
12     System.out.println(d);
13     System.out.println(e);
14   }
15 }
```

Código Java 4.10: TestaOperadoresAritmeticos.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao2.zip

3 Compile e execute a classe **TestaOperadoresAritmeticos**.

```
K19/rafael/operadores$ javac TestaOperadoresAritmeticos.java

K19/rafael/operadores$ java TestaOperadoresAritmeticos
2
8
6
5
2
```

Terminal 4.3: Compilando e executando a classe `TestaOperadoresAritmeticos`

4 Na pasta **operadores**, implemente um programa em Java que realize operações de divisão inteira e de casting.

```
1  class TestaDivisaoInteira {
2    public static void main(String[] args) {
```

```
3       int a = 5;
4       int b = 2;
5
6       System.out.println(a / b);
7       System.out.println((double)a / b);
8       System.out.println(a / (double)b);
9       System.out.println((double)a / (double)b);
10      System.out.println((double)(a / 2));
11    }
12  }
```

Código Java 4.11: TestaDivisaoInteira.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao4.zip

5 Compile e execute a classe **TestaDivisaoInteira**.

```
K19/rafael/operadores$ javac TestaDivisaoInteira.java

K19/rafael/operadores$ java TestaDivisaoInteira
2
2.5
2.5
2.5
2.0
```

Terminal 4.4: Compilando e executando a classe `TestaDivisaoInteira`

6 Na pasta **operadores**, implemente um programa em Java que realize operações de concatenação de strings.

```
1  class TestaConcatenacao {
2    public static void main(String[] args) {
3      String s1 = "K19";
4      String s2 = "Treinamentos";
5
6      System.out.println(s1 + " " + s2);
7    }
8  }
```

Código Java 4.12: TestaConcatenacao.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao6.zip

7 Compile e execute a classe **TestaConcatenacao**.

```
K19/rafael/operadores$ javac TestaConcatenacao.java

K19/rafael/operadores$ java TestaConcatenacao
K19 Treinamentos
```

Terminal 4.5: Compilando e executando a classe `TestaConcatenacao`

Operadores de Atribuição

Nos capítulos anteriores, utilizamos o principal operador de atribuição, o operador = (igual). Os outros operadores de atribuição são:

- Simples =
- Incremental +=
- Decremental -=
- Multiplicativa *=
- Divisória /=
- Modular %=
- Incremento ++
- Decremento --

```
1  int valor = 1;
2  // valor = 1
3
4  valor += 2;
5  // valor = 3
6
7  valor -= 1;
8  // valor = 2
9
10 valor *= 6;
11 // valor = 12
12
13 valor /= 3;
14 // valor = 4
15
16 valor %= 3;
17 // valor = 1
18
19 valor++;
20 // valor = 2
21
22 valor--;
23 // valor = 1
```

Código Java 4.13: Exemplo de uso dos operadores de atribuição.

As instruções acima poderiam ser escritas de outra forma:

```
1  int valor = 1;
2  // valor = 1
3
4  valor = valor + 2;
5  // valor = 3
6
7  valor = valor - 1;
8  // valor = 2
9
10 valor = valor * 6;
11 // valor = 12
12
13 valor = valor / 3;
14 // valor = 4
15
16 valor = valor % 3;
17 // valor = 1
18
19 valor = valor + 1;
20 // valor = 2
21
22 valor = valor - 1;
23 // valor = 1
```

Código Java 4.14: O mesmo exemplo anterior utilizando os operadores aritméticos.

Como podemos observar, os operadores de atribuição, exceto o simples (=), reduzem a quantidade de código escrito. Podemos dizer que esses operadores funcionam como "atalhos" para as operações que utilizam os operadores aritméticos.

Operadores Relacionais

Muitas vezes precisamos determinar a equivalência entre duas variáveis ou a relação de grandeza (se é maior ou menor) em relação à outra variável ou valor. Nessas situações, utilizamos os operadores relacionais. As operações realizadas com os operadores relacionais devolvem valores do tipo boolean em Java ou bool em C#. Os operadores relacionais são:

- Igualdade **==**
- Desigualdade **!=**
- Menor **<**
- Menor ou igual **<=**
- Maior **>**
- Maior ou igual **>=**

```
1 int valor = 2;
2 boolean t = false;
3 t = (valor == 2);   // t = true
4 t = (valor != 2);   // t = false
5 t = (valor < 2);    // t = false
6 t = (valor <= 2);   // t = true
7 t = (valor > 1);    // t = true
8 t = (valor >= 1);   // t = true
```

Código Java 4.15: Exemplo de uso dos operadores relacionais em Java.

```
1 int valor = 2;
2 bool t = false;
3 t = (valor == 2);   // t = true
4 t = (valor != 2);   // t = false
5 t = (valor < 2);    // t = false
6 t = (valor <= 2);   // t = true
7 t = (valor > 1);    // t = true
8 t = (valor >= 1);   // t = true
```

Código C# 4.6: Exemplo de uso dos operadores relacionais em C#.

Operadores Lógicos

As linguagens Java e C# permitem verificar duas condições booleanas através de operadores lógicos. Esses operadores devolvem valores do tipo **boolean** em Java ou **bool** em C#. A seguir descreveremos o funcionamento desses operadores.

- Os operadores "**&**"(E simples) e "**&&**"(E duplo) devolvem **true** se e somente se as duas condições forem **true**.

```
1 double a = Math.random();
```

```
2 double b = Math.random();
3
4 System.out.println(a > 0.2 & b < 0.8);
5 System.out.println(a > 0.2 && b < 0.8);
```

*Código Java 4.16: Exemplo de uso dos operadores **&** e **&&***

```
1 System.Random gerador = new System.Random();
2 double a = gerador.random();
3 double b = gerador.random();
4
5 System.Console.WriteLine(a > 0.2 & b < 0.8);
6 System.Console.WriteLine(a > 0.2 && b < 0.8);
```

*Código C# 4.7: Exemplo de uso dos operadores **&** e **&&***

A **tabela verdade** é uma forma prática de visualizar o resultado dos operadores lógicos. Veja a seguir a tabela verdade dos operadores **&** e **&&**.

a > 0.2	b < 0.8	a > 0.2 & b < 0.8	a > 0.2 && b < 0.8
V	V	V	V
V	F	F	F
F	V	F	F
F	F	F	F

*Figura 4.1: Tabela verdade dos operadores **&** e **&&***

- Os operadores "|"(OU simples) e "||"(OU duplo) devolvem **true** se pelo menos uma das condições for **true**.

```
1 double a = Math.random();
2 double b = Math.random();
3
4 System.out.println(a > 0.2 | b < 0.8);
5 System.out.println(a > 0.2 || b < 0.8);
```

*Código Java 4.17: Exemplo de uso dos operadores **|** e **||***

```
1 System.Random gerador = new System.Random();
2 double a = gerador.random();
3 double b = gerador.random();
4
5 System.Console.WriteLine(a > 0.2 | b < 0.8);
6 System.Console.WriteLine(a > 0.2 || b < 0.8);
```

*Código C# 4.8: Exemplo de uso dos operadores **|** e **||***

Também, podemos utilizar a **tabela verdade** para visualizar o resultado dos operadores **|** e **||**.

a > 0.2	b < 0.8	a > 0.2 | b < 0.8	a > 0.2 || b < 0.8
V	V	V	V
V	F	V	V
F	V	V	V
F	F	F	F

*Figura 4.2: Tabela verdade dos operadores **|** e **||***

- O operador "^"(OU exclusivo) devolve **true** se apenas uma das condições for **true**.

```
1 double a = Math.random();
2 double b = Math.random();
3
4 System.out.println(a > 0.2 ^ b < 0.8);
```

Código Java 4.18: Exemplo de uso do operador ^

```
1 System.Random gerador = new System.Random();
2 double a = gerador.NextDouble();
3 double b = gerador.NextDouble();
4
5 System.Console.WriteLine(a > 0.2 ^ b < 0.8);
```

Código C# 4.9: Exemplo de uso do operador ^

Vamos visualizar resultado do operador ^ através da tabela verdade.

a > 0.2	b < 0.8	a > 0.2 ^ b < 0.8
V	V	F
V	F	V
F	V	V
F	F	F

Figura 4.3: Tabela verdade do operador ^

Os operadores "**&**" e "**&&**" produzem o mesmo resultado lógico. Então, qual é a diferença entre eles? O operador "**&**" sempre avalia as duas condições. Por outro lado, o operador "**&&**" não avalia a segunda condição se o valor da primeira condição for falso. De fato, esse comportamento é plausível pois se o valor da primeira condição for falso o resultado lógico da operação é falso independentemente do valor da segunda condição. Dessa forma, podemos simplificar a tabela verdade do operador "**&&**".

a > 0.2	b < 0.8	a > 0.2 && b < 0.8
V	V	V
V	F	F
F	?	F

*Figura 4.4: Tabela verdade do operador **&&***

Analogamente, podemos deduzir a diferença entre os operadores "|" e "||". As duas condições sempre são avaliadas quando utilizamos o operador "|". Agora, quando utilizamos o operador "||" a segunda condição é avaliada somente se o valor da primeira condição for verdadeiro. Realmente, esse comportamento é aceitável pois o resultado lógico da operação é verdadeiro quando o valor da primeira condição for verdadeiro independentemente do valor da segunda condição. Dessa forma, podemos simplificar a tabela verdade do operador "||".

a > 0.2	b < 0.8	a > 0.2 || b < 0.8
V	?	V
F	V	V
F	F	F

Figura 4.5: Tabela verdade do operador ||

Pare para pensar...
Considerando o comportamento dos operadores lógicos "**&**", "**&&**", "|" e "||". O que seria exibido com as seguintes instruções?

```
int i = 10;

System.out.println(i > 100 && i++ < 500);
System.out.println(i > 0 || i++ < 500);
System.out.println(i);
```

```
int i = 10;

System.Console.WriteLine(i > 100 && i++ < 500);
System.Console.WriteLine(i > 0 || i++ < 500);
System.Console.WriteLine(i);
```

Pare para pensar...
As linguagens Java e C# possuem os operadores lógicos "**&**" e "**&&**". Também possuem os operadores "|" e "||". Agora, a pergunta que não quer calar: por que não existe o operador "^^"?

Exercícios de Fixação Com Java

8 Na pasta **operadores**, implemente um programa em Java que utilize os operadores de atribuição.

```
class TestaOperadoresDeAtribuicao {
  public static void main(String[] args) {
    int a = 1;
    System.out.println(a);

    a += 2;
    System.out.println(a);

    a -= 1;
    System.out.println(a);

    a *= 3;
    System.out.println(a);

    a /= 2;
    System.out.println(a);

```

```
18      a %= 2;
19      System.out.println(a);
20
21      a++;
22      System.out.println(a);
23
24      a--;
25      System.out.println(a);
26    }
27  }
```

Código Java 4.20: TestaOperadoresDeAtribuicao.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao8.zip

9 Compile e execute a classe **TestaOperadoresDeAtribuicao**.

```
K19/rafael/operadores$ javac TestaOperadoresDeAtribuicao.java

K19/rafael/operadores$ java TestaOperadoresDeAtribuicao
1
3
2
6
3
1
2
1
```

Terminal 4.6: Compilando e executando a classe `TestaOperadoresDeAtribuicao`

10 Na pasta **operadores**, implemente um programa em Java que utilize os operadores relacionais.

```
1  class TestaOperadoresRelacionais {
2    public static void main(String[] args) {
3      int a = 1;
4      int b = 2;
5
6      System.out.println(a > b);
7      System.out.println(a >= b);
8      System.out.println(a < b);
9      System.out.println(a <= b);
10     System.out.println(a == b);
11     System.out.println(a != b);
12   }
13 }
```

Código Java 4.21: TestaOperadoresRelacionais.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao10.zip

11 Compile e execute a classe **TestaOperadoresRelacionais**.

```
K19/rafael/operadores$ javac TestaOperadoresRelacionais.java

K19/rafael/operadores$ java TestaOperadoresRelacionais
false
false
true
true
false
true
```

Terminal 4.7: Compilando e executando a classe `TestaOperadoresRelacionais`

12 Na pasta **operadores**, implemente um programa em Java que utilize os operadores lógicos.

```
class TestaOperadoresLogicos {
  public static void main(String[] args) {
    int a = 1;
    int b = 2;
    int c = 3;
    int d = 4;

    System.out.println(a > b | c < d);
    System.out.println(a > b || c < d);
    System.out.println(a > b & c < d);
    System.out.println(a > b && c < d);
    System.out.println(a > b ^ c < d);
  }
}
```

Código Java 4.22: TestaOperadoresLogicos.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao12.zip

13 Compile e execute a classe **TestaOperadoresLogicos**.

```
K19/rafael/operadores$ javac TestaOperadoresLogicos.java

K19/rafael/operadores$ java TestaOperadoresLogicos
true
true
false
false
true
```

Terminal 4.8: Compilando e executando a classe `TestaOperadoresLogicos`

Operador ternário "?:"

Considere um programa que controla as notas dos alunos de uma escola. Para exemplificar, vamos gerar a nota de um aluno aleatoriamente.

```
double nota = Math.random();
```

```
System.Random gerador = new System.Random();
double nota = gerador.NextDouble();
```

O programa deve exibir a mensagem "aprovado" se nota de um aluno for maior ou igual a 0.5 e "reprovado" se a nota for menor do que 0.5. Esse problema pode ser resolvido com o operador ternário do Java e do C#.

Figura 4.6: Operador ternário

Quando a **condição**(**nota >= 0.5**) é verdadeira, o operador ternário devolve o **primeiro resultado**

(**"aprovado"**). Caso contrário, devolve o **segundo resultado**(**"reprovado"**). Podemos guardar o resultado do operador ternário em uma variável ou simplesmente exibi-lo.

```
String resultado = nota >= 0.5 ? "aprovado" : "reprovado";
System.out.println(nota >= 0.5 ? "aprovado" : "reprovado");
```

```
string resultado = nota >= 0.5 ? "aprovado" : "reprovado";
System.Console.WriteLine(nota >= 0.5 ? "aprovado" : "reprovado");
```

Nos exemplos anteriores, o operador ternário foi utilizado com valores do tipo **string**. Contudo, podemos utilizá-lo com qualquer tipo de valor. Veja o exemplo a seguir.

```
int i = nota >= 0.5 ? 1 : 2;
double d = nota >= 0.5 ? 0.1 : 0.2;
```

```
int i = nota >= 0.5 ? 1 : 2;
double d = nota >= 0.5 ? 0.1 : 0.2;
```

Operador "!"

Valores booleanos podem ser invertidos com o operador de "**!**" (negação). Por exemplo, podemos verificar se uma variável do tipo **double** armazena um valor maior do que 0.5 de duas formas diferentes.

```
d > 0.5
```

```
!(d <= 0.5)
```

Pré e Pós Incremento ou Pré e Pós Decremento

Os operadores "**++**" e "**--**" podem ser utilizados de duas formas diferentes, antes ou depois de uma variável numérica.

```
int i = 10;
i++;
i--;
```

```
int i = 10;
++i;
--i;
```

No primeiro exemplo, o operador "**++**" foi utilizado depois da variável **i**. Já no segundo exemplo, ele foi utilizado antes da variável **i**. A primeira forma de utilizar o operador "**++**" é chamada de **pós incremento**. A segunda é chamada de **pré incremento**. Analogamente, o operador "**--**" foi utilizado na forma de **pós decremento** no primeiro exemplo e **pré decremento** no segundo exemplo.

Mas, qual é a diferença entre **pré incremento** e **pós incremento** ou entre **pré decremento** e **pós decremento**? Vamos apresentar a diferença com alguns exemplos.

```
int i = 10;

// true
System.out.println(i++ == 10);
```

```
int i = 10;

// true
System.Console.WriteLine(i++ == 10);
```

Observe que o operador "++" foi utilizado nas expressões dos exemplos acima em conjunto com o operador "==". Como dois operadores foram utilizados na mesma expressão, você pode ter dúvida em relação a ordem de execução desses operadores. O incremento com o operador "++" será realizado antes ou depois da comparação com o operador "=="?

Como o operador "++" foi utilizado na forma de **pós incremento**, a comparação ocorrerá antes do incremento. Analogamente, a comparação ocorreria antes do decremento se o operador "--" fosse utilizado na forma de **pós decremento**.

Agora, considere a utilização do operador "++" na forma de **pré incremento**.

```
int i = 10;

// false
System.out.println(++i == 10);
```

```
int i = 10;

// false
System.Console.WriteLine(++i == 10);
```

Nesse últimos exemplos, a comparação com o operador "==" é realizada depois do incremento do operador "++". Analogamente, a comparação ocorreria depois do decremento se o operador "--" fosse utilizado na forma de **pré decremento**.

Pare para pensar...

Considere o comportamento do **pré incremento**, **pós incremento**, **pré decremento** e **pós decremento**. O que seria exibido nos exemplos abaixo?

```
int i = 10;

int j = ++i + i--;

System.out.println(j);
```

```
int i = 10;

int j = ++i + i--;

System.Console.WriteLine(j);
```

Exercícios de Fixação Com Java

14 Na pasta **operadores**, implemente um programa em Java que utilize o operador ternário.

```
1 class TestaOperadorTernario {
2   public static void main(String[] args) {
3     int a = (int)(Math.random() * 100);
4     int b = (int)(Math.random() * 100);
5
6     System.out.println(a < b ? "a < b" : "a >= b");
7   }
8 }
```

Código Java 4.33: TestaOperadorTernario.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao14.zip

15 Compile uma vez e execute várias vezes a classe **TestaOperadorTernario**. Você obterá um resultado semelhante ao apresentado a seguir.

```
K19/rafael/operadores$ javac TestaOperadorTernario.java

K19/rafael/operadores$ java TestaOperadorTernario
a < b

K19/rafael/operadores$ java TestaOperadorTernario
a >= b

K19/rafael/operadores$ java TestaOperadorTernario
a >= b

K19/rafael/operadores$ java TestaOperadorTernario
a < b

K19/rafael/operadores$ java TestaOperadorTernario
a < b
```

Terminal 4.9: Compilando e executando a classe `TestaOperadorTernario`

16 Na pasta **operadores**, implemente um programa em Java que utilize o operador de negação.

```
1 class TestaOperadorNegacao {
2   public static void main(String[] args) {
3     int a = 10;
4     int b = 20;
5
6     System.out.println(!(a < b));
7   }
8 }
```

Código Java 4.34: TestaOperadorNegacao.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao16.zip

17 Compile e execute a classe **TestaOperadorNegacao**.

```
K19/rafael/operadores$ javac TestaOperadorNegacao.java
```

```
K19/rafael/operadores$ java TestaOperadorNegacao
false
```

Terminal 4.10: Compilando e executando a classe `TestaOperadorNegacao`

18 Na pasta **operadores**, implemente um programa em Java que utilize o operador "**++**" na forma de pré e pós incremento. Analogamente, utilize o "**--**" na forma de pré e pós decremento.

```
1  class TestaPrePosIncrementoDecremento {
2    public static void main(String[] args) {
3      int a = 1;
4
5      System.out.println(a++);
6      System.out.println(++a);
7      System.out.println(a--);
8      System.out.println(--a);
9    }
10 }
```

Código Java 4.35: TestaPrePosIncrementoDecremento.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao18.zip

19 Compile e execute a classe **TestaPrePosIncrementoDecremento**.

```
K19/rafael/operadores$ javac TestaPrePosIncrementoDecremento.java

K19/rafael/operadores$ java TestaPrePosIncrementoDecremento
1
3
3
1
```

Terminal 4.11: Compilando e executando a classe `TestaPrePosIncrementoDecremento`

Operações com Strings

Algumas operações são específicas para valores do tipo **string**. A seguir, apresentaremos algumas dessas operações.

- Descobrir a quantidade de caracteres de uma string.

```
1 String s = "Rafael Cosentino";
2
3 int length = s.length();
4
5 System.out.println(length);
```

```
1 string s = "Rafael Cosentino";
2
3 int length = s.Length;
4
5 System.Console.WriteLine(length);
```

- Recuperar um caractere de acordo com a sua posição na string.

```java
String s = "Rafael Cosentino";

char c = s.charAt(0);

System.out.println(c);
```

```c#
string s = "Rafael Cosentino";

char c = s[0];

System.Console.WriteLine(c);
```

- Podemos verificar se uma determinada sequência de caracteres está contida em uma string.

```java
String s = "K11 - Orientação a Objetos em Java";

boolean resultado1 = s.contains("Java");
boolean resultado2 = s.contains("C#");

// true
System.out.println(resultado1);

// false
System.out.println(resultado2);
```

```c#
string s = "K31 - C# e Orientação a Objetos";

bool resultado1 = s.Contains("Java");
bool resultado2 = s.Contains("C#");

// True
System.Console.WriteLine(resultado1);

// False
System.Console.WriteLine(resultado2);
```

- Podemos verificar se uma string termina com uma determinada sequência de caracteres.

```java
String s = "K11 - Orientação a Objetos em Java";

boolean resultado1 = s.endsWith("Java");
boolean resultado2 = s.endsWith("Objetos");

// true
System.out.println(resultado1);

// false
System.out.println(resultado2);
```

```c#
string s = "K31 - C# e Orientação a Objetos";

bool resultado1 = s.EndsWith("C#");
bool resultado2 = s.EndsWith("Objetos");

// False
System.Console.WriteLine(resultado1);

// True
System.Console.WriteLine(resultado2);
```

- Podemos verificar se uma string começa com uma determinada sequência de caracteres.

```java
String s = "K11 - Orientação a Objetos em Java";
```

```
2
3  boolean resultado1 = s.startsWith("Java");
4  boolean resultado2 = s.startsWith("K11");
5
6  // false
7  System.out.println(resultado1);
8
9  // true
10 System.out.println(resultado2);
```

```
1  string s = "K31 - C# e Orientação a Objetos";
2
3  bool resultado1 = s.StartsWith("C#");
4  bool resultado2 = s.StartsWith("K31");
5
6  // False
7  System.Console.WriteLine(resultado1);
8
9  // True
10 System.Console.WriteLine(resultado2);
```

- Podemos realizar substituições em uma string.

```
1  String s1 = "K19 - Treinamentos";
2
3  String s2 = s1.replaceAll("Treinamentos", "Cursos");
4
5  // K19 - Cursos
6  System.out.println(s2);
```

```
1  string s1 = "K19 - Treinamentos";
2
3  string s2 = s1.Replace("Treinamentos", "Cursos");
4
5  // K19 - Cursos
6  System.Console.WriteLine(s2);
```

- Podemos extrair um trecho de uma string.

```
1  String s1 = "Rafael Cosentino";
2
3  String s2 = s1.substring(7);
4  String s3 = s1.substring(0, 6);
5
6  // Cosentino
7  System.out.println(s2);
8
9  // Rafael
10 System.out.println(s3);
```

```
1  string s1 = "Rafael Cosentino";
2
3  string s2 = s1.Substring(7);
4  string s3 = s1.Substring(0, 6);
5
6  // Cosentino
7  System.Console.WriteLine(s2);
8
9  // Rafael
10 System.Console.WriteLine(s3);
```

- Podemos transformar em maiúsculas todas as letras contidas em uma string.

```
String s1 = "Rafael Cosentino";

String s2 = s1.toUpperCase();

// RAFAEL COSENTINO
System.out.println(s2);
```

```
string s1 = "Rafael Cosentino";

string s2 = s1.ToUpper();

// RAFAEL COSENTINO
System.Console.WriteLine(s2);
```

- Podemos transformar em minúsculas todas as letras contidas em uma string.

```
String s1 = "Rafael Cosentino";

String s2 = s1.toLowerCase();

// rafael cosentino
System.out.println(s2);
```

```
string s1 = "Rafael Cosentino";

string s2 = s1.ToLower();

// rafael cosentino
System.Console.WriteLine(s2);
```

- Podemos eliminar os espaços em branco do começo e do término de uma string.

```
String s1 = "     Rafael Cosentino      ";

String s2 = s1.trim();

// "Rafael Cosentino"
System.out.println(s2);
```

```
string s1 = "     Rafael Cosentino      ";

string s2 = s1.Trim();

// "Rafael Cosentino"
System.Console.WriteLine(s2);
```

Operações com Data e Hora (Conteúdo Extra)

Algumas operações são específicas para data e hora. A seguir, apresentaremos algumas dessas operações.

- Podemos modificar uma data e hora acrescentando ou subtraindo uma quantidade nos campos que definem essa data e hora.

```
java.util.Calendar c = new java.util.GregorianCalendar(2010, 7, 27);

// Acrescentando 140 dias
c.add(java.util.Calendar.DAY_OF_MONTH, 140);
```

```
5
6  // Subtraindo 2 anos
7  c.add(java.util.Calendar.YEAR, -2);
8
9  // Acrescentando 20 segundos
10 c.add(java.util.Calendar.SECOND, 20);
```

```
1  System.DateTime dt = new System.DateTime(2010, 8, 27);
2
3  // Acrescentando 140 dias
4  dt = dt.AddDays(140);
5
6  // Subtraindo 2 anos
7  dt = dt.AddYears(-2);
8
9  // Acrescentando 20 segundos
10 dt = dt.AddSeconds(20);
```

Observe, nos exemplos acima, que 140 dias foram adicionados a data "27 de Agosto de 2010". Automaticamente, o mês e o ano serão atualizados e a data passará a ser "14 de Janeiro de 2009".

- Podemos comparar a ordem das datas e horas.

```
1  java.util.Calendar c1 = new java.util.GregorianCalendar(2010, 7, 27);
2  java.util.Calendar c2 = java.util.Calendar.getInstance();
3
4  // true
5  System.out.println(c1.before(c2));
6
7  // false
8  System.out.println(c1.after(c2));
```

```
1  System.DateTime dt1 = new System.DateTime(2010, 8, 27);
2  System.DateTime dt2 = System.DateTime.Now;
3
4  // True
5  System.Console.WriteLine(dt1 < dt2);
6
7  // False
8  System.Console.WriteLine(dt1 > dt2);
```

Exercícios de Fixação Com Java

20 Na pasta **operadores**, implemente um programa em Java que utilize as principais operações de strings.

```
1  class TestaOperacoesString {
2    public static void main(String[] args) {
3      String s = "Rafael Cosentino";
4
5      System.out.println(s.charAt(7));
6
7      System.out.println(s.contains("Cosentino"));
8      System.out.println(s.contains("Hirata"));
9
10     System.out.println(s.endsWith("Cosentino"));
11     System.out.println(s.endsWith("Hirata"));
12
```

```
13      System.out.println(s.startsWith("Rafael"));
14      System.out.println(s.startsWith("Marcelo"));
15
16      s = s.replaceAll("Rafael", "Jonas");
17      System.out.println(s);
18
19      System.out.println(s.substring(6));
20      System.out.println(s.substring(0, 5));
21
22      s = s.toUpperCase();
23      System.out.println(s);
24
25      s = s.toLowerCase();
26      System.out.println(s);
27
28      s = "     K19 Treinamentos     ";
29      System.out.println(s.trim());
30    }
31  }
```

Código Java 4.48: TestaOperacoesString.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao20.zip

21 Compile e execute a classe **TestaOperacoesString**.

```
K19/rafael/operadores$ javac TestaOperacoesString.java

K19/rafael/operadores$ java TestaOperacoesString
C
true
false
true
false
true
false
Jonas Cosentino
Cosentino
Jonas
JONAS COSENTINO
jonas cosentino
K19 Treinamentos
```

Terminal 4.12: Compilando e executando a classe `TestaOperacoesString`

Erro: Utilizar operadores incompatíveis

Um erro de compilação comum em Java ou C# ocorre quando um operador é aplicado a valores incompatíveis. Veja alguns exemplos de programas em Java com esse problema.

```
1  class Programa {
2    public static void main(String[] args) {
3      String s1 = "K19";
4      String s2 = "Treinamentos";
5
6      System.out.println(s1 - s2);
7    }
8  }
```

Código Java 4.49: Programa.java

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.java:6: error: bad operand types for binary operator '-'
    System.out.println(s1 - s2);
                       ^
  first type:  String
  second type: String
1 error
```

Terminal 4.13: Erro de compilação

```
1 class Programa {
2   public static void main(String[] args) {
3     boolean b1 = true;
4     boolean b2 = false;
5
6     System.out.println(b1 > b2);
7   }
8 }
```

Código Java 4.50: Programa.java

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.java:6: error: bad operand types for binary operator '>'
    System.out.println(b1 > b2);
                       ^
  first type:  boolean
  second type: boolean
1 error
```

Terminal 4.14: Erro de compilação

```
1 class Programa {
2   public static void main(String[] args) {
3     int i = 1;
4
5     System.out.println(!i);
6   }
7 }
```

Código Java 4.51: Programa.java

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.java:5: error: bad operand type int for unary operator '!'
    System.out.println(!i);
                       ^
1 error
```

Terminal 4.15: Erro de compilação

Agora, veja alguns exemplos de programas em C# com esse problema.

```
1  class Programa
2  {
3    static void Main()
4    {
5      string s1 = "K19";
6      string s2 = "Treinamentos";
7
8      System.Console.WriteLine(s1 - s2);
9    }
10 }
```

Código C# 4.29: Programa.cs

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.cs(8,28): error CS0019: Operator '-' cannot be applied to operands of type 'string' and 'string'
```

Terminal 4.16: Erro de compilação

```
1  class Programa
2  {
3    static void Main()
4    {
5      bool b1 = true;
6      bool b2 = false;
7
8      System.Console.WriteLine(b1 > b2);
9    }
10 }
```

Código C# 4.30: Programa.cs

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.cs(8,28): error CS0019: Operator '>' cannot be applied to operands of type 'bool' and 'bool'
```

Terminal 4.17: Erro de compilação

```
1  class Programa
2  {
3    static void Main()
4    {
5      int i = 1;
6
7      System.Console.WriteLine(!i);
8    }
9  }
```

Código C# 4.31: Programa.cs

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.cs(7,28): error CS0023: The '!' operator cannot be applied to operand of type 'int'
```

Terminal 4.18: Erro de compilação

Exercícios de Fixação Com C#

22 Na pasta **operadores**, implemente um programa em C# que utilize os operadores aritméticos.

```
1  class TestaOperadoresAritmeticos
2  {
3    static void Main()
4    {
5      int a = 1 + 1;
6      int b = 10 - 2;
7      int c = 2 * 3;
8      int d = 25 / 5;
9      int e = 10 % 4;
10
11     System.Console.WriteLine(a);
```

```
12      System.Console.WriteLine(b);
13      System.Console.WriteLine(c);
14      System.Console.WriteLine(d);
15      System.Console.WriteLine(e);
16    }
17  }
```

Código C# 4.32: TestaOperadoresAritmeticos.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao22.zip

23 Compile e execute a classe **TestaOperadoresAritmeticos**.

```
C:\Users\K19\rafael\operadores> csc TestaOperadoresAritmeticos.cs

C:\Users\K19\rafael\operadores> TestaOperadoresAritmeticos.exe
2
8
6
5
2
```

Terminal 4.19: Compilando e executando a classe `TestaOperadoresAritmeticos`

24 Na pasta **operadores**, implemente um programa em C# que realize operações de divisão inteira e de casting.

```
1  class TestaDivisaoInteira
2  {
3    static void Main()
4    {
5      int a = 5;
6      int b = 2;
7
8      System.Console.WriteLine(a / b);
9      System.Console.WriteLine((double)a / b);
10     System.Console.WriteLine(a / (double)b);
11     System.Console.WriteLine((double)a / (double)b);
12     System.Console.WriteLine((double)(a / 2));
13   }
14 }
```

Código C# 4.33: TestaDivisaoInteira.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao24.zip

25 Compile e execute a classe **TestaDivisaoInteira**.

```
C:\Users\K19\rafael\operadores> csc TestaDivisaoInteira.cs

C:\Users\K19\rafael\operadores> TestaDivisaoInteira.exe
2
2.5
2.5
2.5
2
```

Terminal 4.20: Compilando e executando a classe `TestaDivisaoInteira`

26 Na pasta **operadores**, implemente um programa em C# que realize operações de concatenação

de strings.

```
class TestaConcatenacao
{
  static void Main()
  {
    string s1 = "K19";
    string s2 = "Treinamentos";

    System.Console.WriteLine(s1 + " " + s2);
  }
}
```

Código C# 4.34: TestaConcatenacao.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao26.zip

27 Compile e execute a classe **TestaConcatenacao**.

```
C:\Users\K19\rafael\operadores> csc TestaConcatenacao.cs

C:\Users\K19\rafael\operadores> TestaConcatenacao.exe
K19 Treinamentos
```

Terminal 4.21: Compilando e executando a classe `TestaConcatenacao`

28 Na pasta **operadores**, implemente um programa em C# que utilize os operadores de atribuição.

```
class TestaOperadoresDeAtribuicao
{
  static void Main()
  {
    int a = 1;
    System.Console.WriteLine(a);

    a += 2;
    System.Console.WriteLine(a);

    a -= 1;
    System.Console.WriteLine(a);

    a *= 3;
    System.Console.WriteLine(a);

    a /= 2;
    System.Console.WriteLine(a);

    a %= 2;
    System.Console.WriteLine(a);
  }
}
```

Código C# 4.35: TestaOperadoresDeAtribuicao.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao28.zip

29 Compile e execute a classe **TestaOperadoresDeAtribuicao**.

```
C:\Users\K19\rafael\operadores> csc TestaOperadoresDeAtribuicao.cs

C:\Users\K19\rafael\operadores> TestaOperadoresDeAtribuicao.exe
```

```
1
3
2
6
3
1
```

Terminal 4.22: Compilando e executando a classe `TestaOperadoresDeAtribuicao`

30 Na pasta **operadores**, implemente um programa em C# que utilize os operadores relacionais.

```
class TestaOperadoresRelacionais
{
  static void Main()
  {
    int a = 1;
    int b = 2;

    System.Console.WriteLine(a > b);
    System.Console.WriteLine(a >= b);
    System.Console.WriteLine(a < b);
    System.Console.WriteLine(a <= b);
    System.Console.WriteLine(a == b);
    System.Console.WriteLine(a != b);
  }
}
```

Código C# 4.36: TestaOperadoresRelacionais.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao30.zip

31 Compile e execute a classe **TestaOperadoresRelacionais**.

```
C:\Users\K19\rafael\operadores> csc TestaOperadoresRelacionais.cs

C:\Users\K19\rafael\operadores> TestaOperadoresRelacionais.exe
False
False
True
True
False
True
```

Terminal 4.23: Compilando e executando a classe `TestaOperadoresRelacionais`

32 Na pasta **operadores**, implemente um programa em C# que utilize os operadores lógicos.

```
class TestaOperadoresLogicos
{
  static void Main()
  {
    int a = 1;
    int b = 2;
    int c = 3;
    int d = 4;

    System.Console.WriteLine(a > b | c < d);
    System.Console.WriteLine(a > b || c < d);
    System.Console.WriteLine(a > b & c < d);
    System.Console.WriteLine(a > b && c < d);
    System.Console.WriteLine(a > b ^ c < d);
  }
}
```

Código C# 4.37: TestaOperadoresLogicos.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao32.zip

33 Compile e execute a classe **TestaOperadoresLogicos**.

```
C:\Users\K19\rafael\operadores> csc TestaOperadoresLogicos.cs

C:\Users\K19\rafael\operadores> TestaOperadoresLogicos.exe
True
True
False
False
True
```

Terminal 4.24: Compilando e executando a classe `TestaOperadoresLogicos`

34 Na pasta **operadores**, implemente um programa em C# que utilize o operador ternário.

```
1  class TestaOperadorTernario
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6      int a = (int)(gerador.NextDouble() * 100);
7      int b = (int)(gerador.NextDouble() * 100);
8
9      System.Console.WriteLine(a < b ? "a < b" : "a >= b");
10   }
11 }
```

Código C# 4.38: TestaOperadorTernario.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao34.zip

35 Compile uma vez e execute várias vezes a classe **TestaOperadorTernario**. Você obterá um resultado semelhante ao apresentado a seguir.

```
C:\Users\K19\rafael\operadores> csc TestaOperadorTernario.cs

C:\Users\K19\rafael\operadores> TestaOperadorTernario.exe
a < b

C:\Users\K19\rafael\operadores> TestaOperadorTernario.exe
a >= b

C:\Users\K19\rafael\operadores> TestaOperadorTernario.exe
a >= b

C:\Users\K19\rafael\operadores> TestaOperadorTernario.exe
a < b

C:\Users\K19\rafael\operadores> TestaOperadorTernario.exe
a < b
```

Terminal 4.25: Compilando e executando a classe `TestaOperadorTernario`

36 Na pasta **operadores**, implemente um programa em C# que utilize o operador de negação.

```
1  class TestaOperadorNegacao
2  {
3    static void Main()
4    {
5      int a = 10;
6      int b = 20;
7
8      System.Console.WriteLine(!(a < b));
9    }
10 }
```

Código C# 4.39: TestaOperadorNegacao.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao36.zip

37 Compile e execute a classe **TestaOperadorNegacao**.

```
C:\Users\K19\rafael\operadores> csc TestaOperadorNegacao.cs

C:\Users\K19\rafael\operadores> TestaOperadorNegacao.exe
False
```

Terminal 4.26: Compilando e executando a classe `TestaOperadorNegacao`

38 Na pasta **operadores**, implemente um programa em C# que utilize o operador "**++**" na forma de pré e pós incremento. Analogamente, utilize o "**--**" na forma de pré e pós decremento.

```
1  class TestaPrePosIncrementoDecremento
2  {
3    static void Main()
4    {
5      int a = 1;
6
7      System.Console.WriteLine(a++);
8      System.Console.WriteLine(++a);
9      System.Console.WriteLine(a--);
10     System.Console.WriteLine(--a);
11   }
12 }
```

Código C# 4.40: TestaPrePosIncrementoDecremento.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao38.zip

39 Compile e execute a classe **TestaPrePosIncrementoDecremento**.

```
C:\Users\K19\rafael\operadores> csc TestaPrePosIncrementoDecremento.cs

C:\Users\K19\rafael\operadores> TestaPrePosIncrementoDecremento.exe
1
3
3
1
```

Terminal 4.27: Compilando e executando a classe `TestaPrePosIncrementoDecremento`

40 Na pasta **operadores**, implemente um programa em C# que utilize as principais operações de strings.

```
class TestaOperacoesString
{
  static void Main()
  {
    string s = "Rafael Cosentino";

    System.Console.WriteLine(s[7]);

    System.Console.WriteLine(s.Contains("Cosentino"));
    System.Console.WriteLine(s.Contains("Hirata"));

    System.Console.WriteLine(s.EndsWith("Cosentino"));
    System.Console.WriteLine(s.EndsWith("Hirata"));

    System.Console.WriteLine(s.StartsWith("Rafael"));
    System.Console.WriteLine(s.StartsWith("Marcelo"));

    s = s.Replace("Rafael", "Jonas");
    System.Console.WriteLine(s);

    System.Console.WriteLine(s.Substring(6));
    System.Console.WriteLine(s.Substring(0, 5));

    s = s.ToUpper();
    System.Console.WriteLine(s);

    s = s.ToLower();
    System.Console.WriteLine(s);

    s = "     K19 Treinamentos     ";
    System.Console.WriteLine(s.Trim());
  }
}
```

Código C# 4.41: TestaOperacoesString.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-fixacao40.zip

41 Compile e execute a classe **TestaOperacoesString**.

```
C:\Users\K19\rafael\operadores> csc TestaOperacoesString.cs

C:\Users\K19\rafael\operadores> TestaOperacoesString.exe
C
True
False
True
False
True
False
Jonas Cosentino
Cosentino
Jonas
JONAS COSENTINO
jonas cosentino
K19 Treinamentos
```

Terminal 4.28: Compilando e executando a classe `TestaOperacoesString`

Exercícios Complementares

1 Na pasta **operadores**, crie uma classe Java chamada **UseOperadoresAritmeticos**. Complete o

código a seguir com os operadores aritméticos: **+**, **-**, *****, **/** e **%**. O programa deve exibir os números **11**, **4**, **12**, **4** e **1** nessa ordem.

```
class UseOperadoresAritmeticos {
  public static void main(String[] args) {
    int x = 3    8;
    int y = 7    3;
    int z = 4    3;
    int q = 8    2;
    int w = 9    4;

    System.out.println(x);
    System.out.println(y);
    System.out.println(z);
    System.out.println(q);
    System.out.println(w);
  }
}
```

Código Java 4.52: UseOperadoresAritmeticos.java

2 Utilizando os operadores aritméticos, crie um programa em Java que mostre a idade média de três pessoas: Rafael Cosentino = 27; Jonas Hirata = 29; Marcelo Martins = 27.

3 Na pasta **operadores**, crie uma classe Java chamada **UseDivisaoCasting**. Complete o código a seguir com operações de divisão e operações de casting. O programa deve exibir os números **20** e **20.5** nessa ordem. Utilize as variáveis **x** e **y**.

```
class UseDivisaoCasting {
  public static void main(String[] args) {
    int x = 41;
    int y = 2;

    System.out.println(                    );
    System.out.println(                    );
  }
}
```

Código Java 4.55: UseDivisaoCasting.java

4 Na pasta **operadores**, crie uma classe Java chamada **UseConcatenacao**. Complete o código a seguir com operações de concatenação. O programa deve exibir as strings "**Rafael Cosentino**", "**Jonas Hirata**" e "**Marcelo Martins**" nessa ordem. Utilize as variáveis **s1**, **s2**, **s3**, **s4**, **s5** e **s6**.

```
class UseConcatenacao {
  public static void main(String[] args) {
    String s1 = "Rafael";
    String s2 = "Jonas";
    String s3 = "Marcelo";
    String s4 = "Cosentino";
    String s5 = "Hirata";
    String s6 = "Martins";

    System.out.println(              );
    System.out.println(              );
    System.out.println(              );
  }
}
```

Código Java 4.57: UseConcatenacao.java

5 Na pasta **operadores**, crie uma classe Java chamada **UseOperadoresAtribuicao**. Complete o código a seguir com operadores de atribuição: =, +=, -=, *=, /=, %=, ++ e --. O programa deve exibir os números **5**, **15**, **12**, **48**, **6**, **1**, **2** e **1** nessa ordem.

```
1  class UseOperadoresAtribuicao {
2    public static void main(String[] args) {
3      int x      5;
4      System.out.println(x);
5      x      10;
6      System.out.println(x);
7      x      3;
8      System.out.println(x);
9      x      4;
10     System.out.println(x);
11     x      8;
12     System.out.println(x);
13     x      5;
14     System.out.println(x);
15     x      ;
16     System.out.println(x);
17     x      ;
18     System.out.println(x);
19   }
20 }
```

Código Java 4.59: UseOperadoresAtribuicao.java

6 Crie um programa em Java que aplique os operadores de atribuição no seguinte telefone: 2387-3791. Esses operadores devem ser aplicados na seguinte ordem: incrementação, decrementação, divisão, multiplicação, módulo e incrementação. Inicie com o valor "23", e a cada dígito do telefone, utilize um operador.

7 Na pasta **operadores**, crie uma classe Java chamada **UseOperadoresRelacionais**. Complete o código a seguir com operadores relacionais: >, <, >=, <=, == e **!=**. O programa deve exibir os valores **true**, **true**, **false**, **false**, **false** e **true** nessa ordem.

```
1  class UseOperadoresRelacionais {
2    public static void main(String[] args) {
3      int x = 20;
4      int y = 15;
5
6      System.out.println(x      y);
7      System.out.println(x      y);
8      System.out.println(x      y);
9      System.out.println(x      y);
10     System.out.println(x      y);
11     System.out.println(x      y);
12   }
13 }
```

Código Java 4.62: UseOperadoresRelacionais.java

8 Crie um programa em Java que verifica se o valor do ano do atentado das Torres Gêmeas somado

com 19 e dividido por 4 é maior ou igual que o valor do ano em que o cantor Michael Jackson faleceu somado com 129 e dividido por 5.

9 Na pasta **operadores**, crie uma classe Java chamada **UseOperadoresLogicos**. Complete o código a seguir com os operadores lógicos: ^, **&**, **&&**, | e ||. O programa deve exibir os valores **true**, **false**, **true**, **false** e **false** sejam exibidos nessa ordem. Não utilize operadores repetidos.

```
class UseOperadoresLogicos {
  public static void main(String[] args) {
    int q = 10;
    int w = 5;
    int e = 8;
    int r = 11;

    System.out.println(q > w      e < r);
    System.out.println(q > r      e < w);
    System.out.println(q > e      w < r);
    System.out.println(q > w      r < e);
    System.out.println(q > w      e < r);
  }
}
```

Código Java 4.65: UseOperadoresLogicos.java

10 Na pasta **operadores**, crie uma classe Java chamada **UseTernarioNegacaoIncrementoDecremento**. Complete o código a seguir utilizando o operador ternário, o operador de negação e os operadores "**++**" e "**--**". O programa deve exibir **8**, "**Marcelo**", **9** e **10** nessa ordem. Não é necessário preencher todas as caixas.

```
class UseTernarioNegacaoIncrementoDecremento {
  public static void main(String[] args) {
    int a = 10;
    int b = 8;

    System.out.println(  (a < b)    a    b);
    System.out.println(  (a < b)    "Marcelo"    "Jonas");
    System.out.println(  (a < b)    a      b);
    System.out.println(  (  a == b)    a    b + 1);
  }
}
```

Código Java 4.67: UseTernarioNegacaoIncrementoDecremento.java

11 Na pasta **operadores**, crie um programa em Java que exiba as datas de vencimento das parcelas de uma casa. A data de vencimento da primeira parcela é "15 de Agosto de 2015". As outras três parcelas devem vencer exatamente 30, 60 e 90 dias após a primeira.

12 Na pasta **operadores**, crie uma classe C# chamada **UseOperadoresAritmeticos**. Complete o código a seguir com os operadores aritméticos: **+**, **-**, *****, **/** e **%**. O programa deve exibir os números **11**, **4**, **12**, **4** e **1** nessa ordem.

```
class UseOperadoresAritmeticos
{
  static void Main()
  {
    int x = 3    8;
```

```
6     int y = 7    3;
7     int z = 4    3;
8     int q = 8    2;
9     int w = 9    4;
10
11    System.Console.WriteLine(x);
12    System.Console.WriteLine(y);
13    System.Console.WriteLine(z);
14    System.Console.WriteLine(q);
15    System.Console.WriteLine(w);
16  }
17 }
```

Código C# 4.42: UseOperadoresAritmeticos.cs

13 Utilizando os operadores aritméticos, crie um programa em C# que mostre a idade média de três pessoas: Rafael Cosentino = 27; Jonas Hirata = 29; Marcelo Martins = 27.

14 Na pasta **operadores**, crie uma classe C# chamada **UseDivisaoCasting**. Complete o código a seguir com operações de divisão e operações de casting. O programa deve exibir os números **20** e **20.5** nessa ordem. Utilize as variáveis **x** e **y**.

```
1  class UseDivisaoCasting
2  {
3    static void Main()
4    {
5      int x = 41;
6      int y = 2;
7
8      System.Console.WriteLine(                    );
9      System.Console.WriteLine(                    );
10   }
11 }
```

Código C# 4.45: UseDivisaoCasting.cs

15 Na pasta **operadores**, crie uma classe C# chamada **UseConcatenacao**. Complete o código a seguir com operações de concatenação. O programa deve exibir as strings "**Rafael Cosentino**", "**Jonas Hirata**" e "**Marcelo Martins**" nessa ordem. Utilize as variáveis **s1**, **s2**, **s3**, **s4**, **s5** e **s6**.

```
1  class UseConcatenacao
2  {
3    static void Main()
4    {
5      string s1 = "Rafael";
6      string s2 = "Jonas";
7      string s3 = "Marcelo";
8      string s4 = "Cosentino";
9      string s5 = "Hirata";
10     string s6 = "Martins";
11
12     System.Console.WriteLine(                );
13     System.Console.WriteLine(                );
14     System.Console.WriteLine(                );
15   }
16 }
```

Código C# 4.47: UseConcatenacao.cs

16 Na pasta **operadores**, crie uma classe C# chamada **UseOperadoresAtribuicao**. Complete o código a seguir com operadores de atribuição: **=**, **+=**, **-=**, ***=**, **/=**, **%=**, **++** e **--**. O programa deve exibir os números **5**, **15**, **12**, **48**, **6**, **1**, **2** e **1** nessa ordem.

```
class UseOperadoresAtribuicao
{
  static void Main()
  {
    int x      5;
    System.Console.WriteLine(x);
    x      10;
    System.Console.WriteLine(x);
    x      3;
    System.Console.WriteLine(x);
    x      4;
    System.Console.WriteLine(x);
    x      8;
    System.Console.WriteLine(x);
    x      5;
    System.Console.WriteLine(x);
    x      ;
    System.Console.WriteLine(x);
    x      ;
    System.Console.WriteLine(x);
  }
}
```

Código C# 4.48: UseOperadoresAtribuicao.cs

17 Crie um programa em Java que aplique os operadores de atribuição no seguinte telefone: 2387-3791. Esses operadores devem ser aplicados na seguinte ordem: incrementação, decrementação, divisão, multiplicação, módulo e incrementação. Inicie com o valor "23", e a cada dígito do telefone, utilize um operador.

18 Na pasta **operadores**, crie uma classe C# chamada **UseOperadoresRelacionais**. Complete o código a seguir com operadores relacionais: **>**, **<**, **>=**, **<=**, **==** e **!=**. O programa deve exibir os valores **True**, **True**, **False**, **False**, **False** e **True** nessa ordem.

```
class UseOperadoresRelacionais
{
  static void Main()
  {
    int x = 20;
    int y = 15;

    System.Console.WriteLine(x      y);
    System.Console.WriteLine(x      y);
    System.Console.WriteLine(x      y);
    System.Console.WriteLine(x      y);
    System.Console.WriteLine(x      y);
    System.Console.WriteLine(x      y);
  }
}
```

Código C# 4.51: UseOperadoresRelacionais.cs

19 Crie um programa em C# que verifica se o valor do ano do atentado das Torres Gêmeas somado com 19 e dividido por 4 é maior ou igual que o valor do ano em que o cantor Michael Jackson faleceu

somado com 129 e dividido por 5.

20 Na pasta **operadores**, crie uma classe C# chamada **UseOperadoresLogicos**. Complete o código a seguir com os operadores lógicos: ^, **&**, **&&**, | e ||. O programa deve exibir os valores **True**, **False**, **True**, **False** e **False** sejam exibidos nessa ordem. Não utilize operadores repetidos.

```
class UseOperadoresLogicos
{
  static void Main()
  {
    int q = 10;
    int w = 5;
    int e = 8;
    int r = 11;

    System.Console.WriteLine(q > w      e < r);
    System.Console.WriteLine(q > r      e < w);
    System.Console.WriteLine(q > e      w < r);
    System.Console.WriteLine(q > w      r < e);
    System.Console.WriteLine(q > w      e < r);
  }
}
```

Código C# 4.54: UseOperadoresLogicos.cs

21 Na pasta **operadores**, crie uma classe C# chamada **UseTernarioNegacaoIncrementoDecremento**. Complete o código a seguir utilizando o operador ternário, o operador de negação e os operadores "**++**" e "**--**". O programa deve exibir **8**, "**Marcelo**", **9** e **10** nessa ordem. Não é necessário preencher todas as caixas.

```
class UseTernarioNegacaoIncrementoDecremento
{
  static void Main()
  {
    int a = 10;
    int b = 8;

    System.Console.WriteLine(  (a < b)    a    b);
    System.Console.WriteLine(  (a < b)    "Marcelo"    "Jonas");
    System.Console.WriteLine(  (a < b)    a      b);
    System.Console.WriteLine(  (  a == b)    a    b + 1);
  }
}
```

Código C# 4.56: UseTernarioNegacaoIncrementoDecremento.cs

22 Na pasta **operadores**, crie um programa em C# que exiba as datas de vencimento das parcelas de uma casa. A data de vencimento da primeira parcela é "15 de Agosto de 2015". As outras três parcelas devem vencer exatamente 30, 60 e 90 dias após a primeira.

Resumo do Capítulo

1 Os **operadores** são utilizados para **manipular** os **valores** armazenados nas variáveis ou valores

literais.

2 As **operações aritméticas** de soma, subtração, multiplicação, divisão e resto são realizadas respectivamente através dos operadores: `+ - * / %`

3 A **divisão** entre valores **inteiros** desconsidera a parte **fracionária** do resultado.

4 O operador + também é utilizado para realizar a **concatenação** de strings.

5 O conteúdo de uma variável pode ser modificado através dos operadores de atribuição: `= += -= *= /= %= ++ --`.

6 Podemos **comparar** o conteúdo das variáveis ou os valores literais através dos operadores relacionais: `== != < <= > >=`.

7 Operadores relacionais devolvem valores booleanos.

8 As operações lógicas E, OU e OU EXCLUSIVO são realizadas através dos operadores: `& | ^ && ||`.

9 O primeiro argumento do operador ternário **?:** deve ser um valor booleano.

10 O operador de negação **!** inverte os valores booleanos.

11 O operador "++" pode ser utilizado na forma de pré e pós incremento.

12 O operador "--" pode ser utilizado na forma de pré e pós decremento.

Prova

1 Qual é o resultado da operação abaixo?

5%2

a) 2

b) 2.5

c) 0.1

d) 1

e) 5.1

2 Quais são os resultados das operações abaixo?

```
5/2
5.0/2
(double)5/2
```

a) 2, 2 e 2

b) 2.5, 2.5 e 2.5

c) 2, 2.5 e 2.5

d) 2, 2 e 2.5

e) 2, 2.5 e 2

3 Quais são os resultados das operações baixo?

```
(double)(5/2)
(double)5/2
5/(double)2
```

a) 2.5, 2.5 e 2.5

b) 2, 2 e 2

c) 2, 2 e 2.5

d) 2.5, 2.5 e 2

e) 2, 2.5 e 2.5

4 Qual é o resultado da operação abaixo?

```
1 + 2 + "rafael" + 3 + 4
```

a) 12rafael34

b) 3rafael34

c) 3rafael7

d) 12rafael7

e) ocorrerá um erro

5 Qual é o valor armazenado na variável **i** depois das seguintes operações?

```
int i = 10;
i++;
i += 10;
--i;
i %= 3;
```

a) 2

b) 10

c) 3

d) 5

e) 0

6 Qual é o tipo dos valores devolvidos pelos operadores relacionais?

a) números inteiros

b) números reais

c) caracteres

d) string

e) booleanos

7 Quais são os resultados das operações baixo?

```
10 > 5 & 7 < 10
10 > 5 & 7 > 10
10 < 5 & 7 < 10
10 < 5 & 7 > 10
```

a) true, true, true e true

b) true, true, true e false

c) true, false, true e false

d) true, false, false e false

e) false, true, true e false

8 Quais são os resultados das operações abaixo?

```
10 > 5 | 7 < 10
10 > 5 | 7 > 10
10 < 5 | 7 < 10
10 < 5 | 7 > 10
```

a) true, true, true e true

b) true, true, true e false

c) true, false, true e false

d) true, false, false e false

e) false, true, true e false

9 Quais são os resultados das operações abaixo?

```
10 > 5 ^ 7 < 10
10 > 5 ^ 7 > 10
10 < 5 ^ 7 < 10
10 < 5 ^ 7 > 10
```

a) true, true, true e true

b) true, true, true e false

c) true, false, true e false

d) true, false, false e false

e) false, true, true e false

10 Quais são os resultados das operações abaixo?

```
10 > 5 ? 10 : 5
!(10 > 5) ? 10 : 5
(10 < 5) ? "k01" : "k02"
```

a) 10, 10, k01

b) 5, 5, k02

c) 10, 5, k02

d) 10, 5, k01

e) 5, 10, k02

11 O que é exibido com o código Java a seguir?

```
1 int i = 10;
2 System.out.println(i++);
3 System.out.println(++i);
4 System.out.println(i--);
5 System.out.println(--i);
```

a) 10, 12, 12 e 10

b) 11, 12, 11, 10

c) 10, 11, 11, 10

d) 10, 10, 10, 10

e) 11, 11, 11, 10

Minha Pontuação [ ] Pontuação Mínima: [8] Pontuação Máxima: [11]

CONTROLE DE FLUXO

CAPÍTULO 5

Neste capítulo, mostraremos instruções que permitem controlar o fluxo de um programa. Essas instruções aumentam a "inteligência" do código. Basicamente, as linguagens de programação oferecem dois tipos de instruções para controlar o fluxo de execução dos programas: instruções de **decisão** e de **repetição**.

Instruções de Decisão

Considere um parque de diversões como os da **Disney**. Nesses parques, para garantir a segurança, alguns brinquedos possuem restrições de acesso. Em geral, essas restrições estão relacionadas à altura dos visitantes. Em alguns parques, a altura do visitante é obtida por sensores instalados na entrada dos brinquedos e um programa de computador libera ou bloqueia o acesso de acordo com altura obtida. Então, o programa deve decidir se executa um trecho de código de acordo com uma condição. Essa decisão pode ser realizada através das instruções de decisão oferecidas pelas linguagens de programação.

Nos exemplos vistos nos capítulos anteriores, a ordem da execução das linhas de um programa é exatamente a ordem na qual elas foram definidas no código fonte. As instruções de decisão proporcionarão uma forma de decidirmos se queremos executar um bloco de código ou não, ou seja, se desejamos pular um trecho de código ou não. As instruções de decisão são capazes de criar um "desvio" no fluxo de execução de um programa.

Instrução if

A instrução **if** (se), é utilizada quando queremos testar uma condição antes de executarmos um ou mais comandos. A sintaxe da instrução **if** é a seguinte:

```
1 if(condição) {
2   // comando 1
3   // comando 2
4   // comando 3
5 }
6 // comando 4
7 // comando 5
```

Como funciona a instrução **if**? Se a condição na **linha 1** for verdadeira, os comandos das **linhas 2, 3 e 4** serão executadas e depois o fluxo de execução do programa segue normalmente e executa a partir da **linha 6** em diante. Por outro lado, se a condição for falsa, as **linhas 2, 3 e 4** não serão executadas e o fluxo de execução do programa "pula" direto para a **linha 6**.

Pare para pensar...
O que é essa tal condição?

A condição é qualquer expressão válida em Java ou C# que devolva um valor booleano. Por exemplo, a expressão "1 < 2" é uma expressão que devolve o valor **true**. Já a expressão "8%3 == 0" devolve o valor **false**

Simulação - Debug

A altura mínima para o ingresso na atração "The Barnstormer" do parque temático da Disney "Magic Kingdom" é **0.89 metros**. Vamos simular a execução do programa que controla o acesso dos visitantes a essa atração.

1 Na **linha 1**, um número aleatório do tipo **double** é gerado com o trecho de código **Math.random()**. Vamos utilizar esse número para representar a altura de um visitante que deseja ingressar na atração "The Barnstormer". Esse valor é armazenado na variável **altura**. Suponha que o valor **0.75** foi gerado.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

2 Na **linha 2**, o valor armazenado na variável **altura** é exibido no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
```

3 Na **linha 3**, a comparação da condição do **if** devolve **true** pois o valor da variável **altura** é menor do que **0.89**.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
```

4 A **linha 4** é executada porque a condição do **if** da **linha 3** é verdadeira. Dessa forma, a mensagem "Acesso bloqueado" é exibida no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
Acesso bloqueado
```

5 Por fim, a **linha 6** é executada e a mensagem "The Barnstormer" é exibida no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
Acesso bloqueado
The Barnstormer
```

Simulação - Debug

1 Na **linha 1**, um número aleatório do tipo **double** é gerado com o trecho de código **Math.random()**. Vamos utilizar esse número para representar a altura de um visitante que deseja ingressar na atração "The Barnstormer". Esse valor é armazenado na variável **altura**. Suponha que o valor **0.97** foi gerado.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

2 Na **linha 2**, o valor armazenado na variável **altura** é exibido no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
```

3 Na **linha 3**, a comparação da condição do **if** devolve **false** pois o valor da variável **altura** não é menor do que **0.89**.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
```

4 A **linha 4** não é executada porque a condição do **if** da **linha 3** é falsa. Dessa forma, o fluxo de execução vai direto para a **linha 6** e a mensagem "The Barnstormer" é exibida no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 }
6 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
The Barnstormer
```

Instrução else

Muitas vezes, queremos executar um bloco de comandos caso uma condição seja verdadeira e outro bloco de comandos caso essa condição seja falsa. Para isso, podemos utilizar as instruções **if** e **else**. Veja abaixo, a estrutura dessas instruções.

```
1  if(condição) {
2    // comando 1
3    // comando 2
4    // comando 3
5  } else {
6    // comando 4
7    // comando 5
8    // comando 6
9  }
10 // comando 7
```

No exemplo acima, se a condição na **linha 1** for verdadeira, as **linhas 2, 3 e 4** serão executadas e depois o fluxo de execução do programa segue para a **linha 10**. Por outro lado, se a condição na **linha 1** for falsa, as **linhas 6, 7 e 8** serão executadas e depois o fluxo de execução do programa segue para a **linha 10**.

A instrução **else** não pode aparecer sozinha no código sem estar vinculada a uma instrução **if**. A instrução **else** pode ser traduzida em português para "senão". Em português, assim como em Java e C#, não podemos dizer "senão" sem antes ter dito "se". Por isso, não podemos utilizar a instrução **else** sem antes ter utilizado a instrução **if**.

Simulação - Debug

A altura mínima para o ingresso na atração "The Barnstormer" do parque temático da Disney "Magic Kingdom" é **0.89 metros**. Vamos simular a execução do programa que controla o acesso dos visitantes a essa atração.

1 Na **linha 1**, um número aleatório do tipo **double** é gerado com o trecho de código **Math.random()**. Vamos utilizar esse número para representar a altura de um visitante que deseja ingressar na atração "The Barnstormer". Esse valor é armazenado na variável **altura**. Suponha que o valor **0.75** foi gerado.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

2 Na **linha 2**, o valor armazenado na variável **altura** é exibido no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
```

3 Na **linha 3**, a comparação da condição do **if** devolve **true** pois o valor da variável **altura** é menor do que **0.89**.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
```

4 A **linha 4** é executada porque a condição do **if** da **linha 3** é verdadeira. Dessa forma, a mensagem "Acesso bloqueado" é exibida no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
Acesso bloqueado
```

5 Por fim, o fluxo de execução "pula" para a **linha 8** e a mensagem "The Barnstormer" é exibida no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.75

```
0.75
Acesso bloqueado
The Barnstormer
```

Simulação - Debug

1 Na **linha 1**, um número aleatório do tipo **double** é gerado com o trecho de código **Math.random()**. Vamos utilizar esse número para representar a altura de um visitante que deseja ingressar na atração "The Barnstormer". Esse valor é armazenado na variável **altura**. Suponha que o valor **0.97** foi gerado.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

2 Na **linha 2**, o valor armazenado na variável **altura** é exibido no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
```

3 Na **linha 3**, a comparação da condição do **if** devolve **false** pois o valor da variável **altura** não é menor do que **0.89**.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
```

4 A **linha 4** não é executada porque a condição do **if** da **linha 3** é falsa. Dessa forma, o fluxo de execução vai direto para a **linha 6** e a mensagem "Acesso liberado" é exibida no terminal.

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
Acesso liberado
```

5 Por fim, o fluxo de execução continua e a **linha 10** é executada exibindo a mensagem "The Barnstormer".

```
1 double altura = Math.random();
2 System.out.println(altura);
3 if(altura < 0.89) {
4   System.out.println("Acesso bloqueado");
5 } else {
6   System.out.println("Acesso liberado");
7 }
8 System.out.println("The Barnstormer");
```

Variáveis

altura = 0.97

```
0.97
Acesso liberado
The Barnstormer
```

Instruções de Decisão Encadeadas

Considere um programa de computador que controla os saques efetuados nos caixas eletrônicos de um banco. Nesse banco, os saques efetuados das 6 horas até as 22 horas não podem ser superiores a R$ 5.000,00. Por outro lado, os saques efetuados depois das 22 horas e antes das 6 horas não podem ser superiores a R$ 400,00. Podemos implementar essa lógica utilizando as instruções de decisão oferecidas pelas linguagens de programação.

```
1 if(hora >= 6 && hora <= 22) {
2   if(valor <= 5000) {
3     System.out.println("Saque efetuado com sucesso");
4   } else {
5     System.out.println("Valor máximo de saque é R$ 5000,00");
6   }
```

```
7  } else {
8    if(valor <= 400) {
9      System.out.println("Saque efetuado com sucesso");
10   } else {
11     System.out.println("Valor máximo de saque é R$ 400,00");
12   }
13 }
```

Exercícios de Fixação Com Java

1 Abra um terminal; Entre na pasta dos seus exercícios e crie uma pasta chamada **controle-de-fluxo** para os arquivos desenvolvidos nesse capítulo.

```
K19/rafael$ mkdir controle-de-fluxo

K19/rafael$ cd controle-de-fluxo

K19/rafael/controle-de-fluxo$
```

Terminal 5.16: Criando a pasta `controle-de-fluxo` *no Linux*

```
C:\Users\K19\rafael> md controle-de-fluxo

C:\Users\K19\rafael> cd controle-de-fluxo

C:\Users\K19\rafael\controle-de-fluxo>
```

Terminal 5.17: Criando a pasta `controle-de-fluxo` *no Windows*

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao1.zip

2 Na pasta **controle-de-fluxo**, crie um arquivo chamado **AprovadoReprovado.java**.

```
1  class AprovadoReprovado {
2    public static void main(String[] args) {
3      double nota = Math.random() * 10;
4
5      System.out.println("A nota do aluno é: " + nota);
6
7      if (nota < 6) {
8        System.out.println("REPROVADO");
9      } else {
10       System.out.println("APROVADO");
11     }
12   }
13 }
```

Código Java 5.23: AprovadoReprovado.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao2.zip

3 Compile e execute algumas vezes a classe **AprovadoReprovado**.

```
K19/rafael/controle-de-fluxo$ javac AprovadoReprovado.java

K19/rafael/controle-de-fluxo$ java AprovadoReprovado
A nota do aluno é: 0.2638522892234163
REPROVADO
```

```
K19/rafael/controle-de-fluxo$ java AprovadoReprovado
A nota do aluno é: 6.979547724462908
APROVADO

K19/rafael/controle-de-fluxo$ java AprovadoReprovado
A nota do aluno é: 9.817359518391823
APROVADO

K19/rafael/controle-de-fluxo$ java AprovadoReprovado
A nota do aluno é: 1.19357817403141
REPROVADO

K19/rafael/controle-de-fluxo$ java AprovadoReprovado
A nota do aluno é: 9.182158940064294
APROVADO
```

Terminal 5.18: Compilando e executando a classe AprovadoReprovado

4 Na pasta **controle-de-fluxo**, crie um arquivo chamado **VerificaValorProduto.java**.

```
1  class VerificaValorProduto {
2    public static void main(String[] args) {
3      double precoDoProduto1 = Math.random() * 1000;
4      double precoDoProduto2 = Math.random() * 1000;
5
6      System.out.println("O preço do produto 1 é: " + precoDoProduto1);
7      System.out.println("O preço do produto 2 é: " + precoDoProduto2);
8
9      if (precoDoProduto1 < precoDoProduto2) {
10       System.out.println("O produto 1 é o mais barato");
11     } else {
12       if(precoDoProduto2 < precoDoProduto1) {
13         System.out.println("O produto 2 é o mais barato");
14       } else {
15         System.out.println("Os preços dos dois produtos são iguais");
16       }
17     }
18   }
19 }
```

Código Java 5.24: VerificaValorProduto.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao4.zip

5 Compile e execute algumas vezes a classe **VerificaValorProduto**.

```
K19/rafael/controle-de-fluxo$ javac VerificaValorProduto.java

K19/rafael/controle-de-fluxo$ java VerificaValorProduto
O preço do produto 1 é: 407.0629216803149
O preço do produto 2 é: 470.09065026412986
O produto 1 é o mais barato

K19/rafael/controle-de-fluxo$ java VerificaValorProduto
O preço do produto 1 é: 822.5829453982788
O preço do produto 2 é: 462.3926076619123
O produto 2 é o mais barato

K19/rafael/controle-de-fluxo$ java VerificaValorProduto
O preço do produto 1 é: 939.8075230341649
O preço do produto 2 é: 883.5996030373839
O produto 2 é o mais barato

K19/rafael/controle-de-fluxo$ java VerificaValorProduto
O preço do produto 1 é: 992.427819296529
O preço do produto 2 é: 992.427819296529
Os preços dos dois produtos são iguais

K19/rafael/controle-de-fluxo$ java VerificaValorProduto
```

```
O preço do produto 1 é: 450.09190082181505
O preço do produto 2 é: 950.05644529133
O produto 1 é o mais barato
```

Terminal 5.19: Compilando e executando a classe `VerificaValorProduto`

6 Na pasta **controle-de-fluxo**, crie um arquivo chamado **EscolheCaminho.java**.

```
1  class EscolheCaminho {
2    public static void main(String[] args) {
3      double valor = Math.random();
4
5      System.out.println("VALOR: " + valor);
6
7      if (valor < 0.5) {
8        System.out.println("Vire à esquerda");
9
10       valor = Math.random();
11
12       System.out.println("VALOR: " + valor);
13
14       if (valor < 0.5) {
15         System.out.println("Vire à esquerda");
16       } else {
17         System.out.println("Vire à direita");
18       }
19
20     } else {
21       System.out.println("Vire à direita");
22
23       valor = Math.random();
24
25       System.out.println("VALOR: " + valor);
26
27       if (valor < 0.5) {
28         System.out.println("Vire à esquerda");
29       } else {
30         System.out.println("Vire à direita");
31       }
32     }
33   }
34 }
```

Código Java 5.25: EscolheCaminho.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao6.zip

7 Compile e execute algumas vezes a classe **EscolheCaminho**.

```
K19/rafael/controle-de-fluxo$ javac EscolheCaminho.java

K19/rafael/controle-de-fluxo$ java EscolheCaminho
VALOR: 0.3017715862415762
Vire à esquerda
VALOR: 0.9011271709760207
Vire à direita

K19/rafael/controle-de-fluxo$ java EscolheCaminho
VALOR: 0.9884547100858677
Vire à direita
VALOR: 0.4083159531305627
Vire à esquerda

K19/rafael/controle-de-fluxo$ java EscolheCaminho
VALOR: 0.6411095634177562
Vire à direita
VALOR: 0.9297619245394584
```

```
Vire à direita
```

Terminal 5.20: Compilando e executando a classe `EscolheCaminho`

8 Na pasta **controle-de-fluxo**, crie um arquivo chamado **EscolheRoupa.java**.

```
class EscolheRoupa {
  public static void main(String[] args) {
    double valor = Math.random();

    if (valor < 0.5) {
      System.out.println("camiseta preta");
    } else {
      System.out.println("camiseta vermelha");
    }

    valor = Math.random();

    if (valor < 0.5) {
      System.out.println("calça jeans");
    } else {
      System.out.println("bermuda");
    }

    valor = Math.random();

    if (valor < 0.5) {
      System.out.println("tênis");
    } else {
      System.out.println("sapato");
    }

    valor = Math.random();

    if (valor < 0.5) {
      System.out.println("boné");
    } else {
      System.out.println("óculos");
    }
  }
}
```

Código Java 5.26: EscolheRoupa.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao8.zip

9 Compile e execute algumas vezes a classe **EscolheRoupa**.

```
K19/rafael/controle-de-fluxo$ javac EscolheRoupa.java

K19/rafael/controle-de-fluxo$ java EscolheRoupa
camiseta vermelha
bermuda
sapato
óculos

K19/rafael/controle-de-fluxo$ java EscolheRoupa
camiseta preta
calça jeans
tênis
óculos

K19/rafael/controle-de-fluxo$ java EscolheRoupa
camiseta preta
bermuda
tênis
boné
```

Terminal 5.21: Compilando e executando a classe `EscolheRoupa`

10 Na pasta **controle-de-fluxo**, crie um arquivo chamado **ADivisivelPorB.java**. Implemente um programa em Java que guarde dois valores numéricos: **a** e **b**. Imprima na tela a mensagem "É divisível" quando **a** for divisível por **b** ou a mensagem "Não é divisível", caso contrário.

```
1  class ADivisivelPorB {
2    public static void main(String[] args) {
3      int a = (int)(Math.random() * 1000);
4      int b = (int)(Math.random() * 20);
5
6      System.out.println("a: " + a);
7      System.out.println("b: " + b);
8
9      if (a % b == 0) {
10       System.out.println("É divisível");
11     } else {
12       System.out.println("Não é divisível");
13     }
14   }
15 }
```

Código Java 5.27: ADivisivelPorB.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao10.zip

11 Compile e execute algumas vezes a classe **ADivisivelPorB**.

```
K19/rafael/controle-de-fluxo$ javac ADivisivelPorB.java

K19/rafael/controle-de-fluxo$ java ADivisivelPorB
a: 779
b: 16
Não é divisível

K19/rafael/controle-de-fluxo$ java ADivisivelPorB
a: 784
b: 16
É divisível

K19/rafael/controle-de-fluxo$ java ADivisivelPorB
a: 20
b: 10
É divisível

K19/rafael/controle-de-fluxo$ java ADivisivelPorB
a: 628
b: 9
Não é divisível

K19/rafael/controle-de-fluxo$ java ADivisivelPorB
a: 615
b: 11
Não é divisível
```

Terminal 5.22: Compilando e executando a classe `ADivisivelPorB`

12 Na pasta **controle-de-fluxo**, crie um arquivo chamado **Saudacao.java**. Implemente um programa em Java que contenha uma variável chamada **hora**. Essa variável deve armazenar a hora do dia. Esse programa deve imprimir a mensagem "Bom dia" se a hora estiver no intervalo [0, 11], "Boa tarde" se a hora estiver no intervalo [12, 17] ou "Boa noite" se a hora estiver no intervalo [18, 23].

```
1  class Saudacao {
2    public static void main(String[] args) {
3      double hora = Math.random() * 24;
4
5      if (hora >= 0 && hora < 12) {
6        System.out.println("Bom dia");
7      } else if (hora >= 12 && hora < 18) {
8        System.out.println("Boa tarde");
9      } else if (hora >= 18 && hora < 24) {
10       System.out.println("Boa noite");
11     }
12   }
13 }
```

Código Java 5.28: Saudacao.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao12.zip

13 Compile e execute a classe **Saudacao**.

```
K19/rafael/controle-de-fluxo$ javac Saudacao.java

K19/rafael/controle-de-fluxo$ java Saudacao
Boa noite

K19/rafael/controle-de-fluxo$ java Saudacao
Boa tarde

K19/rafael/controle-de-fluxo$ java Saudacao
Boa noite

K19/rafael/controle-de-fluxo$ java Saudacao
Boa dia

K19/rafael/controle-de-fluxo$ java Saudacao
Boa tarde
```

Terminal 5.23: Compilando e executando a classe Saudacao

Instruções de Repetição

Considere um programa que gera bilhetes de loteria. O número do primeiro bilhete é 1000, do segundo 1001, do terceiro 1002 e assim por diante até o último bilhete numerado com 9999. Para esse tipo de tarefa, podemos utilizar as instruções de repetição oferecidas pelas linguagens de programação.

Basicamente, as instruções de decisão permitem que um determinado trecho de código seja executado ou não. Em algumas situações, é necessário repetir a execução de um determinado trecho de código. Nessas situações, devemos utilizar as instruções de repetição.

Instrução while

A instrução **while** indica o início de um laço e recebe como parâmetro uma condição. Essa condição é chamada de **condição de parada**, pois quando ela for falsa, o laço é interrompido. A estrutura ou sintaxe da instrução **while** é a seguinte:

```
1 while(condição de parada){
```

```
2   // comando 1
3   // comando 2
4   // comando 3
5 }
```

Se traduzirmos para o português a instrução **while** como **enquanto**, fica mais fácil entender o seu funcionamento. O código acima poderia ser lido da seguinte forma:

"Enquanto a condição de parada for verdadeira, execute comando 1, comando 2 e comando 3."

Considere um programa que exibe na tela cem mensagens de acordo com o seguinte padrão:

```
Mensagem número 1
Mensagem número 2
Mensagem número 3
...
```

Terminal 5.24: Programa que exibe mensagens

Esse programa poderia ser implementado em Java ou C# de uma forma não prática. Veja os exemplos abaixo.

```
1 System.out.println("Mensagem número 1");
2 System.out.println("Mensagem número 2");
3 System.out.println("Mensagem número 3");
4 ...
5 System.out.println("Mensagem número 100");
```

Código Java 5.30: Imprimindo a frase "Mensagem número x".

```
1 System.Console.WriteLine("Mensagem número 1");
2 System.Console.WriteLine("Mensagem número 2");
3 System.Console.WriteLine("Mensagem número 3");
4 ...
5 System.Console.WriteLine("Mensagem número 100");
```

Código C# 5.1: Imprimindo a frase "Mensagem número x".

Contudo, utilizando a instrução **while** o código fica bem mais simples. Observe.

```
1 // Contador de vezes que a mensagem foi impressa.
2 int i = 1;
3
4 while(i <= 100){
5   System.out.println("Mensagem número " + i);
6   i++;
7 }
```

Código Java 5.31: Imprimindo a frase "Mensagem número x".

```
1 // Contador de vezes que a mensagem foi impressa.
2 int i = 1;
3
4 while(i <= 100){
5   System.Console.WriteLine("Mensagem número " + i);
6   i++;
7 }
```

Código C# 5.2: Imprimindo a frase "Mensagem número x".

Até agora, o uso da instrução **while** parece ser mais uma conveniência do que uma necessidade. Vamos mudar um pouco o exemplo anterior para verificar a importância das instruções de repetição. Considere que a frase "Mensagem número x" tenha que ser impressa um número aleatório de vezes. Dessa forma, durante a codificação, não sabemos quantas vezes a frase deverá ser impressa.

Um possível código para solucionar esse novo problema seria:

```
class ExemploWhile {
  public static void main(String[] args) {
    int i = 0

    // número no intervalo [0,99]
    int numeroAleatorio = (int)(Math.random() * 100);

    while(i < numeroAleatorio) {
      System.out.println("Mensagem número " + (i + 1));
      i++;
    }
  }
}
```

Código Java 5.32: Imprimindo a frase "Mensagem número x" um número aleatório de vezes.

```
class ExemploWhile
{
  static void Main()
  {
    int i = 0

    System.Random gerador = new System.Random();

    // número no intervalo [0,99]
    int numeroAleatorio = (int)(gerador.NextDouble() * 100);

    while(i < numeroAleatorio) {
      System.out.println("Mensagem número " + (i + 1));
      i++;
    }
  }
}
```

Código C# 5.3: Imprimindo a frase "Mensagem número x" um número aleatório de vezes.

A cada vez que é executado, o programa acima pode imprimir uma quantidade diferente de mensagens. Esse comportamento seria possível sem a utilização de uma instrução de repetição?

Simulação - Debug

Vamos simular a execução de um programa que gera bilhetes de loteria. Para não alongar muito a simulação, apenas 3 bilhetes serão gerados. Esses bilhetes devem ser numerados sequencialmente iniciando com o número 1000.

1 Na **linha 1**, a variável **numero** é declarada e inicializada com o valor **1000**.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1000

2 Na **linha 2**, a condição de parada do **while** é testada. Como o valor da variável **numero** é menor do que 1002, a condição "numero <= 1002" devolve **true**.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1000

3 Como a condição da **linha 2** devolveu **true**, o corpo do **while** será executado. Ao executar a **linha 3** a mensagem "Bilhete 1000" é exibida no terminal.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1000

```
Bilhete 1000
```

4 Ao executar a **linha 4** a variável **numero** é incrementada para 1001.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1001

```
Bilhete 1000
```

5 O fluxo de execução volta para a **linha 2** e a condição do **while** é testada novamente. Mais uma vez, o valor da variável **numero** é menor do que 1002. Dessa forma, a condição "numero <= 1002" devolve **true**.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1001

```
Bilhete 1000
```

6 Como a condição da **linha 2** devolveu **true**, o corpo do **while** será executado. Ao executar a **linha 3** a mensagem "Bilhete 1001" é exibida no terminal.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1001

```
Bilhete 1000
Bilhete 1001
```

7 Ao executar a **linha 4** a variável **numero** é incrementada para 1002.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1002

```
Bilhete 1000
Bilhete 1001
```

8 Agora, o fluxo de execução volta para a **linha 2** e a condição do **while** é testada novamente. O valor da variável **numero** é igual a 1002. Dessa forma, a condição "numero <= 1002" ainda devolve **true**.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1002

```
Bilhete 1000
Bilhete 1001
```

9 Como a condição da **linha 2** devolveu **true**, o corpo do **while** será executado. Ao executar a **linha 3** a mensagem "Bilhete 1002" é exibida no terminal.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1002

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
```

10 Ao executar a **linha 4** a variável **numero** é incrementada para 1003.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1003

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
```

11 Mais uma vez, o fluxo de execução volta para a **linha 2** para testar a condição do **while**. Finalmente, o valor da variável **numero** não é menor ou igual a 1002. Dessa forma, a condição devolve **false**.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1003

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
```

12 Como a condição da **linha 2** é falsa, o corpo do **while** não será mais executado. Portanto, o laço é interrompido e o fluxo de execução "pula" para a **linha 6**. Ao executar essa linha, a mensagem "FIM" é exibida no terminal.

```
1 int numero = 1000;
2 while(numero <= 1002) {
3   System.out.println("Bilhete " + numero);
4   numero++;
5 }
6 System.out.println("FIM");
```

Variáveis

numero = 1003

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
FIM
```

Instrução for

A instrução **for** é uma outra instrução de repetição e tem a mesma finalidade da instrução **while**. Na maioria dos casos, podemos resolver questões que envolvem repetições com **while** ou **for**. A diferença é que, geralmente, utilizamos a instrução **for** nos casos em que precisamos de um contador em nossa condição de parada. Para ficar mais claro, veja a estrutura ou sintaxe da instrução **for**:

```
1 for(inicialização; condição de parada; atualização){
2   // comandos
3 }
```

No lugar da **inicialização**, devemos inserir os comandos que serão executados antes do início do laço. No lugar da **atualização**, devemos inserir os comandos que serão executadas ao final de cada

iteração(repetição).

Importante

O termo iteração é utilizado quando nos referimos à repetição de uma ou mais ações. Portanto, quando dizemos que "algo deve ser executado a cada iteração de um laço" estamos querendo dizer que "a cada rodada desse laço algo deve ser executado".

Considere o seguinte trecho de código que utiliza a instrução de repetição **while**.

```
1  // inicialização
2  int i = 1;
3
4  // condição
5  while(i <= 100) {
6    //corpo
7    System.out.println("Mensagem número " + i);
8
9    // atualização
10   i++;
11 }
```

Podemos reescrever esse código com a instrução de repetição **for**.

```
1  // inicialização; condição; atualização
2  for(int i = 1; i <= 100; i++) {
3    //corpo
4    System.out.println("Mensagem número " + i);
5  }
```

Perceba que o código ficou mais compacto sem prejudicar a compreensão. Na linha em destaque, declaramos e inicializamos a variável **i** (**int i = 1**); definimos a condição de parada (**i <= 100**) e definimos que ao final de cada iteração devemos atualizar a variável **i** (**i++**). Diferentemente do **while**, no **for**, a inicialização, a condição e a atualização do laço são definidas na mesma linha.

Mais Sobre

Vimos que a instrução **for** possui 3 argumentos: inicialização, condição e atualização. Esses argumentos podem ser mais complexos do que os utilizados anteriormente. Podemos declarar e/ou inicializar diversas variáveis na inicialização. Podemos definir condições mais sofisticadas com uso dos operadores lógicos. Podemos atualizar o valor de diversas variáveis na atualização. Veja um exemplo.

```
1  for(int i = 1, j = 2; i % 2 != 0 || j % 2 == 0; i += j, j += i) {
2    // comandos
3  }
```

Mais Sobre

Os três argumentos da instrução **for** (inicialização, condição e atualização) são opcionais. Consequentemente, o seguinte código é válido apesar de ser estranho no primeiro momento.

```
1 for(;;) {
2   // comandos
3 }
```

O segundo argumento do **for**, a condição, possui o valor padrão **true**.

Pare para pensar...
Sabendo que o segundo argumento do **for**, a condição, possui o valor padrão **true**. Como podemos parar o laço do exemplo a seguir?

```
1 for(;;) {
2   // comandos
3 }
```

Simulação - Debug

Novamente, vamos simular a execução de um programa que gera bilhetes de loteria. Só que agora utilizando a instrução de repetição **for**. Para não alongar muito a simulação, apenas 3 bilhetes serão gerados. Esses bilhetes devem ser numerados sequencialmente iniciando com o número 1000.

1 Na **linha 1**, a variável **numero** é declarada e inicializada com o valor **1000**.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1000

2 Na **linha 1**, a condição de parada do **for** é testada. Como o valor da variável **numero** é menor ou igual a 1002, a condição devolve **true**.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1000

3 Como a condição da **linha 1** devolveu **true**, o corpo do **for** será executado e a mensagem "Bilhete 1000" é exibida no terminal.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1000

```
Bilhete 1000
```

4 Agora, o fluxo de execução volta para a **linha 1** e a atualização do **for** é executada. Dessa forma, a variável **numero** é incrementada para 1001.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1001

```
Bilhete 1000
```

5 Depois da atualização, a condição do **for** é testada novamente. Mais uma vez, o valor da variável **numero** é menor ou igual a 1002. Dessa forma, a condição devolve **true**.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1001

```
Bilhete 1000
```

6 Como a condição da **linha 1** devolveu **true**, o corpo do **for** será executado. Ao executar a **linha 3** a mensagem "Bilhete 1001" é exibida no terminal.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1001

```
Bilhete 1000
Bilhete 1001
```

7 Mais uma vez, o fluxo de execução volta para a atualização do **for** da **linha 1**. Dessa forma, a variável **numero** é incrementada para 1002.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1002

```
Bilhete 1000
Bilhete 1001
```

8 Agora, a condição do **for** é testada novamente. O valor da variável **numero** ainda é menor ou igual a 1002. Dessa forma, a condição devolve **true**.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1002

```
Bilhete 1000
Bilhete 1001
```

9 Como a condição da **linha 1** devolveu **true**, o corpo do **for** será executado. Ao executar a **linha 3**

a mensagem “Bilhete 1002” é exibida no terminal.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1002

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
```

10 Mais uma vez, o fluxo de execução retorna para executar a atualização do **for** da **linha 1**. Assim, a variável **numero** é incrementada para 1003.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1003

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
```

11 Agora, a condição do **for** é testada novamente. Finalmente, o valor da variável **numero** não é menor ou igual a 1002. Dessa forma, a condição devolve **false**.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1003

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
```

12 Como a condição do **for** da **linha 1** é falsa, o laço é interrompido e o fluxo de execução “pula” para a **linha 4**. Ao executar essa linha, a mensagem “FIM” é exibida no terminal.

```
1 for(int numero = 1000; numero <= 1002; numero++) {
2   System.out.println("Bilhete " + numero);
3 }
4 System.out.println("FIM");
```

Variáveis

numero = 1003

```
Bilhete 1000
Bilhete 1001
Bilhete 1002
FIM
```

Instruções de Repetição Encadeadas

Considere o programa de computador que gera os ingressos das apresentações de um determinado teatro. Esse teatro foi dividido em 4 setores com 200 cadeiras cada. Os ingressos devem conter o número do setor e o número da cadeira. Podemos utilizar laços encadeados para implementar esse programa.

```
1 for(int i = 1; i <= 4; i++) {
2   for(int j = 1; j <= 200; j++) {
```

```
3       System.out.println("SETOR: " + i + " CADEIRA: " + j);
4     }
5   }
```

No exemplo acima, para cada iteração do laço externo, há 200 iterações do laço interno. Portanto, o corpo do laço interno executa 800 vezes. Esse valor é exatamente a quantidade de ingressos.

Além de encadear **for**s, podemos encadear **while**s. Veja algumas variações do exemplo anterior.

```
1  int i = 1;
2  while(i <= 4) {
3    int j = 1;
4    while(j <= 200) {
5      System.out.println("SETOR: " + i + " CADEIRA: " + j);
6      j++;
7    }
8    i++;
9  }
```

```
1  int i = 1;
2  while(i <= 4) {
3    int j = 1;
4    for(int j = 1; j <= 200; j++) {
5      System.out.println("SETOR: " + i + " CADEIRA: " + j);
6    }
7    i++;
8  }
```

```
1  for(int i = 1; i <= 4; i++) {
2    int j = 1;
3    while(j <= 200) {
4      System.out.println("SETOR: " + i + " CADEIRA: " + j);
5      j++;
6    }
7  }
```

Exercícios de Fixação Com Java

14 Na pasta **controle-de-fluxo**, crie um arquivo chamado **LoremIpsum.java**. Implemente um programa que exiba no terminal a mensagem "Lorem ipsum dolor sit amet" cinco vezes.

```
1  class LoremIpsum {
2    public static void main(String[] args) {
3      for (int i = 0; i < 5; i++) {
4        System.out.println("Lorem ipsum dolor sit amet");
5      }
6    }
7  }
```

Código Java 5.67: LoremIpsum.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao14.zip

15 Compile e execute a classe **LoremIpsum**.

```
K19/rafael/controle-de-fluxo$ javac LoremIpsum.java

K19/rafael/controle-de-fluxo$ java LoremIpsum
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
```

Terminal 5.45: Compilando e executando a classe `LoremIpsum`

16 Na pasta **controle-de-fluxo**, crie um arquivo chamado **Imprime100.java**. Implemente um programa que exiba no terminal os números de 1 até 100.

```
1 class Imprime100 {
2   public static void main(String[] args) {
3     for (int i = 1; i <= 100; i++) {
4       System.out.println(i);
5     }
6   }
7 }
```

Código Java 5.68: Imprime100.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao16.zip

17 Compile e execute a classe **Imprime100**.

```
K19/rafael/controle-de-fluxo$ javac Imprime100.java

K19/rafael/controle-de-fluxo$ java Imprime100
1
2
3
...
100
```

Terminal 5.46: Compilando e executando a classe `Imprime100`

18 Na pasta **controle-de-fluxo**, crie um arquivo chamado **Imprime100ExcetoMultiplo3.java**. Implemente um programa que exiba no terminal os números de 1 até 100 exceto os números múltiplos de 3.

```
1 class Imprime100ExcetoMultiplo3 {
2   public static void main(String[] args) {
3     for (int i = 1; i <= 100; i++) {
4       if (i % 3 != 0) {
5         System.out.println(i);
6       }
7     }
8   }
9 }
```

Código Java 5.69: Imprime100ExcetoMultiplo3.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao18.zip

19 Compile e execute a classe **Imprime100ExcetoMultiplo3**.

```
K19/rafael/controle-de-fluxo$ javac Imprime100ExcetoMultiplo3.java

K19/rafael/controle-de-fluxo$ java Imprime100ExcetoMultiplo3
1
2
4
5
...
100
```

Terminal 5.47: Compilando e executando a classe `Imprime100ExcetoMultiplo3`

20 Na pasta **controle-de-fluxo**, crie um arquivo chamado **DivideMaiorInteiro.java**. Implemente um programa que declare e inicialize uma variável que receberá o maior número possível do tipo **int**. Divida o valor dessa variável por 2 até que o resultado obtido seja inferior a 100 (não inclusivo). A cada iteração imprima o resultado.

```
1  class DivideMaiorInteiro {
2    public static void main(String[] args) {
3      int numero = 2147483647;
4
5      while (numero >= 100) {
6        numero /= 2;
7        System.out.println(numero);
8      }
9    }
10 }
```

Código Java 5.70: DivideMaiorInteiro.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao20.zip

21 Compile e execute a classe **DivideMaiorInteiro**.

```
K19/rafael/controle-de-fluxo$ javac DivideMaiorInteiro.java

K19/rafael/controle-de-fluxo$ java DivideMaiorInteiro
1073741823
536870911
268435455
...
```

Terminal 5.48: Compilando e executando a classe `DivideMaiorInteiro`

22 Na pasta **controle-de-fluxo**, crie um arquivo chamado **GeradorDeIngressos.java**. Implemente um programa para gerar os ingressos das apresentações de um teatro. Considere que esse teatro possui 4 setores e cada setor possui 20 lugares.

```
1  class GeradorDeIngressos {
2    public static void main(String[] args) {
3      for(int i = 1; i <= 4; i++) {
4        for(int j = 1; j <= 20; j++) {
5          System.out.println("Setor: " + i + " Cadeira: " + j);
6        }
7      }
8    }
9  }
```

Código Java 5.71: GeradorDeIngressos.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao22.zip

23 Compile e execute a classe **GeradorDeIngressos**.

```
K19/rafael/controle-de-fluxo$ javac GeradorDeIngressos.java

K19/rafael/controle-de-fluxo$ java GeradorDeIngressos
Setor 1 Cadeira 1
Setor 1 Cadeira 2
Setor 1 Cadeira 3
...
Setor 4 Cadeira 20
```

Terminal 5.49: Compilando e executando a classe `GeradorDeIngressos`

Instrução break

Considere um jogo de dados no qual o jogador ganha quando a soma dos números obtidos em lançamentos consecutivos de um dado ultrapassar um determinado valor. Antes de começar o jogo, é necessário definir a quantidade máxima de lançamentos e o valor que deve ser ultrapassado para obter a vitória. Eventualmente, se o valor desejado for ultrapassado antes do último lançamento, não é necessário continuar jogando o dado pois a vitória já está garantida. Podemos implementar um programa de computador para simular a execução desse jogo. Nesse programa, podemos utilizar a instrução **break** para interromper os lançamentos se o valor desejado for ultrapassado.

A instrução **break** não é uma instrução de repetição, mas está fortemente relacionada às instruções **while** e **for**. Ela é utilizada para forçar a parada de um laço.

No exemplo abaixo, a lógica para simular o jogo descrito anteriormente considera que a quantidade máxima de lançamentos é 100 e o valor desejado é 360.

```
1  int soma = 0;
2
3  for(int i = 1; i <= 100; i++) {
4    System.out.println("Lançamento: " + i);
5
6    int numero = (int)(Math.random() * 6 + 1);
7
8    System.out.println("Número: " + numero);
9
10   soma += numero;
11
12   System.out.println("Soma: " + soma);
13
14   if(soma > 360) {
15     System.out.println("Você ganhou com " + i + " lançamentos");
16     break;
17   }
18 }
19 System.out.println("Jogue Novamente");
```

O trecho "**(int)(Math.random() * 6 + 1)**" gera um número entre 1 e 6. Esse trecho simula o lançamento de um dado. A variável **soma** acumula os números gerados. A instrução **if** é utilizada para verificar se a soma ultrapassou o valor desejado. Dentro do corpo do **if**, utilizamos o comando **break** para interromper as interações do laço **for** pois, quando a soma ultrapassa 360, a vitória já está garantida.

Simulação - Debug

Vamos simular a execução do jogo de dados descrito anteriormente. Para não alongar muito a simulação, considere que o número máximo de lançamentos é 2 e o valor que deve ser ultrapassado é 7.

1 Na **linha 1**, a variável **soma** é declarada e inicializada com o valor **0**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 0

2 Na **linha 2**, a inicialização do **for** é executada. A variável **i** é criada e inicializada com o valor **1**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 0

i = 1

3 Na **linha 2**, a condição de parada do **for** é testada. Como o valor da variável **i** é menor ou igual a 2, a condição devolve **true**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 0

i = 1

4 Como a condição da **linha 2** devolveu **true**, o corpo do **for** será executado. Ao executar a **linha 3**, a mensagem "Lançamento: 1" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 0

i = 1

```
Lançamento: 1
```

5 Na sequência, a **linha 4** é executada. Um número aleatório entre 1 e 6 é gerado e armazenado na variável **numero**. Suponha que o número gerado foi **5**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 0

i = 1

numero = 5

```
Lançamento: 1
```

6 Prosseguindo, a **linha 6** é executada e a mensagem "Número: 5" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis
soma = 0
i = 1
numero = 5

```
Lançamento: 1
Número: 5
```

7 Adiante, a **linha 7** é executada e o valor da variável **numero** é incrementado na variável **soma**. Dessa forma, a variável **soma** passa a armazenar o valor **5**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis
soma = 5
i = 1
numero = 5

```
Lançamento: 1
Número: 5
```

8 Na sequência, a **linha 9** é executada e a mensagem "Soma: 5" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis
soma = 5
i = 1
numero = 5

```
Lançamento: 1
Número: 5
Soma: 5
```

9 Prosseguindo, a **linha 10** é executada e a condição do **if** é testada. Como o valor da variável **soma** não é maior do que 7, a condição devolve **false**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 5

i = 1

numero = 5

```
Lançamento: 1
Número: 5
Soma: 5
```

10 Como a condição da **linha 10** é falsa, o corpo do **if** não será executado e o fluxo de execução vai para a atualização do **for** na **linha 2**. Dessa forma, a variável **i** é incrementada para **2**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 5

i = 2

```
Lançamento: 1
Número: 5
Soma: 5
```

11 Novamente, na **linha 2**, a condição de parada do **for** é testada. Como o valor da variável **i** é menor ou igual a 2, a condição devolve **true**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 5

i = 2

```
Lançamento: 1
Número: 5
Soma: 5
```

12 Como a condição da **linha 2** devolveu **true**, o corpo do **for** será executado. Ao executar a **linha 3**, a mensagem "Lançamento: 2" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 5

i = 2

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
```

13 Na sequência, a **linha 4** é executada. Um número aleatório entre 1 e 6 é gerado e armazenado na variável **numero**. Suponha que o número gerado foi **3**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 5

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
```

14 Prosseguindo, a **linha 6** é executada e a mensagem "Número: 3" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 5

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
```

15 Adiante, a **linha 7** é executada e o valor da variável **numero** é incrementado na variável **soma**. Dessa forma, a variável **soma** passa a armazenar o valor **8**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 8

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
```

16 Na sequência, a **linha 9** é executada e a mensagem “Soma: 8” é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 8

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
Soma: 8
```

17 Prosseguindo, a **linha 10** é executada e a condição do **if** é testada. Como o valor da variável **soma** é maior do que 7, a condição devolve **true**.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 8

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
Soma: 8
```

18 Como a condição da **linha 10** é verdadeira, o corpo do **if** é será executado. Ao executar a **linha 11** a mensagem "Você ganhou com 2 lançamentos" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 8

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
Soma: 8
Você ganhou com 2 lançamentos
```

19 Agora, a **linha 12** é executada. Dessa forma, a instrução **break** interrompe o laço.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 8

i = 2

numero = 3

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
Soma: 8
Você ganhou com 2 lançamentos
```

20 Por fim, a **linha 15** é executada e a mensagem "Jogue Novamente" é exibida no terminal.

```
1  int soma = 0;
2  for(int i = 1; i <= 2; i++) {
3    System.out.println("Lançamento: " + i);
4    int numero = (int)(Math.random() * 6 + 1);
5
6    System.out.println("Número: " + numero);
7    soma += numero;
8
9    System.out.println("Soma: " + soma);
10   if(soma > 7) {
11     System.out.println("Você ganhou com " + i + " lançamentos");
12     break;
13   }
14 }
15 System.out.println("Jogue Novamente");
```

Variáveis

soma = 8

```
Lançamento: 1
Número: 5
Soma: 5
Lançamento: 2
Número: 3
Soma: 8
Você ganhou com 2 lançamentos
Jogue Novamente
```

Instrução continue

Considere uma variação do jogo de dados proposto anterioremente. Nessa nova versão, somente valores pares devem ser somados. Em outras palavras, os valores ímpares devem ser descartados. Nesse caso, podemos utilizar a instrução **continue**. Essa instrução permite que, durante a execução de um laço, uma determinada iteração seja abortada fazendo com que o fluxo de execução continue para a próxima iteração.

O código abaixo simula o jogo de dados discutido anteriormente com a variação proposta.

```
1  int soma = 0;
2
3  for(int i = 1; i <= 100; i++) {
4    System.out.println("Lançamento: " + i);
5
6    int numero = (int)(Math.random() * 6 + 1);
7
8    System.out.println("Número: " + numero);
9
10   if(numero % 2 != 0) {
11     continue;
12   }
13
14   soma += numero;
15
16   System.out.println("Soma: " + soma);
17
18   if(soma > 180) {
19     System.out.println("Você ganhou com " + i + " lançamentos");
20     break;
21   }
22 }
23 System.out.println("Jogue Novamente");
```

No trecho destacado, calculamos o resto da divisão do número gerado aleatoriamente por dois. Além disso, na condição do **if**, verificamos se esse valor é diferente de zero. Se essa condição for verdadeira significa que o número gerado aleatoriamente é ímpar e consequentemente deve ser descartado. No corpo do **if**, utilizamos a instrução **continue** para abortar a iteração atual.

Importante
Quando aplicada a laços **while**, a instrução **continue** "pula" para a condição. Por outro lado, quando aplicada a laços **for**, ela "pula" para a atualização.

Simulação - Debug

Vamos simular a execução de um programa que gera aleatoriamente 2 números inteiros entre 1 e 100 e exibe no terminal apenas os ímpares.

1 Na **linha 1**, a variável **i** é declarada e inicializada com o valor **1**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 1

2 Na sequência, a condição do **for** é testada. Como valor da variável **i** é menor ou igual a **2**, essa condição devolve **true**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 1

3 O corpo do **for** é executado porque a condição da **linha 1** devolveu **true**. Ao executar a **linha 2**, um número aleatório entre 1 e 100 é gerado e armazenado na variável **numero**. Suponha que o valor gerado é **38**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 1

numero = 38

4 Na **linha 3**, verificamos se o resto da divisão do valor da variável **numero** por 2 é igual a 0. Como essa variável está armazenando o valor **38**, a condição do **if** devolve **true** pois o resto da divisão de 38 por 2 é 0.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 1

numero = 38

5 Como a condição da **linha 3** devolveu **true**, o corpo do **if** é executado. Ao executar a **linha 4**, a instrução **continue** "pula" para a próxima iteração.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 1

numero = 38

6 Devido ao desvio causado pela instrução **continue**, o fluxo de execução vai para a atualização do **for** na **linha 1**. Dessa forma, a variável **i** é incrementada para **2**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 2

7 Na sequência, a condição do **for** é testada. Como valor da variável **i** é menor ou igual a **2**, essa condição devolve **true**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 2

8 O corpo do **for** é executado porque a condição da **linha 1** devolveu **true**. Ao executar a **linha 2**, um número aleatório entre 1 e 100 é gerado e armazenado na variável **numero**. Suponha que o valor gerado é **97**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 2

numero = 97

9 Na **linha 3**, verificamos se o resto da divisão do valor da variável **numero** por 2 é igual a 0. Como essa variável está armazenando o valor **97**, a condição do **if** devolve **false** pois o resto da divisão de 97 por 2 não é 0.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 2

numero = 97

10 Como a condição da **linha 3** devolveu **false**, o corpo do **if** não é executado. Dessa forma, o fluxo de execução vai direto para a **linha 6** e o valor **97** é exibido no terminal.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 2

numero = 97

```
97
```

11 Prosseguindo, o fluxo de execução vai para a atualização do **for** na **linha 1**. Dessa forma, a variável **i** é incrementada para **3**.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 3

```
97
```

12 Na sequência, a condição do **for** é testada. Como valor da variável **i** não é menor ou igual a **2**, essa condição devolve **false**. Dessa forma, o laço é finalizado.

```
1 for(int i = 1; i <= 2; i++) {
2   int numero = (int)(Math.random() * 100 + 1);
3   if(numero % 2 == 0) {
4     continue;
5   }
6   System.out.println(i);
7 }
```

Variáveis

i = 2

```
97
```

Exercícios de Fixação Com Java

24 Na pasta **controle-de-fluxo**, crie um arquivo chamado **JogoDeDado.java**. Implemente um programa que simula a execução de um jogo de dados, no qual são permitidos 5 lançamentos e o jogador ganha se a soma dos valores obtidos nos lançamentos ultrapassar 19.

```
1  class JogoDeDado {
2    public static void main(String[] args) {
3      int soma = 0;
4      for(int i = 1; i <= 5; i++) {
5        System.out.println("Lançamento: " + i);
6        int numero = (int)(Math.random() * 6 + 1);
7
8        System.out.println("Número: " + numero);
9        soma += numero;
10
11       System.out.println("Soma: " + soma);
```

```
12        System.out.println("----------------------");
13
14        if(soma > 19) {
15          System.out.println("Você ganhou com " + i + " lançamentos");
16          break;
17        }
18      }
19    }
20  }
```

Código Java 5.106: JogoDeDado.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao24.zip

25 Compile e execute algumas vezes a classe **JodoDeDado**.

```
K19/rafael/controle-de-fluxo$ javac JogoDeDado.java

K19/rafael/controle-de-fluxo$ java JogoDeDado
...
```

Terminal 5.70: Compilando e executando a classe JodoDeDado

26 Na pasta **controle-de-fluxo**, crie um arquivo chamado **JogoDeDado2.java**. Implemente um programa que simula a execução de um jogo de dados, no qual são permitidos 4 lançamentos e o jogador ganha se a soma dos valores ímpares obtidos nos lançamentos ultrapassar 9.

```
1  class JogoDeDado2 {
2    public static void main(String[] args) {
3      int soma = 0;
4      for(int i = 1; i <= 5; i++) {
5        System.out.println("Lançamento: " + i);
6        int numero = (int)(Math.random() * 6 + 1);
7
8        System.out.println("Número: " + numero);
9
10       if(numero % 2 == 0) {
11         System.out.println("----------------------");
12         continue;
13       }
14
15       soma += numero;
16
17       System.out.println("Soma: " + soma);
18       System.out.println("----------------------");
19
20       if(soma > 9) {
21         System.out.println("Você ganhou com " + i + " lançamentos");
22         break;
23       }
24     }
25   }
26 }
```

Código Java 5.107: JogoDeDado2.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao26.zip

27 Compile e execute algumas vezes a classe **JodoDeDado2**.

```
K19/rafael/controle-de-fluxo$ javac JogoDeDado2.java

K19/rafael/controle-de-fluxo$ java JogoDeDado2
...
```

Terminal 5.71: Compilando e executando a classe JodoDeDado2

Blocos Sem Chaves

Normalmente, os blocos de código associados às instruções de decisão **if** e **else** ou às instruções de repetição **while** e **for** são delimitados com chaves "{}". Contudo, nas linguagens Java e C#, as chaves podem ser omitidas nos blocos que possuem apenas um comando. Veja alguns exemplos.

```
1 if(a < 10)
2   a = a * 2 + 1;
3 else
4   a = a / 2 + 1;
```

```
1 while(a < 10)
2   a = a * 2 + 1;
```

```
1 for(int i = 1; i < 10; i++)
2   a = a * 2 + 1;
```

Lembre-se

Nas linguagens Java e C#, apenas blocos de código com apenas um comando podem ser associados às instruções de decisão **if** e **else** ou às instruções de repetição **while** e **for**.

Normalmente, não delimitar com chaves os blocos de código com dois ou mais comandos gera erros de lógica ou até mesmo erros de compilação. Para evitar esses problemas, a utilização das chaves mesmo em blocos com apenas um comando é recomendada.

Considere o seguinte trecho de código.

```
1 if(a < 10)
2   a = a * 2 + 1;
3 else  {
4   if(a < 20)
5     a = a * 3 + 1;
6   else
7     a = a * 4 + 1;
8 }
```

O trecho em destaque, apesar de conter diversas linhas de código, é considerado um comando único. Dessa forma, podemos omitir as chaves que envolvem esse trecho. Reescrevendo o código, teríamos o seguinte resultado:

```
1 if(a < 10)
2   a = a * 2 + 1;
3 else if(a < 20)
4   a = a * 3 + 1;
```

```
5 else
6   a = a * 4 + 1;
```

Os leitores mais desavisados desse código podem assumir a existência da instrução **else if**. Contudo, essa instrução não existe nas linguagens Java e C#. Na verdade, nesse exemplo, o segundo **if** pertence ao corpo do primeiro **else**.

"Laços Infinitos"

Um laço é interrompido quando a condição de parada for falsa ou quando utilizamos a instrução **break**. Dessa forma, considere os seguintes laços.

```
1 int i = 1;
2 while(i < 10) {
3   System.out.println("K19");
4 }
```

```
1 for(int i = 1; i < 10;) {
2   System.out.println("K19");
3 }
```

Observe que a condição de parada nunca devolverá o valor **false**. Dessa forma, os laços acima nunca serão interrompidos. Esses laços são chamados popularmente de "Laços Infinitos".

Exercícios de Fixação Com Java

28 Na pasta **controle-de-fluxo**, crie um arquivo chamado **BartChalkboard.java**. Implemente um programa para ajudar o Bart Simpson a cumprir o seu castigo.

```
1 class BartChalkboard {
2   public static void main(String[] args) {
3     for(;;)
4       System.out.println("I WILL NOT XEROX MY BUTT");
5   }
6 }
```

Código Java 5.115: BartChalkboard.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao28.zip

29 Compile e execute a classe **BartChalkboard**.

```
K19/rafael/controle-de-fluxo$ javac BartChalkboard.java

K19/rafael/controle-de-fluxo$ java BartChalkboard
I WILL NOT XEROX MY BUTT
I WILL NOT XEROX MY BUTT
I WILL NOT XEROX MY BUTT
...
```

Terminal 5.72: Compilando e executando a classe `BartChalkboard`

Para interromper a execução do programa no terminal digite **CTRL+C**.

Erro: Não utilizar condições booleanas

Um erro de compilação comum em Java ou C# ocorre quando não utilizamos condições booleanas nas instruções **if**, **while** ou **for**.

```
1  class Programa {
2    public static void main(String[] args) {
3      double a = Math.random();
4
5      double b = Math.random();
6
7      if(a + b) {
8        a *= 2;
9      } else {
10       a /= 2;
11     }
12   }
13 }
```

Código Java 5.116: Programa.java

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.java:7: error: incompatible types
    if(a + b) {
       ^
  required: boolean
  found:    double
1 error
```

Terminal 5.73: Erro de compilação

Agora, veja um exemplo de programa em C# com esse problema.

```
1  class Programa
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      double a = gerador.NextDouble();
8
9      double b = gerador.NextDouble();
10
11     if(a + b)
12     {
13       a *= 2;
14     }
15     else
16     {
17       a /= 2;
18     }
19   }
20 }
```

Código C# 5.4: Programa.cs

A mensagem de erro de compilação seria semelhante a apresenta abaixo.

```
Programa.cs(11,6): error CS0029: Cannot implicitly convert type ‘double’ to ‘bool’
```

Terminal 5.74: Erro de compilação

Erro: Excesso de “;”

Um erro de compilação comum em Java ou C# ocorre quando o caractere “;” é adicionado em excesso.

```
1 class Programa {
2   public static void main(String[] args) {
3     for(int i = 0; i < 10; i++);{
4       System.out.println("***************");
5     }
6   }
7 }
```

Código Java 5.117: Programa.java

Observe o caractere “;” depois dos argumentos do **for**. Na verdade, não há erros de compilação nesse código. Contudo, podemos considerar que há um erro de lógica pois o laço não tem **corpo**. O bloco depois do **for** executará apenas uma vez pois não está associado ao laço.

Veja o resultado da execução desse programa.

```
**************
```

Terminal 5.75: Erro de lógica

Agora, veja um exemplo de programa em C# com esse problema.

```
1  class Programa
2  {
3    static void Main()
4    {
5      for(int i = 0; i < 10; i++);
6      {
7        System.Console.WriteLine("***************");
8      }
9    }
10 }
```

Código C# 5.5: Programa.cs

Veja o resultado da execução desse programa.

```
**************
```

Terminal 5.76: Erro de lógica

Exercícios de Fixação Com C#

30 Na pasta **controle-de-fluxo**, crie um arquivo chamado **AprovadoReprovado.cs**.

```
class AprovadoReprovado
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    double nota = gerador.NextDouble() * 10;

    System.Console.WriteLine("A nota do aluno é: " + nota);

    if (nota < 6)
    {
      System.Console.WriteLine("REPROVADO");
    }
    else
    {
      System.Console.WriteLine("APROVADO");
    }
  }
}
```

Código C# 5.6: AprovadoReprovado.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao30.zip

31 Compile e execute algumas vezes a classe **AprovadoReprovado**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc AprovadoReprovado.cs

C:\Users\K19\rafael\controle-de-fluxo> AprovadoReprovado.exe
A nota do aluno é: 0.2638522892234163
REPROVADO

C:\Users\K19\rafael\controle-de-fluxo> AprovadoReprovado.exe
A nota do aluno é: 6.979547724462908
APROVADO

C:\Users\K19\rafael\controle-de-fluxo> AprovadoReprovado.exe
A nota do aluno é: 9.817359518391823
APROVADO

C:\Users\K19\rafael\controle-de-fluxo> AprovadoReprovado.exe
A nota do aluno é: 1.19357817403141
REPROVADO

C:\Users\K19\rafael\controle-de-fluxo> AprovadoReprovado.exe
A nota do aluno é: 9.182158940064294
APROVADO
```

Terminal 5.77: Compilando e executando a classe AprovadoReprovado

32 Na pasta **controle-de-fluxo**, crie um arquivo chamado **VerificaValorProduto.cs**.

```
class VerificaValorProduto
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    double precoDoProduto1 = gerador.NextDouble() * 1000;
    double precoDoProduto2 = gerador.NextDouble() * 1000;

    System.Console.WriteLine("O preço do produto 1 é: " + precoDoProduto1);
    System.Console.WriteLine("O preço do produto 2 é: " + precoDoProduto2);

    if (precoDoProduto1 < precoDoProduto2)
    {
```

```
15        System.Console.WriteLine("O produto 1 é o mais barato");
16      }
17      else
18      {
19        if(precoDoProduto2 < precoDoProduto1)
20        {
21          System.Console.WriteLine("O produto 2 é o mais barato");
22        }
23        else
24        {
25          System.Console.WriteLine("Os preços dos dois produtos são iguais");
26        }
27      }
28    }
29  }
```

Código C# 5.7: VerificaValorProduto.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao32.zip

33 Compile e execute algumas vezes a classe **VerificaValorProduto**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc VerificaValorProduto.cs

C:\Users\K19\rafael\controle-de-fluxo> VerificaValorProduto.exe
O preço do produto 1 é: 407.0629216803149
O preço do produto 2 é: 470.09065026412986
O produto 1 é o mais barato

C:\Users\K19\rafael\controle-de-fluxo> VerificaValorProduto.exe
O preço do produto 1 é: 822.5829453982788
O preço do produto 2 é: 462.3926076619123
O produto 2 é o mais barato

C:\Users\K19\rafael\controle-de-fluxo> VerificaValorProduto.exe
O preço do produto 1 é: 939.8075230341649
O preço do produto 2 é: 883.5996030373839
O produto 2 é o mais barato

C:\Users\K19\rafael\controle-de-fluxo> VerificaValorProduto.exe
O preço do produto 1 é: 992.427819296529
O preço do produto 2 é: 992.427819296529
Os preços dos dois produtos são iguais

C:\Users\K19\rafael\controle-de-fluxo> VerificaValorProduto.exe
O preço do produto 1 é: 450.09190082181505
O preço do produto 2 é: 950.05644529133
O produto 1 é o mais barato
```

Terminal 5.78: Compilando e executando a classe `VerificaValorProduto`

34 Na pasta **controle-de-fluxo**, crie um arquivo chamado **EscolheCaminho.cs**.

```
1   class EscolheCaminho
2   {
3     static void Main()
4     {
5       System.Random gerador = new System.Random();
6
7       double valor = gerador.NextDouble();
8
9       System.Console.WriteLine("VALOR: " + valor);
10
11      if (valor < 0.5)
12      {
13        System.Console.WriteLine("Vire à esquerda");
14
15        valor = gerador.NextDouble();
```

```
16
17        System.Console.WriteLine("VALOR: " + valor);
18
19        if (valor < 0.5)
20        {
21          System.Console.WriteLine("Vire à esquerda");
22        }
23        else
24        {
25          System.Console.WriteLine("Vire à direita");
26        }
27
28      }
29      else
30      {
31        System.Console.WriteLine("Vire à direita");
32
33        valor = gerador.NextDouble();
34
35        System.Console.WriteLine("VALOR: " + valor);
36
37        if (valor < 0.5)
38        {
39          System.Console.WriteLine("Vire à esquerda");
40        }
41        else
42        {
43          System.Console.WriteLine("Vire à direita");
44        }
45      }
46    }
47  }
```

Código C# 5.8: EscolheCaminho.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao34.zip

35 Compile e execute algumas vezes a classe **EscolheCaminho**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc EscolheCaminho.cs

C:\Users\K19\rafael\controle-de-fluxo> EscolheCaminho.exe
VALOR: 0.3017715862415762
Vire à esquerda
VALOR: 0.9011271709760207
Vire à direita

C:\Users\K19\rafael\controle-de-fluxo> EscolheCaminho.exe
VALOR: 0.9884547100858677
Vire à direita
VALOR: 0.4083159531305627
Vire à esquerda

C:\Users\K19\rafael\controle-de-fluxo> EscolheCaminho.exe
VALOR: 0.6411095634177562
Vire à direita
VALOR: 0.9297619245394584
Vire à direita
```

Terminal 5.79: Compilando e executando a classe `EscolheCaminho`

36 Na pasta **controle-de-fluxo**, crie um arquivo chamado **EscolheRoupa.cs**.

```
1  class EscolheRoupa
2  {
3    static void Main()
4    {
```

```
5       System.Random gerador = new System.Random();
6
7       double valor = gerador.NextDouble();
8
9       if (valor < 0.5)
10      {
11        System.Console.WriteLine("camiseta preta");
12      }
13      else
14      {
15        System.Console.WriteLine("camiseta vermelha");
16      }
17
18      valor = gerador.NextDouble();
19
20      if (valor < 0.5)
21      {
22        System.Console.WriteLine("calça jeans");
23      }
24      else
25      {
26        System.Console.WriteLine("bermuda");
27      }
28
29      valor = gerador.NextDouble();
30
31      if (valor < 0.5)
32      {
33        System.Console.WriteLine("tênis");
34      }
35      else
36      {
37        System.Console.WriteLine("sapato");
38      }
39
40      valor = gerador.NextDouble();
41
42      if (valor < 0.5)
43      {
44        System.Console.WriteLine("boné");
45      }
46      else
47      {
48        System.Console.WriteLine("óculos");
49      }
50    }
51  }
```

Código C# 5.9: EscolheRoupa.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao36.zip

37 Compile e execute algumas vezes a classe **EscolheRoupa**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc EscolheRoupa.cs

C:\Users\K19\rafael\controle-de-fluxo> EscolheRoupa.exe
camiseta vermelha
bermuda
sapato
óculos

C:\Users\K19\rafael\controle-de-fluxo> EscolheRoupa.exe
camiseta preta
calça jeans
tênis
óculos

C:\Users\K19\rafael\controle-de-fluxo> EscolheRoupa.exe
```

```
camiseta preta
bermuda
tênis
boné
```

Terminal 5.80: Compilando e executando a classe `EscolheRoupa`

38 Na pasta **controle-de-fluxo**, crie um arquivo chamado **ADivisivelPorB.cs**. Implemente um programa em Java que guarde dois valores numéricos: **a** e **b**. Imprima na tela a mensagem "É divisível" quando **a** for divisível por **b** ou a mensagem "Não é divisível", caso contrário.

```
1  class ADivisivelPorB
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      int a = (int)(gerador.NextDouble() * 1000);
8      int b = (int)(gerador.NextDouble() * 20);
9
10     System.Console.WriteLine("a: " + a);
11     System.Console.WriteLine("b: " + b);
12
13     if (a % b == 0)
14     {
15       System.Console.WriteLine("É divisível");
16     }
17     else
18     {
19       System.Console.WriteLine("Não é divisível");
20     }
21   }
22 }
```

Código C# 5.10: ADivisivelPorB.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao38.zip

39 Compile e execute algumas vezes a classe **ADivisivelPorB**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc ADivisivelPorB.cs

C:\Users\K19\rafael\controle-de-fluxo> ADivisivelPorB.exe
a: 779
b: 16
Não é divisível

C:\Users\K19\rafael\controle-de-fluxo> ADivisivelPorB.exe
a: 784
b: 16
É divisível

C:\Users\K19\rafael\controle-de-fluxo> ADivisivelPorB.exe
a: 20
b: 10
É divisível

C:\Users\K19\rafael\controle-de-fluxo> ADivisivelPorB.exe
a: 628
b: 9
Não é divisível

C:\Users\K19\rafael\controle-de-fluxo> ADivisivelPorB.exe
a: 615
b: 11
Não é divisível
```

Terminal 5.81: Compilando e executando a classe `ADivisivelPorB`

40 Na pasta **controle-de-fluxo**, crie um arquivo chamado **Saudacao.cs**. Implemente um programa em Java que contenha uma variável chamada **hora**. Essa variável deve armazenar a hora do dia. Esse programa deve imprimir a mensagem "Bom dia" se a hora estiver no intervalo [0, 11], "Boa tarde" se a hora estiver no intervalo [12, 17] ou "Boa noite" se a hora estiver no intervalo [18, 23].

```
class Saudacao
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    double hora = gerador.NextDouble() * 24;

    if (hora >= 0 && hora < 12)
    {
      System.Console.WriteLine("Bom dia");
    }
    else if (hora >= 12 && hora < 18)
    {
      System.Console.WriteLine("Boa tarde");
    }
    else if (hora >= 18 && hora < 24)
    {
      System.Console.WriteLine("Boa noite");
    }
  }
}
```

Código C# 5.11: Saudacao.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao40.zip

41 Compile e execute a classe **Saudacao**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc Saudacao.cs

C:\Users\K19\rafael\controle-de-fluxo> Saudacao.exe
Boa noite

C:\Users\K19\rafael\controle-de-fluxo> Saudacao.exe
Boa tarde

C:\Users\K19\rafael\controle-de-fluxo> Saudacao.exe
Boa noite

C:\Users\K19\rafael\controle-de-fluxo> Saudacao.exe
Boa dia

C:\Users\K19\rafael\controle-de-fluxo> Saudacao.exe
Boa tarde
```

Terminal 5.82: Compilando e executando a classe Saudacao

42 Na pasta **controle-de-fluxo**, crie um arquivo chamado **LoremIpsum.cs**. Implemente um programa que exiba no terminal a mensagem "Lorem ipsum dolor sit amet" cinco vezes.

```
class LoremIpsum
{
  static void Main()
  {
    for (int i = 0; i < 5; i++)
    {
      System.Console.WriteLine("Lorem ipsum dolor sit amet");
    }
```

```
9     }
10 }
```

Código C# 5.12: LoremIpsum.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao42.zip

43 Compile e execute a classe **LoremIpsum**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc LoremIpsum.cs

C:\Users\K19\rafael\controle-de-fluxo> LoremIpsum.exe
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet
```

Terminal 5.83: Compilando e executando a classe `LoremIpsum`

44 Na pasta **controle-de-fluxo**, crie um arquivo chamado **Imprime100.cs**. Implemente um programa que exiba no terminal os números de 1 até 100.

```
1  class Imprime100
2  {
3    static void Main()
4    {
5      for (int i = 1; i <= 100; i++)
6      {
7        System.Console.WriteLine(i);
8      }
9    }
10 }
```

Código C# 5.13: Imprime100.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao44.zip

45 Compile e execute a classe **Imprime100**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc Imprime100.cs

C:\Users\K19\rafael\controle-de-fluxo> Imprime100.exe
1
2
3
...
100
```

Terminal 5.84: Compilando e executando a classe `Imprime100`

46 Na pasta **controle-de-fluxo**, crie um arquivo chamado **Imprime100ExcetoMultiplo3.cs**. Implemente um programa que exiba no terminal os números de 1 até 100 exceto os números múltiplos de 3.

```
1  class Imprime100ExcetoMultiplo3
2  {
3    static void Main()
4    {
```

```
5       for (int i = 1; i <= 100; i++)
6       {
7         if (i % 3 != 0)
8         {
9           System.Console.WriteLine(i);
10        }
11      }
12    }
13  }
```

Código C# 5.14: Imprime100ExcetoMultiplo3.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao46.zip

47 Compile e execute a classe **Imprime100ExcetoMultiplo3**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc Imprime100ExcetoMultiplo3.cs

C:\Users\K19\rafael\controle-de-fluxo> Imprime100ExcetoMultiplo3.exe
1
2
4
5
...
100
```

Terminal 5.85: Compilando e executando a classe `Imprime100ExcetoMultiplo3`

48 Na pasta **controle-de-fluxo**, crie um arquivo chamado **DivideMaiorInteiro.cs**. Implemente um programa que declare e inicialize uma variável que receberá o maior número possível do tipo **int**. Divida o valor dessa variável por 2 até que o resultado obtido seja inferior a 100 (não inclusivo). A cada iteração imprima o resultado.

```
1   class DivideMaiorInteiro
2   {
3     static void Main()
4     {
5       int numero = 2147483647;
6
7       while (numero >= 100)
8       {
9         numero /= 2;
10        System.Console.WriteLine(numero);
11      }
12    }
13  }
```

Código C# 5.15: DivideMaiorInteiro.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao48.zip

49 Compile e execute a classe **DivideMaiorInteiro**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc DivideMaiorInteiro.cs

C:\Users\K19\rafael\controle-de-fluxo> DivideMaiorInteiro.exe
1073741823
536870911
268435455
...
```

Terminal 5.86: Compilando e executando a classe `DivideMaiorInteiro`

50 Na pasta **controle-de-fluxo**, crie um arquivo chamado **GeradorDeIngressos.cs**. Implemente um programa para gerar os ingressos das apresentações de um teatro. Considere que esse teatro possui 4 setores e cada setor possui 20 lugares.

```
class GeradorDeIngressos
{
  static void Main()
  {
    for(int i = 1; i <= 4; i++)
    {
      for(int j = 1; j <= 20; j++)
      {
        System.Console.WriteLine("Setor: " + i + " Cadeira: " + j);
      }
    }
  }
}
```

Código C# 5.16: GeradorDeIngressos.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao50.zip

51 Compile e execute a classe **GeradorDeIngressos**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc GeradorDeIngressos.cs

C:\Users\K19\rafael\controle-de-fluxo> GeradorDeIngressos.exe
Setor 1 Cadeira 1
Setor 1 Cadeira 2
Setor 1 Cadeira 3
...
Setor 4 Cadeira 20
```

Terminal 5.87: Compilando e executando a classe `GeradorDeIngressos`

52 Na pasta **controle-de-fluxo**, crie um arquivo chamado **JogoDeDado.cs**. Implemente um programa que simula a execução de um jogo de dados, no qual são permitidos 5 lançamentos e o jogador ganha se a soma dos valores obtidos nos lançamentos ultrapassar 19.

```
class JogoDeDado
{
  static void Main()
  {
    int soma = 0;
    for(int i = 1; i <= 5; i++)
    {
      System.Random gerador = new System.Random();

      System.Console.WriteLine("Lançamento: " + i);
      int numero = (int)(gerador.NextDouble() * 6 + 1);

      System.Console.WriteLine("Número: " + numero);
      soma += numero;

      System.Console.WriteLine("Soma: " + soma);
      System.Console.WriteLine("----------------------");

      if(soma > 19)
      {
        System.Console.WriteLine("Você ganhou com " + i + " lançamentos");
        break;
      }
    }
```

```
25    }
26 }
```

Código C# 5.17: JogoDeDado.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao52.zip

53 Compile e execute algumas vezes a classe **JodoDeDado**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc JogoDeDado.cs

C:\Users\K19\rafael\controle-de-fluxo> JogoDeDado.exe
...
```

Terminal 5.88: Compilando e executando a classe JodoDeDado

54 Na pasta **controle-de-fluxo**, crie um arquivo chamado **JogoDeDado2.cs**. Implemente um programa que simula a execução de um jogo de dados, no qual são permitidos 4 lançamentos e o jogador ganha se a soma dos valores ímpares obtidos nos lançamentos ultrapassar 9.

```
1  class JogoDeDado2
2  {
3    static void Main()
4    {
5      int soma = 0;
6      for(int i = 1; i <= 5; i++)
7      {
8        System.Random gerador = new System.Random();
9
10       System.Console.WriteLine("Lançamento: " + i);
11       int numero = (int)(gerador.NextDouble() * 6 + 1);
12
13       System.Console.WriteLine("Número: " + numero);
14
15       if(numero % 2 == 0)
16       {
17         System.Console.WriteLine("-----------------------");
18         continue;
19       }
20
21       soma += numero;
22
23       System.Console.WriteLine("Soma: " + soma);
24       System.Console.WriteLine("-----------------------");
25
26       if(soma > 9)
27       {
28         System.Console.WriteLine("Você ganhou com " + i + " lançamentos");
29         break;
30       }
31     }
32   }
33 }
```

Código C# 5.18: JogoDeDado2.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao54.zip

55 Compile e execute algumas vezes a classe **JodoDeDado2**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc JogoDeDado2.cs

C:\Users\K19\rafael\controle-de-fluxo> JogoDeDado2.exe
...
```

Terminal 5.89: Compilando e executando a classe JodoDeDado2

56 Na pasta **controle-de-fluxo**, crie um arquivo chamado **BartChalkboard.cs**. Implemente um programa para ajudar o Bart Simpson a cumprir o seu castigo.

```
1 class BartChalkboard
2 {
3   static void Main()
4   {
5     for(;;)
6       System.Console.WriteLine("I WILL NOT XEROX MY BUTT");
7   }
8 }
```

Código C# 5.19: BartChalkboard.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-fixacao56.zip

57 Compile e execute a classe **BartChalkboard**.

```
C:\Users\K19\rafael\controle-de-fluxo> csc BartChalkboard.java

C:\Users\K19\rafael\controle-de-fluxo> BartChalkboard.exe
I WILL NOT XEROX MY BUTT
I WILL NOT XEROX MY BUTT
I WILL NOT XEROX MY BUTT
...
```

Terminal 5.90: Compilando e executando a classe BartChalkboard

Exercícios Complementares

1 Crie um programa em Java que gere dois números aleatórios e exiba o valor desses números. Além disso, esse programa deve exibir a mensagem "Primeiro > Segundo" se o primeiro número for maior do que o segundo, a mensagem "Segundo > Primeiro" se o segundo número for maior do que o primeiro e mensagem "Primeiro = Segundo" se o primeiro número for igual ao segundo. Complete o código abaixo.

```
1  class ComparaValores {
2    public static void main(String[] args) {
3      double primeiro = Math.random();
4      double segundo = Math.random();
5
6      System.out.println("Primeiro: " + primeiro);
7      System.out.println("Segundo: " + segundo);
8
9
10   }
11 }
```

Código Java 5.118: ComparaValores.java

2 Crie um programa em Java que exiba o seguinte padrão no terminal.

```
******************
******************
******************
******************
******************
```

Utilize laços de repetição e complete o código abaixo.

```
1 class BlocoDeAstericos {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.120: BlocoDeAstericos.java

3 Crie um programa em Java que exiba o seguinte padrão no terminal.

```
*
**
***
****
*****
******
```

Utilize laços de repetição e complete o código abaixo.

```
1 class TrianguloDeAstericos {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.122: TrianguloDeAstericos.java

4 Crie um programa em Java que exiba o seguinte padrão no terminal.

```
*
**
***
****
*****
******
*
**
***
```

```
****
*****
******
*
**
***
****
*****
******
```

Utilize laços de repetição e complete o código abaixo.

```
1 class TresTriangulosDeAstericos {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.124: TresTriangulosDeAstericos.java

5 Crie um programa em Java que exiba o seguinte padrão no terminal.

```
**********
 **********
  **********
   **********
    **********
```

Utilize laços de repetição e complete o código abaixo.

```
1 class LosangoDeAstericos {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.126: LosangoDeAstericos.java

6 Crie um programa em Java que exiba o seguinte padrão no terminal.

```
**********
 **********
  **********
   **********
    **********
**********
 **********
  **********
   **********
```

```
    **********
**********
 **********
  **********
   **********
    **********
```

Utilize laços de repetição e complete o código abaixo.

```
1 class TresLlosangosDeAstericos {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.128: TresLosangosDeAstericos.java

7 Para controlar o estacionamento de um condomínio, devemos implementar um programa em Java para gerar os cartões das vagas dos moradores. Nos cartões, é necessário constar o número do bloco e o número do apartamento. Nesse condômino, há três blocos numerados de 1 a 3. Em cada bloco, há 9 andares. Em cada andar, há 4 apartamentos. No primeiro andar, os números dos apartamentos são: 11, 12, 13 e 14. No segundo andar, os números dos apartamentos são: 21, 22, 23 e 24. Nos outros andares, a lógica de numeração é a mesma. Complete o código a seguir.

```
1 class CartoesDeEstacionamento {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.130: CartoesDeEstacionamento.java

8 Escreva um programa em Java que exiba no terminal a tabuada dos números de 1 a 10 de acordo com o padrão abaixo.

```
1x1 = 1
1x2 = 2
1x3 = 3
...
10x8 = 80
10x9 = 90
10x10 = 100
```

Complete o código a seguir.

```
1 class Tabuada {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.132: Tabuada.java

9 Escreva um programa que desenhe uma pirâmide de asteriscos. A saída do seu programa deve seguir o padrão abaixo:

```
   *
  ***
 *****
*******
```

Complete o código abaixo.

```
1 class Piramide {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.134: Piramide.java

10 Escreva um programa que imprime uma árvore de natal de acordo com o padrão abaixo.

```
      *
     ***
    *****
   *******
    *****
   *******
  *********
 ***********
  *********
 ***********
*************
***************
```

Complete o código abaixo.

```
1 class ArvoreNatal {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 5.136: ArvoreNatal.java

11 Crie um programa em Java que gere e exiba no terminal um número aleatório. Além disso, esse programa deve contabilizar a quantidade de dígitos "1" do número aleatório e exibir essa quantidade no terminal. Complete o código abaixo.

```
1 class ContaUns {
2   public static void main(String[] args) {
3     double numero = Math.random();
```

```
4       System.out.println(numero);
5
6       String s = "" + numero;
7       int resposta = 0;
8
9
10
11      System.out.println(resposta);
12    }
13  }
```

Código Java 5.138: ContaUns.java

12 Considere um jogo no qual o jogador lança um dado 10 vezes. O jogador ganha se a soma dos valores obtidos nos lançamentos for ímpar. Se o número 6 for sorteado 2 vezes o jogador perde imediatamente. O número 1 deve ser desconsiderado na somatória. Implemente um programa em Java para simular a execução desse jogo.

```
1  class JogoDaSomaImpar {
2    public static void main(String[] args) {
3
4    }
5  }
```

Código Java 5.140: JogoDaSomaImpar.java

13 Crie um programa em C# que gere dois números aleatórios e exiba o valor desses números. Além disso, esse programa deve exibir a mensagem "Primeiro > Segundo" se o primeiro número for maior do que o segundo, a mensagem "Segundo > Primeiro" se o segundo número for maior do que o primeiro e mensagem "Primeiro = Segundo" se o primeiro número for igual ao segundo. Complete o código abaixo.

```
1  class ComparaValores
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      double primeiro = gerador.NextDouble();
8      double segundo = gerador.NextDouble();
9
10     System.Console.WriteLine("Primeiro: " + primeiro);
11     System.Console.WriteLine("Segundo: " + segundo);
12
13
14   }
15 }
```

Código C# 5.20: ComparaValores.cs

14 Crie um programa em C# que exiba o seguinte padrão no terminal.

```
**********
**********
**********
**********
```

****

Utilize laços de repetição e complete o código abaixo.

```
1 class BlocoDeAstericos
2 {
3   static void Main()
4 {
5
6   }
7 }
```

Código C# 5.22: BlocoDeAstericos.cs

15 Crie um programa em C# que exiba o seguinte padrão no terminal.

```
*
**
***
****
*****
******
```

Utilize laços de repetição e complete o código abaixo.

```
1 class TrianguloDeAstericos
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 5.24: TrianguloDeAstericos.cs

16 Crie um programa em C# que exiba o seguinte padrão no terminal.

```
*
**
***
****
*****
******
*
**
***
****
*****
******
*
```

```
**
***
****
*****
******
```

Utilize laços de repetição e complete o código abaixo.

```
1 class TresTriangulosDeAstericos
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 5.26: TresTriangulosDeAstericos.cs

17 Crie um programa em C# que exiba o seguinte padrão no terminal.

```
**********
 **********
  **********
   **********
    **********
```

Utilize laços de repetição e complete o código abaixo.

```
1 class LosangoDeAstericos
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 5.28: LosangoDeAstericos.cs

18 Crie um programa em C# que exiba o seguinte padrão no terminal.

```
**********
 **********
  **********
   **********
    **********
**********
 **********
  **********
   **********
    **********
```

```
**********
 **********
  **********
   **********
    **********
```

Utilize laços de repetição e complete o código abaixo.

```
1 class TresLosangosDeAstericos
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 5.30: TresLosangosDeAstericos.cs

19 Para controlar o estacionamento de um condomínio, devemos implementar um programa em C# para gerar os cartões das vagas dos moradores. Nos cartões, é necessário constar o número do bloco e o número do apartamento. Nesse condômino, há três blocos numerados de 1 a 3. Em cada bloco, há 9 andares. Em cada andar, há 4 apartamentos. No primeiro andar, os números dos apartamentos são: 11, 12, 13 e 14. No segundo andar, os números dos apartamentos são: 21, 22, 23 e 24. Nos outros andares, a lógica de numeração é a mesma. Complete o código a seguir.

```
1 class CartoesDeEstacionamento
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 5.32: CartoesDeEstacionamento.cs

20 Escreva um programa em C# que exiba no terminal a tabuada dos números de 1 a 10 de acordo com o padrão abaixo.

```
1x1 = 1
1x2 = 2
1x3 = 3
...
10x8 = 80
10x9 = 90
10x10 = 100
```

Complete o código a seguir.

```
1 class Tabuada
2 {
3   static void Main()
4   {
```

```
5
6     }
7 }
```

Código C# 5.34: Tabuada.cs

21 Escreva um programa que desenhe uma pirâmide de asteriscos. A saída do seu programa deve seguir o padrão abaixo:

```
   *
  ***
 *****
*******
```

Complete o código abaixo.

```
1 class Piramide
2 {
3     static void Main()
4     {
5
6     }
7 }
```

Código C# 5.36: Piramide.cs

22 Escreva um programa que imprime uma árvore de natal de acordo com o padrão abaixo.

```
      *
     ***
    *****
   *******
    *****
   *******
  *********
 ***********
  *********
 ***********
*************
***************
```

Complete o código abaixo.

```
1 class ArvoreNatal
2 {
3     static void Main()
4     {
5
6     }
7 }
```

Código C# 5.38: ArvoreNatal.cs

23 Crie um programa em C# que gere e exiba no terminal um número aleatório. Além disso, esse programa deve contabilizar a quantidade de dígitos "1" do número aleatório e exibir essa quantidade no terminal. Complete o código abaixo.

```
class ContaUns
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    double numero = gerador.NextDouble();
    System.Console.WriteLine(numero);

    String s = "" + numero;
    int resposta = 0;

    

    System.Console.WriteLine(resposta);
  }
}
```

Código C# 5.40: ContaUns.cs

24 Considere um jogo no qual o jogador lança um dado 10 vezes. O jogador ganha se a soma dos valores obtidos nos lançamentos for ímpar. Se o número 6 for sorteado 2 vezes o jogador perde imediatamente. O número 1 deve ser desconsiderado na somatória. Implemente um programa em C# para simular a execução desse jogo.

```
class JogoDaSomaImpar
{
  static void Main()
  {
    
  }
}
```

Código C# 5.42: JogoDaSomaImpar.cs

Resumo do Capítulo

1 Os programas de computador utilizam as instruções de decisão para definir se um bloco de código será executado ou não de acordo com determinada condição.

2 Para utilizar a instrução de decisão **if**, devemos definir uma **condição** e um **corpo**. O corpo é um bloco de código executado somente se a condição for **true**.

3 Para utilizar a instrução **else**, devemos definir um corpo. Essa instrução sempre está relacionada à instrução **if**. O corpo do **else** é executado somente se a condição do **if** correspondente for **false**.

4 Instruções de decisão podem ser encadeadas. Em outra palavras, podemos definir **if**s e **else**s dentro de **if**s ou **else**s.

5 Os programas de computador utilizam as instruções de repetição para executar repetidas vezes um determinado bloco de código.

6 Para utilizar a instrução de repetição **while**, devemos definir uma **condição** e um **corpo**. O corpo é executado enquanto a condição for **true**. Toda vez, depois que o corpo é executado, a condição é reavaliada para decidir se o laço deve ou não continuar.

7 Para utilizar a instrução de repetição **for**, devemos definir uma **inicialização**, uma **condição**, uma **atualização** e um **corpo**. O fluxo de execução começa pela inicialização e, enquanto a condição for **true**, ele executa de forma cíclica a condição, o corpo e a atualização nessa ordem.

8 Instruções de repetição podem ser encadeadas. Em outra palavras, podemos definir **while**s e **for**s dentro de **while**s ou **for**s.

9 A instrução **break** interrompe a execução de um laço.

10 A instrução **continue** interrompe a execução de uma iteração. No **while**, o **continue** desvia o fluxo de execução para a condição. No **for**, o **continue** desvia o fluxo de execução para a atualização.

11 Quando o corpo do **if** possui apenas um comando, ele não precisa ser delimitado com **chaves**. A mesma regra vale para o **else**, **while** e **for**.

12 Se a condição de um laço sempre é **true**, o corpo desse laço será executado repetidamente sem parar("laços infinitos").

Prova

1 Qual alternativa está correta?

a) **if** e **else** são instruções de decisão.

b) **if** e **while** são instruções de decisão.

c) **else** e **while** são instruções de decisão.

d) **while** e **for** são instruções de decisão.

e) **if** e **for** são instruções de decisão.

2 Qual alternativa está correta?

a) No **if**, a condição pode ser um valor booleano ou numérico.

b) No **if**, a condição só pode ser um valor numérico.

c) No **if**, a condição só pode ser um valor booleano.

d) No **if**, a condição pode ser uma string ou um valor numérico.

e) No **if**, a condição pode ser qualquer coisa.

3 Qual alternativa está correta?

a) Para cada **if**, tem que existir um **else**.

b) O corpo do **else** é executado quando a condição do **if** é verdadeira.

c) O corpo do **if** é executado quando a condição é falsa.

d) Não podemos definir **if**s no corpo dos **else**s.

e) Para cada **else**, tem que existir um **if**.

4 Considere o seguinte código.

```
1  int a = 1;
2  int b = 1;
3
4  if(a++ > b) {
5    if(a > --b) {
6      a = 10
7    } else
8      a = 11;
9    a = 12;
10 } else
11   if(a > --b)
12     a = 13;
13   else {
14     a = 14;
15   }
```

Ao final desse código, qual é o valor da variável **a**.

a) 10

b) 11

c) 12

d) 13

e) 14

5 Qual alternativa está correta?

a) **if** e **else** são instruções de repetição.

b) **if** e **while** são instruções de repetição.

c) **else** e **while** são instruções de repetição.

d) **while** e **for** são instruções de repetição.

e) **if** e **for** são instruções de repetição.

6 Qual alternativa está correta?

a) No **while** e **for**, a condição só pode ser um valor numérico.

b) No **while** e **for**, a condição pode ser um valor booleano ou numérico.

c) No **while** e **for**, a condição pode ser uma string ou um valor numérico.

d) No **while** e **for**, a condição pode ser qualquer coisa.

e) No **while** e **for**, a condição só pode ser um valor booleano.

7 Considere o seguinte código.

```
1 int a = 1;
2
3 while(a > 10) {
4   a++;
5 }
```

Ao final desse código, qual é o valor da variável **a**.

a) 1

b) 2

c) 9

d) 10

e) 11

8 Considere o seguinte código.

```
1 int a = 1;
2
3 for(int i = 10; i > 0; i--) {
4   a += 2;
5 }
```

Ao final desse código, qual é o valor da variável **a**.

a) 1

b) 11

c) 12

d) 21

e) 23

9 Considere o seguinte código.

```
1  int a = 0;
2
3  for(int i = 1; i < 10; i++) {
4    if(i == 3 || i == 5) {
5      continue;
6    }
7
8    if(i == 9) {
9      break;
10   }
11
12   a += i;
13 }
```

Ao final desse código, qual é o valor da variável **a**.

a) 0

b) 3

c) 8

d) 28

e) 36

10 Considere o seguinte código.

```
1  int a = 0;
2  for(int i = 0; i < 10; i++) {
3    for(int j = 0; j < 10; j++) {
4      a++;
5    }
6  }
```

Ao final desse código, qual é o valor da variável **a**.

a) 9

b) 10

c) 18

d) 20

e) 100

11 Considere o seguinte código.

```
int a = 0;
for(int i = 0; i < 10; i++) {
  for(int j = 0; j < 10; j++) {
    if(i == j) {
      continue;
    }
    a++;
  }
}
```

Ao final desse código, qual é o valor da variável **a**.

a) 18

b) 50

c) 90

d) 99

e) 100

Minha Pontuação [ ] Pontuação Mínima: [8] Pontuação Máxima: [11]

ARRAY

CAPÍTULO 6

Considere um programa de computador que realizará cálculos matemáticos com os preços dos produtos de um supermercado. Por exemplo, esse programa deverá calcular a média dos preços ou encontrar o produto mais barato.

Para manipular os preços dos produtos, dentro de um programa, esses valores devem ser armazenados em variáveis.

```
1 double preco1;
2 double preco2;
3 double preco3;
4 ...
```

Como uma variável do tipo **double** armazena somente um valor de cada vez, seria necessário uma variável para cada produto. Considerando a existência de uma grande quantidade de produtos, essa abordagem será pouco prática. Nesses casos, podemos utilizar os chamados **arrays** ou **vetores**.

O que é um Array?

Um array ou vetor é uma estrutura de dados utilizada para armazenar uma coleção de itens. Cada item é identificado através de um índice. Podemos imaginar um array como sendo um armário com um determinado número de gavetas e cada gaveta possui um rótulo com um número de identificação.

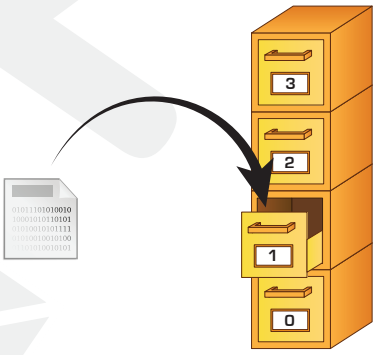

Figura 6.1: Abstração de array como um armário com gavetas

Quando criamos um array, devemos informar qual o tipo de dado pretendemos armazenar em cada posição. Na analogia com armário, seria como se tivéssemos que definir o que o é permitido guardar em cada gaveta. Por exemplo, se definirmos que um armário deve guardar livros, então somente livros podem ser armazenados nas gavetas desse armário. Não poderemos guardar revistas ou jornais.

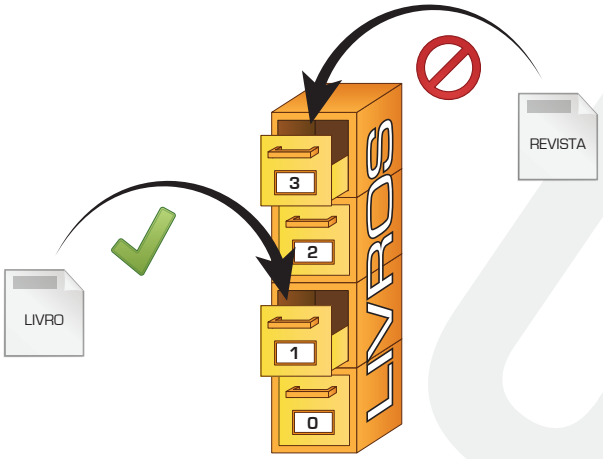

Figura 6.2: Um armário de livros não pode guardar revistas

Como declarar e inicializar um array?

Para utilizarmos um array, devemos criar uma variável para guardar a referência desse array. A declaração dessa variável é semelhante à declaração das variáveis que vimos até agora.

```
1 int[] nomeDoArray;
```

Código Java 6.2: Declaração de um array

Lembre-se que sempre devemos inicializar as variáveis para não ocorrer um erro de compilação. Portanto, vamos inicializar o nosso array:

```
1 int[] nomeDoArray = new int[10];
```

Código Java 6.3: Declaração e inicialização de um array

A inicialização de um array se dá através da instrução **new** tanto em Java quanto em C#. No exemplo acima, criamos um array de tamanho 10, ou seja, teremos 10 posições para armazenar valores do tipo **int**. A instrução **new** é abordada com mais detalhes nos cursos "K11 - Orientação a Objetos em Java" e "K31 - C# e Orientação a Objetos".

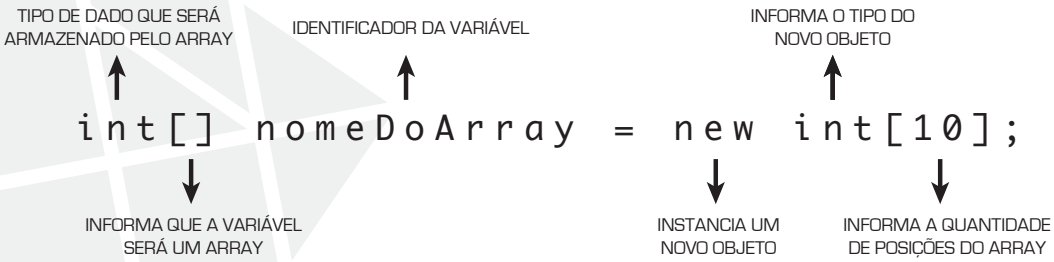

Figura 6.3: Declaração e inicialização de um array

Inserindo valores de um array

Existem diversas formas de inserirmos valores em um array. A forma mais comum é a seguinte:

```
1 int[] a = new int[3];
2 a[0] = 124;
3 a[1] = 43;
4 a[2] = 1023;
```

Código Java 6.4: Inserindo valores em um array

Na linha 1 declaramos e inicializamos um array do tipo **int** com três posições. Nas linhas 2, 3 e 4 inserimos no array os valores 124, 43 e 1023 nas posições 0, 1 e 2 respectivamente. Repare que a numeração dos índices de um array começa pelo número zero.

As outras formas de se inserir valores em um array fazem muito mais do que simplesmente inserir tais valores. Na verdade essas formas declaram, inicializam e inserem os valores, tudo em apenas uma linha de código.

```
1 int[] b = new int[] {1, 62, 923, 15};
2 int[] c = {125, 76432, 23};
```

Código Java 6.5: Outras formas de se inserir valores em um array

```
1 int[] b = new int[] {1, 62, 923, 15};
2 int[] c = {125, 76432, 23};
3 int[] d = new int[2] {634, 5};
```

Código C# 6.1: Outras formas de se inserir valores em um array

Repare que, no momento da criação dos arrays acima, os valores de cada posição devem ser definidos. Já na primeira forma apresentada, esses valores poderiam ser definidos depois.

Acessando os valores de um array

Para acessarmos o valor armazenado em uma das posições de um array, basta conhecermos o índice de tal posição. Veja o exemplo abaixo:

```
1 int[] a = new int[] {3215, 754, 23};
2
3 System.out.println("Valor na posição de índice 0: " + a[0]);
4 System.out.println("Valor na posição de índice 2: " + a[2]);
```

Código Java 6.6: Acessando os valores de um array

```
1 int[] a = new int[] {3215, 754, 23};
2
3 System.Console.WriteLine("Valor na posição de índice 0: " + a[0]);
4 System.Console.WriteLine("Valor na posição de índice 2: " + a[2]);
```

Código C# 6.2: Acessando os valores de um array.

Percorrendo um array

Quando trabalhamos com arrays, uma das tarefas mais comuns é acessarmos todas ou algumas de suas posições sistematicamente. Geralmente, fazemos isso para resgatar todos ou alguns dos valores armazenados e realizar algum processamento sobre tais valores.

Para percorrermos um array utilizaremos a instrução de repetição **for**. Podemos utilizar a instrução **while** também, porém logo perceberemos que a sintaxe da instrução **for** é mais apropriada quando estamos trabalhando com arrays.

```
1 int[] numeros = new int[100];
2
3 for(int i = 0; i < 100; i++) {
4   numeros[i] = i * 2;
5 }
6
7 for(int i = 0; i < 100; i++) {
8   System.out.println(numeros[i]);
9 }
```

Código Java 6.7: Percorrendo um array para inserir e acessar valores

```
1 int[] numeros = new int[100];
2
3 for(int i = 0; i < 100; i++)
4 {
5   numeros[i] = i * 2;
6 }
7
8 for(int i = 0; i < 100; i++)
9 {
10   System.Console.WriteLine(numeros[i]);
11 }
```

Código C# 6.3: Percorrendo um array para inserir e acessar valores

Imagine que exista uma grande quantidade de linhas de código entre as linhas destacadas no exemplo acima, ou seja, entre a inicialização do array **numeros** e o **for** que o percorre. Além disso, imagine também que o código tenha que ser modificado, mais especificamente, a quantidade de posições deve ser alterada de 100 para 1000. Seria muito fácil esquecermos de atualizar os argumentos da instrução **for** para que o laço considere o intervalo de 0 a 1000 e não o intervalo de 0 a 100.

Para evitar esse tipo de problema, uma boa prática é utilizar o atributo **length** dos arrays da linguagem Java ou a propriedade **Length** dos arrays da linguagem C# para descobrir qual a sua dimensão (tamanho) do array. Veja como ficaria o exemplo com essas modificações.

```
1 int[] numeros = new int[100];
2
3 for(int i = 0; i < numeros.length; i++) {
4   numeros[i] = i*2;
5 }
6
7 for(int i = 0; i < numeros.length; i++) {
8   System.out.println(numeros[i]);
9 }
```

Código Java 6.8: Utilizando o atributo `length` *do array*

```
1 int[] numeros = new int[100];
2
3 for(int i = 0; i < numeros.Length; i++)
4 {
5   numeros[i] = i * 2;
```

```
6  }
7
8  for(int i = 0; i < numeros.Length; i++)
9  {
10   System.Console.WriteLine(numeros[i]);
11 }
```

Código C# 6.4: Utilizando a propriedade `Length` *do array*

Array de arrays

Até agora trabalhamos com arrays de uma dimensão. Porém, tanto em Java como em C#, podemos criar arrays com mais de uma dimensão (arrays multidimensionais). Isso nos permite trabalhar com arrays para representar tabelas, matrizes ou até um tabuleiro de batalha naval. Voltando à analogia que fizemos com um armário cheio de gavetas, seria como se pudéssemos guardar dentro da gaveta de um armário um outro armário com gavetas. Veja a figura abaixo:

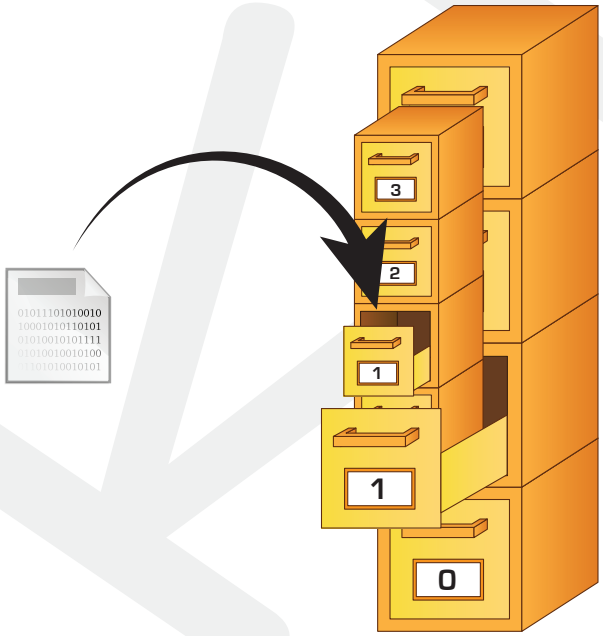

Figura 6.4: Abstração de um array multidimensional

Importante
Em Java, array multidimensional é sinônimo de array de arrays. Em C#, jagged arrays é sinônimo de array de arrays. Em C# a denominação array multidimensional refere-se a um outro tipo de array. Um array multidimensional em C# é assunto do curso "K31 - C# e Orientação a Objetos".

Neste curso, para evitarmos confusões, sempre utilizaremos a denominação array de arrays.

A declaração de um array de arrays é muito semelhante à declaração e inicialização de um array simples.

```
int[][] arrays = new int[4][];

arrays[0] = new int[1];
arrays[1] = new int[3];
arrays[2] = new int[2];
arrays[3] = new int[7];
```

Código Java 6.9: Declarando um array de arrays

Em cada posição do nosso array de arrays, devemos criar um novo array. Por esse motivo, ele recebe o nome array de arrays. Além disso, repare que podemos criar arrays de diferentes tamanhos em cada posição.

Assim como nos arrays unidimensionais, para inserir ou acessar valores de um array de arrays, devemos utilizar os índices de cada posição. Podemos pensar nos índices como um esquema de coordenadas. Por exemplo, se quiséssemos representar um gráfico no sistema cartesiano de eixos xy através de um array de arrays, a coordenada de cada ponto do gráfico seria equivalente ao par de índices do nosso array de arrays (supondo que no gráfico seja permitido apenas coordenadas inteiras).

```
boolean[][] pontosDoGrafico = new boolean[10][];

for(int i = 0; i < pontosDoGrafico.length; i++){
  pontosDoGrafico[i] = new boolean[10];
}

pontosDoGrafico[0][0] = true;
pontosDoGrafico[1][1] = true;
pontosDoGrafico[2][1] = true;
pontosDoGrafico[2][2] = true;
pontosDoGrafico[3][2] = true;
pontosDoGrafico[4][1] = true;
```

Código Java 6.10: Conjunto de pontos de um gráfico de eixos xy armazenados em um array de arrays

```
bool[][] pontosDoGrafico = new bool[10][];

for(int i = 0; i < pontosDoGrafico.Length; i++)
{
  pontosDoGrafico[i] = new bool[10];
}

pontosDoGrafico[0][0] = true;
pontosDoGrafico[1][1] = true;
pontosDoGrafico[2][1] = true;
pontosDoGrafico[2][2] = true;
pontosDoGrafico[3][2] = true;
pontosDoGrafico[4][1] = true;
```

Código C# 6.5: Conjunto de pontos de um gráfico de eixos xy armazenados em um array de arrays

Percorrendo um array de arrays

Para percorrer um array de arrays, utilizaremos novamente as instruções de repetição **while** e **for**. Porém, como estamos trabalhando com arrays com mais de uma dimensão, teremos uma ou mais laços encadeados.

```
int[][] tabelaDeNumeros = new int[5][];

for(int i = 0; i < tabelaDeNumeros.length; i++) {
  tabelaDeNumeros[i] = new int[5];
}

for(int i = 0; i < tabelaDeNumeros.length; i++) {
  for(int j = 0; j < tabelaDeNumeros[i].length; j++) {
    tabelaDeNumeros[i][j] = i*j;
  }
}
```

Código Java 6.11: Percorrendo um array de arrays com instruções `for` *aninhadas*

Exercícios de Fixação Com Java

1 Abra um terminal; Entre na pasta dos seus exercícios e crie uma pasta chamada **arrays** para os arquivos desenvolvidos nesse capítulo.

```
K19/rafael$ mkdir arrays

K19/rafael$ cd arrays

K19/rafael/arrays$
```

Terminal 6.1: Criando a pasta `arrays` *no Linux*

```
C:\Users\K19\rafael> md arrays

C:\Users\K19\rafael> cd arrays

C:\Users\K19\rafael\arrays>
```

Terminal 6.2: Criando a pasta `arrays` *no Windows*

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao1.zip

2 Na pasta **arrays**, crie um arquivo chamado **SequenciaQualquer.java**. Implemente um programa em Java que armazene 10 números inteiros em um array. Todas as posições do array devem ser preenchidas e o valor armazenado fica à sua escolha. Após preencher o array, exiba os seus valores no terminal.

```
class SequenciaQualquer {
  public static void main(String[] args) {
    int[] array = new int[10];

    array[0] = 57;
    array[1] = 436;
    array[2] = 724;
    array[3] = 564;
    array[4] = 245;
    array[5] = 47;
    array[6] = 34;
    array[7] = 1;
    array[8] = 347735;
    array[9] = 83;

    for(int i = 0; i < array.length; i++) {
      System.out.println(array[i]);
```

```
18      }
19    }
20  }
```

Código Java 6.12: SequenciaQualquer.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao2.zip

3 Compile e execute a classe **SequenciaQualquer**.

```
K19/rafael/arrays$ javac SequenciaQualquer.java

K19/rafael/arrays$ java SequenciaQualquer
57
436
724
564
245
47
34
1
347735
83
```

Terminal 6.3: Compilando e executando a classe `SequenciaQualquer`

4 Na pasta **arrays**, crie um arquivo chamado **SequenciaCrescente.java**. Implemente um programa em Java que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais. Ao final, exiba no terminal esse valores.

```
1  class SequenciaCrescente {
2    public static void main(String[] args) {
3      int[] array = new int[10];
4
5      for (int i = 0; i < array.length; i++) {
6        array[i] = i;
7      }
8
9      for (int i = 0; i < array.length; i++) {
10       System.out.println(array[i]);
11     }
12   }
13 }
```

Código Java 6.13: SequenciaCrescente.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao4.zip

5 Compile e execute a classe **SequenciaCrescente**.

```
K19/rafael/arrays$ javac SequenciaCrescente.java

K19/rafael/arrays$ java SequenciaCrescente
0
1
2
3
4
5
6
7
8
9
```

Terminal 6.4: Compilando e executando a classe `SequenciaCrescente`

6 Na pasta **arrays**, crie um arquivo chamado **SequenciaDecrescente.java**. Implemente um programa em Java que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais decrescentes. Ao final, exiba no terminal esse valores.

```
1  class SequenciaDecrescente {
2    public static void main(String[] args) {
3      int[] array = new int[10];
4
5      for (int i = 0; i < array.length; i++) {
6        array[i] = array.length - 1 - i;
7      }
8
9      for (int i = 0; i < array.length; i++) {
10       System.out.println(array[i]);
11     }
12   }
13 }
```

Código Java 6.14: SequenciaDecrescente.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao6.zip

7 Compile e execute a classe **SequenciaDecrescente**.

```
K19/rafael/arrays$ javac SequenciaDecrescente.java

K19/rafael/arrays$ java SequenciaDecrescente
9
8
7
6
5
4
3
2
1
0
```

Terminal 6.5: Compilando e executando a classe `SequenciaDecrescente`

8 Na pasta **arrays**, crie um arquivo chamado **SequenciaImpar.java**. Implemente um programa em Java que armazene 10 números inteiros ímpares em um array. Ao final, exiba no terminal esse valores.

```
1  class SequenciaImpar {
2    public static void main(String[] args) {
3      int[] array = new int[10];
4
5      for (int i = 0; i < array.length; i++) {
6        array[i] = 2 * i + 1;
7      }
8
9      for (int i = 0; i < array.length; i++) {
10       System.out.println(array[i]);
11     }
12   }
13 }
```

Código Java 6.15: SequenciaImpar.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao8.zip

9 Compile e execute a classe **SequenciaImpar**.

```
K19/rafael/arrays$ javac SequenciaImpar.java

K19/rafael/arrays$ java SequenciaImpar
1
3
5
7
9
11
13
15
17
19
```

Terminal 6.6: Compilando e executando a classe `SequenciaImpar`

10 Na pasta **arrays**, crie um arquivo chamado **SequenciaAleatoria.java**. Implemente um programa em Java que armazene 10 números inteiros aleatórios em um array. Ao final, exiba no terminal esse valores.

```
1  class SequenciaAleatoria {
2    public static void main(String[] args) {
3      int[] array = new int[10];
4
5      for (int i = 0; i < array.length; i++) {
6        array[i] = (int)(Math.random() * 100);
7      }
8
9      for (int i = 0; i < array.length; i++) {
10       System.out.println(array[i]);
11     }
12   }
13 }
```

Código Java 6.16: SequenciaAleatoria.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao10.zip

11 Compile e execute a classe **SequenciaAleatoria**.

```
K19/rafael/arrays$ javac SequenciaAleatoria.java

K19/rafael/arrays$ java SequenciaAleatoria
20
76
88
45
39
91
33
25
18
70
```

Terminal 6.7: Compilando e executando a classe `SequenciaAleatoria`

12 Na pasta **arrays**, crie um arquivo chamado **TabelaQualquer.java**. Implemente um programa em Java que armazene números inteiros em um array de arrays. Ao final, exiba no terminal esse valores.

```
class TabelaQualquer {
  public static void main(String[] args) {
    int[][] array = new int[3][3];

    array[0][0] = 19;
    array[0][1] = 22;
    array[0][2] = 31;
    array[1][0] = 2;
    array[1][1] = 51;
    array[1][2] = 12;
    array[2][0] = 41;
    array[2][1] = 11;
    array[2][2] = 3;


    for (int i = 0; i < array.length; i++) {
      for (int j = 0; j < array[i].length; j++) {
        System.out.println(array[i][j]);
      }
    }
  }
}
```

Código Java 6.17: TabelaQualquer.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao12.zip

13 Compile e execute a classe **TabelaQualquer**.

```
K19/rafael/arrays$ javac TabelaQualquer.java

K19/rafael/arrays$ java TabelaQualquer
19
22
31
2
51
12
41
11
3
```

Terminal 6.8: Compilando e executando a classe `TabelaQualquer`

14 Na pasta **arrays**, crie um arquivo chamado **TabelaAleatoria.java**. Implemente um programa em Java que armazene números inteiros em um array de arrays. Ao final, exiba no terminal esse valores.

```
class TabelaAleatoria {
  public static void main(String[] args) {
    int[][] array = new int[3][3];

    for (int i = 0; i < array.length; i++) {
      for (int j = 0; j < array[i].length; j++) {
        array[i][j] = (int)(Math.random() * 100);
      }
    }

    for (int i = 0; i < array.length; i++) {
      for (int j = 0; j < array[i].length; j++) {
        System.out.println(array[i][j]);
      }
    }
```

```
16    }
17 }
```

Código Java 6.18: TabelaAleatoria.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao14.zip

15 Compile e execute a classe **TabelaAleatoria**.

```
K19/rafael/arrays$ javac TabelaAleatoria.java

K19/rafael/arrays$ java TabelaAleatoria
35
72
13
47
6
74
47
30
27
```

Terminal 6.9: Compilando e executando a classe `TabelaAleatoria`

16 Na pasta **arrays**, crie um arquivo chamado **Tabuada.java**. Implemente um programa em Java que armazene números inteiros em um array de arrays. Ao final, exiba no terminal esse valores.

```
1  class Tabuada {
2    public static void main(String[] args) {
3      int[][] tabuada = new int[10][10];
4
5      for (int i = 0; i < tabuada.length; i++) {
6        for (int j = 0; j < tabuada[i].length; j++) {
7          tabuada[i][j] = (i + 1) * (j + 1);
8        }
9      }
10
11     for (int i = 0; i < tabuada.length; i++) {
12       for (int j = 0; j < tabuada[i].length; j++) {
13         System.out.println((i + 1) + " x " + (j + 1) + " = " + tabuada[i][j]);
14       }
15     }
16   }
17 }
```

Código Java 6.19: Tabuada.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao16.zip

17 Compile e execute a classe **Tabuada**.

```
K19/rafael/arrays$ javac Tabuada.java

K19/rafael/arrays$ java Tabuada
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
...
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

Terminal 6.10: Compilando e executando a classe `Tabuada`

Erro: Acessar uma posição inexistente

Um erro de **execução** comum em Java ou C# ocorre quando tentamos acessar uma posição que não existe em array.

```
1 class Programa {
2   public static void main(String[] args) {
3     int[] array = new int[10];
4
5     array[10] = 10;
6   }
7 }
```

Código Java 6.20: Programa.java

A mensagem de erro de **execução** seria semelhante a apresenta abaixo.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
  at Programa.main(Programa.java:5)
```

Terminal 6.11: Erro de execução

Agora, veja um exemplo de programa em C# com esse problema.

```
1 class Programa
2 {
3   static void Main()
4   {
5     int[] array = new int[10];
6
7     array[10] = 10;
8   }
9 }
```

Código C# 6.6: Programa.cs

A mensagem de erro de **execução** seria semelhante a apresenta abaixo.

```
Unhandled Exception: System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at Programa.Main()
```

Terminal 6.12: Erro de execução

Exercícios de Fixação Com C#

18 Na pasta **arrays**, crie um arquivo chamado **SequenciaQualquer.cs**. Implemente um programa em Java que armazene 10 números inteiros em um array. Todas as posições do array devem ser preenchidas e o valor armazenado fica à sua escolha. Após preencher o array, exiba os seus valores no terminal.

```
1 class SequenciaQualquer
2 {
3   static void Main()
4   {
```

```
5       int[] array = new int[10];
6
7       array[0] = 57;
8       array[1] = 436;
9       array[2] = 724;
10      array[3] = 564;
11      array[4] = 245;
12      array[5] = 47;
13      array[6] = 34;
14      array[7] = 1;
15      array[8] = 347735;
16      array[9] = 83;
17
18      for(int i = 0; i < array.Length; i++)
19      {
20        System.Console.WriteLine(array[i]);
21      }
22    }
23  }
```

Código C# 6.7: SequenciaQualquer.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao18.zip

19 Compile e execute a classe **SequenciaQualquer**.

```
C:\Users\K19\rafael\arrays> csc SequenciaQualquer.cs

C:\Users\K19\rafael\arrays> SequenciaQualquer.exe
57
436
724
564
245
47
34
1
347735
83
```

Terminal 6.13: Compilando e executando a classe `SequenciaQualquer`

20 Na pasta **arrays**, crie um arquivo chamado **SequenciaCrescente.cs**. Implemente um programa em Java que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais. Ao final, exiba no terminal esse valores.

```
1   class SequenciaCrescente
2   {
3     static void Main()
4     {
5       int[] array = new int[10];
6
7       for (int i = 0; i < array.Length; i++)
8       {
9         array[i] = i;
10      }
11
12      for (int i = 0; i < array.Length; i++)
13      {
14        System.Console.WriteLine(array[i]);
15      }
16    }
17  }
```

Código C# 6.8: SequenciaCrescente.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao20.zip

21 Compile e execute a classe **SequenciaCrescente**.

```
C:\Users\K19\rafael\arrays> csc SequenciaCrescente.cs

C:\Users\K19\rafael\arrays> SequenciaCrescente.exe
0
1
2
3
4
5
6
7
8
9
```

Terminal 6.14: Compilando e executando a classe `SequenciaCrescente`

22 Na pasta **arrays**, crie um arquivo chamado **SequenciaDecrescente.cs**. Implemente um programa em Java que armazene 10 números inteiros em um array. Preencha todas as posições do array com valores sequenciais descrescentes. Ao final, exiba no terminal esse valores.

```
1  class SequenciaDecrescente
2  {
3    static void Main()
4    {
5      int[] array = new int[10];
6
7      for (int i = 0; i < array.Length; i++)
8      {
9        array[i] = array.Length - 1 - i;
10     }
11
12     for (int i = 0; i < array.Length; i++)
13     {
14       System.Console.WriteLine(array[i]);
15     }
16   }
17 }
```

Código C# 6.9: SequenciaDecrescente.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao22.zip

23 Compile e execute a classe **SequenciaDecrescente**.

```
C:\Users\K19\rafael\arrays> csc SequenciaDecrescente.cs

C:\Users\K19\rafael\arrays> SequenciaDecrescente.exe
9
8
7
6
5
4
3
2
1
0
```

Terminal 6.15: Compilando e executando a classe `SequenciaDecrescente`

24 Na pasta **arrays**, crie um arquivo chamado **SequenciaImpar.cs**. Implemente um programa em Java que armazene 10 números inteiros ímpares em um array. Ao final, exiba no terminal esse valores.

```
class SequenciaImpar
{
  static void Main()
  {
    int[] array = new int[10];

    for (int i = 0; i < array.Length; i++)
    {
      array[i] = 2 * i + 1;
    }

    for (int i = 0; i < array.Length; i++)
    {
      System.Console.WriteLine(array[i]);
    }
  }
}
```

Código C# 6.10: SequenciaImpar.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao24.zip

25 Compile e execute a classe **SequenciaImpar**.

```
C:\Users\K19\rafael\arrays> csc SequenciaImpar.cs

C:\Users\K19\rafael\arrays> SequenciaImpar.exe
1
3
5
7
9
11
13
15
17
19
```

Terminal 6.16: Compilando e executando a classe `SequenciaImpar`

26 Na pasta **arrays**, crie um arquivo chamado **SequenciaAleatoria.cs**. Implemente um programa em Java que armazene 10 números inteiros aleatórios em um array. Ao final, exiba no terminal esse valores.

```
class SequenciaAleatoria
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    int[] array = new int[10];

    for (int i = 0; i < array.Length; i++)
    {
      array[i] = (int)(gerador.NextDouble() * 100);
    }

    for (int i = 0; i < array.Length; i++)
    {
      System.Console.WriteLine(array[i]);
    }
```

```
18    }
19 }
```

Código C# 6.11: SequenciaAleatoria.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao26.zip

27 Compile e execute a classe **SequenciaAleatoria**.

```
C:\Users\K19\rafael\arrays> csc SequenciaAleatoria.cs

C:\Users\K19\rafael\arrays> SequenciaAleatoria.exe
20
76
88
45
39
91
33
25
18
70
```

Terminal 6.17: Compilando e executando a classe `SequenciaAleatoria`

28 Na pasta **arrays**, crie um arquivo chamado **TabelaQualquer.cs**. Implemente um programa em Java que armazene números inteiros em um array de arrays. Ao final, exiba no terminal esse valores.

```
1  class TabelaQualquer
2  {
3    static void Main()
4    {
5      int[][] array = new int[3][];
6
7      for (int i = 0; i < array.Length; i++)
8      {
9        array[i] = new int[3];
10     }
11
12     array[0][0] = 19;
13     array[0][1] = 22;
14     array[0][2] = 31;
15     array[1][0] = 2;
16     array[1][1] = 51;
17     array[1][2] = 12;
18     array[2][0] = 41;
19     array[2][1] = 11;
20     array[2][2] = 3;
21
22
23     for (int i = 0; i < array.Length; i++)
24     {
25       for (int j = 0; j < array[i].Length; j++)
26       {
27         System.Console.WriteLine(array[i][j]);
28       }
29     }
30   }
31 }
```

Código C# 6.12: TabelaQualquer.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao28.zip

29 Compile e execute a classe **TabelaQualquer**.

```
C:\Users\K19\rafael\arrays> csc TabelaQualquer.cs

C:\Users\K19\rafael\arrays> TabelaQualquer.exe
19
22
31
2
51
12
41
11
3
```

Terminal 6.18: Compilando e executando a classe `TabelaQualquer`

30 Na pasta **arrays**, crie um arquivo chamado **TabelaAleatoria.cs**. Implemente um programa em Java que armazene números inteiros em um array de arrays. Ao final, exiba no terminal esse valores.

```
1  class TabelaAleatoria
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      int[][] array = new int[3][];
8
9      for (int i = 0; i < array.Length; i++)
10     {
11       array[i] = new int[3];
12       for (int j = 0; j < array[i].Length; j++)
13       {
14         array[i][j] = (int)(gerador.NextDouble() * 100);
15       }
16     }
17
18     for (int i = 0; i < array.Length; i++)
19     {
20       for (int j = 0; j < array[i].Length; j++)
21       {
22         System.Console.WriteLine(array[i][j]);
23       }
24     }
25   }
26 }
```

Código C# 6.13: TabelaAleatoria.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao30.zip

31 Compile e execute a classe **TabelaAleatoria**.

```
C:\Users\K19\rafael\arrays> csc TabelaAleatoria.cs

C:\Users\K19\rafael\arrays> TabelaAleatoria.exe
20
76
88
45
39
91
33
25
18
70
```

Terminal 6.19: Compilando e executando a classe `TabelaAleatoria`

32 Na pasta **arrays**, crie um arquivo chamado **Tabuada.cs**. Implemente um programa em Java que armazene números inteiros em um array de arrays. Ao final, exiba no terminal esse valores.

```
1  class Tabuada
2  {
3    static void Main()
4    {
5      int[][] tabuada = new int[10][];
6
7      for (int i = 0; i < tabuada.Length; i++)
8      {
9        tabuada[i] = new int[10];
10       for (int j = 0; j < tabuada[i].Length; j++)
11       {
12         tabuada[i][j] = (i + 1) * (j + 1);
13       }
14     }
15
16     for (int i = 0; i < tabuada.Length; i++)
17     {
18       for (int j = 0; j < tabuada[i].Length; j++)
19       {
20         System.Console.WriteLine((i + 1) + " x " + (j + 1) + " = " + tabuada[i][j]);
21       }
22     }
23   }
24 }
```

Código C# 6.14: Tabuada.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-fixacao32.zip

33 Compile e execute a classe **Tabuada**.

```
C:\Users\K19\rafael\arrays> csc Tabuada.cs

C:\Users\K19\rafael\arrays> Tabuada.exe
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
...
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

Terminal 6.20: Compilando e executando a classe `Tabuada`

Exercícios Complementares

1 Considere um programa de computador que corrige provas de múltipla escolha. Esse programa deve armazenar em um array o gabarito de uma prova. Implemente um programa em Java que gere aleatoriamente o gabarito de uma prova com 10 questões de múltipla escolha. Considere que cada questão possui três alternativas numeradas de 1 a 3. Complete o código a seguir.

```
class GeraradorDeGabarito {
  public static void main(String[] args) {

  }
}
```

Código Java 6.21: GeraradorDeGabarito.java

2 Considere um programa de computador que corrige provas de múltipla escolha. Esse programa deve armazenar as respostas dos alunos em uma tabela (um array de arrays). Na primeira linha dessa tabela, devem ser armazenadas as respostas do primeiro aluno. Na segunda linha, as notas do segundo aluno. E assim por diante. Implemente um programa em Java que preencha uma tabela com respostas aleatórias. Para implementar esse programa, considere que a prova tenha 10 questões de múltipla escolha com 3 alternativas cada numeradas de 1 a 3. Além disso, considere que que 5 alunos realizaram essa prova. Complete o código a seguir.

```
class GeraradorDeRespostasAleatorias {
  public static void main(String[] args) {

  }
}
```

Código Java 6.23: GeraradorDeRespostasAleatorias.java

3 Considere um programa de computador que corrige provas de múltipla escolha. Esse programa deve armazenar em um array o gabarito da prova. As respostas dos alunos devem ser armazenadas em uma tabela (um array de arrays). Na primeira linha dessa tabela, devem ser armazenadas as respostas do primeiro aluno. Na segunda linha, as notas do segundo aluno. E assim por diante. O programa deve exibir a quantidade de acertos de cada aluno. Implemente um programa em Java para realizar a correção dessas provas. Complete o código a seguir.

```
class CorretorDeProva {
  public static void main(String[] args) {
    int[] gabarito = new int[10];

    for(int i = 0; i < gabarito.length; i++) {
      gabarito[i] = (int)(Math.random() * 3 + 1);
      System.out.print(gabarito[i] + " ");
    }
    System.out.println("gabarito");

    int[][] respostas = new int[5][10];

    for(int i = 0; i < respostas.length; i++) {
      for(int j = 0; j < respostas[i].length; j++) {
        respostas[i][j] = (int)(Math.random() * 3 + 1);
        System.out.print(respostas[i][j] + " ");
      }
      System.out.println("aluno " + (i + 1));
    }

  }
}
```

Código Java 6.25: CorretorDeProva.java

4 Considere um programa de computador que controla as vagas de um estacionamento. Esse programa deve armazenar em uma tabela (array de arrays) a situação das vagas (ocupada ou livre) por andar. Implemente um programa em Java que defina aleatoriamente a situação das vagas de um estacionamento de quatro andares numerados de 1 a 4. Considere que a capacidade de cada andar é de 10 vagas. Complete o código a seguir.

```
1 class ControleDeVagas {
2   public static void main(String[] args) {
3
4   }
5 }
```

Código Java 6.27: ControleDeVagas.java

5 Considere um programa de computador que controla as vagas de um estacionamento. Esse programa deve armazenar em uma tabela (array de arrays) a situação das vagas (ocupada ou livre). Implemente um programa em Java que exiba no terminal a quantidade de vagas livres por andar. Complete o código a seguir.

```
1  class Estacionamento {
2    public static void main(String[] args) {
3      boolean[][] vagas = new boolean[4][10];
4      for(int i = 0; i < vagas.length; i++) {
5        for(int j = 0; j < vagas[i].length; j++) {
6          vagas[i][j] = Math.random() < 0.5;
7          System.out.print(vagas[i][j] ?  "- " : "X ");
8        }
9        System.out.println("andar " + (i + 1));
10     }
11
12
13   }
14 }
```

Código Java 6.29: Estacionamento.java

6 Considere um programa de computador que corrige provas de múltipla escolha. Esse programa deve armazenar em um array o gabarito de uma prova. Implemente um programa em Java que gere aleatoriamente o gabarito de uma prova com 10 questões de múltipla escolha. Considere que cada questão possui três alternativas numeradas de 1 a 3. Complete o código a seguir.

```
1 class GeraradorDeGabarito
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 6.15: GeraradorDeGabarito.cs

7 Considere um programa de computador que corrige provas de múltipla escolha. Esse programa deve armazenar as respostas dos alunos em uma tabela (um array de arrays). Na primeira linha dessa tabela, devem ser armazenadas as respostas do primeiro aluno. Na segunda linha, as notas do segundo aluno. E assim por diante. Implemente um programa em Java que preencha uma tabela

com respostas aleatórias. Para implementar esse programa, considere que a prova tenha 10 questões de múltipla escolha com 3 alternativas cada numeradas de 1 a 3. Além disso, considere que que 5 alunos realizaram essa prova. Complete o código a seguir.

```
1 class GeraradorDeRespostasAleatorias
2 {
3   static void Main()
4   {
5
6   }
7 }
```

Código C# 6.17: GeraradorDeRespostasAleatorias.cs

8 Considere um programa de computador que corrige provas de múltipla escolha. Esse programa deve armazenar em um array o gabarito da prova. As respostas dos alunos devem ser armazenadas em uma tabela (um array de arrays). Na primeira linha dessa tabela, devem ser armazenadas as respostas do primeiro aluno. Na segunda linha, as notas do segundo aluno. E assim por diante. O programa deve exibir a quantidade de acertos de cada aluno. Implemente um programa em Java para realizar a correção dessas provas. Complete o código a seguir.

```
1  class CorretorDeProva
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      int[] gabarito = new int[10];
8
9      for(int i = 0; i < gabarito.Length; i++)
10     {
11       gabarito[i] = (int)(gerador.NextDouble() * 3 + 1);
12       System.Console.Write(gabarito[i] + " ");
13     }
14     System.Console.WriteLine("gabarito");
15
16     int[][] respostas = new int[5][];
17
18     for(int i = 0; i < respostas.Length; i++)
19     {
20       respostas[i] = new int[10];
21       for(int j = 0; j < respostas[i].Length; j++)
22       {
23         respostas[i][j] = (int)(gerador.NextDouble() * 3 + 1);
24         System.Console.Write(respostas[i][j] + " ");
25       }
26       System.Console.WriteLine("aluno " + (i + 1));
27     }
28
29
30   }
31 }
```

Código C# 6.19: CorretorDeProva.cs

9 Considere um programa de computador que controla as vagas de um estacionamento. Esse programa deve armazenar em uma tabela (array de arrays) a situação das vagas (ocupada ou livre) por andar. Implemente um programa em Java que defina aleatoriamente a situação das vagas de um estacionamento de quatro andares numerados de 1 a 4. Considere que a capacidade de cada andar

é de 10 vagas. Complete o código a seguir.

```
class ControleDeVagas
{
  static void Main()
  {

  }
}
```

Código C# 6.21: ControleDeVagas.cs

10 Considere um programa de computador que controla as vagas de um estacionamento. Esse programa deve armazenar em uma tabela (array de arrays) a situação das vagas (ocupada ou livre). Implemente um programa em Java que exiba no terminal a quantidade de vagas livres por andar. Complete o código a seguir.

```
class Estacionamento
{
  static void Main()
  {
    System.Random gerador = new System.Random();
    bool[][] vagas = new bool[4][];
    for(int i = 0; i < vagas.Length; i++)
    {
      vagas[i] = new bool[10];
      for(int j = 0; j < vagas[i].Length; j++)
      {
        vagas[i][j] = gerador.NextDouble() < 0.5;
        System.Console.Write(vagas[i][j] ?  "- " : "X ");
      }
      System.Console.WriteLine("andar " + (i + 1));
    }

  }
}
```

Código C# 6.23: Estacionamento.cs

Resumo do Capítulo

1 Os arrays são **estruturas de dados** simples que permitem o armazenamento **sequencial** de dados.

2 As **posições** de um array são **numeradas** sequencialmente iniciando com **0**.

3 Em Java, a capacidade de um array pode ser obtida através do atributo **length**.

4 Em C#, a capacidade de um array pode ser obtida através da propriedade **Length**.

5 Na tentativa de acesso à uma **posição inexistente** de um array, um **erro de execução** é gerado tanto no Java quanto no C#.

Prova

1 Qual alternativa está correta?

a) As posições de um array são numeradas iniciando no 0.

b) As posições de um array são numeradas iniciando no 1.

c) Um array pode ter no máximo 100 posições.

d) Os arrays armazenam apenas números inteiros.

e) As posições de um array são acessadas com **chaves**.

2 Considere o seguinte código.

```
1 int[] array = new int[10];
2 array[10] = 10;
```

Qual alternativa está correta?

a) Nesse código, há um erro de compilação.

b) Nesse código, há um erro de execução.

c) Não há nada de errado nesse código.

d) O valor 10 será exibido no terminal.

e) O valor 0 será exibido no terminal.

3 Em Java, como a capacidade de um array é recuperada?

a) Através da propriedade **Length**.

b) Através da propriedade **Size**.

c) Através do atributo **capacity**.

d) Através do atributo **size**.

e) Através do atributo **length**.

4 Em C#, como a capacidade de um array é recuperada?

a) Através da propriedade **Length**.

b) Através da propriedade **Size**.

c) Através do atributo **capacity**.

d) Através do atributo **size**.

e) Através do atributo **length**.

5 Em Java ou C#, qual é a forma correta de acessar a quinta posição de um array?

a) array[5].

b) array{5}.

c) array(5).

d) array[4].

e) array{4}.

6 Considere o seguinte código.

```
1 int[] array = new int[10];
2
3 for(int i = 0; i < 100; i++) {
4   array[i] = i;
5 }
```

Qual alternativa está correta?

a) Na compilação, um erro ocorrerá.

b) Na execução, as 100 posições do array serão preenchidas.

c) Na compilação, as 100 posições do array serão preenchidas.

d) Na execução, todos os valores armazenados no array serão exibidos no terminal.

e) Na execução, um erro ocorrerá.

Minha Pontuação [ ] Pontuação Mínima: [4] Pontuação Máxima: [6]

CAPÍTULO 7

FUNÇÕES OU MÉTODOS

Considere uma empresa que precisa gerar diversos tipos de documentos como recibos, atestados ou relatórios. Os dados da empresa devem aparecer no cabeçalho dos documentos. Para automatizar a criação desses documentos, podemos implementar um programa de computador.

```
System.out.println("------------ K19 Treinamentos ------------");
System.out.println("----------- contato@k19.com.br -----------");
```

O trecho de código acima exibe no terminal o cabeçalho que deve aparecer nos documentos. Toda vez que um documento for exibido no terminal, as linhas acima serão adicionadas no código. Dessa forma, esse trecho de código se repetirá muitas vezes.

Agora, considere uma mudança simples no cabeçalho dos documentos, o telefone da empresa deve aparecer depois do email. Essa pequena alteração implicará em modificações em muitos pontos do código fonte. Na verdade, qualquer alteração no cabeçalho implicará em modificações em muitos lugares do código fonte. Para facilitar eventuais modificações no cabeçalho, podemos utilizar o conceito de **função** ou **método**.

Importante
Linguagens Orientadas a Objetos como Java ou C# utilizam o termo **método** e não o termo **função**. Contudo, nesse momento, não discutiremos as diferenças conceituais entre métodos e funções pois essa discussão está totalmente fora do escopo desse treinamento.

Então, definiremos um método para exibir o cabeçalho dos documentos no terminal e reaproveitá-lo toda vez que for necessário.

```
static void exibeCabecalho() {
  System.out.println("------------ K19 Treinamentos ------------");
  System.out.println("----------- contato@k19.com.br -----------");
}
```

Quando necessário, um método pode ser chamado através do seu nome. Observe, no código abaixo, que o método **exibeCabecalho** foi chamado duas vezes.

```
class Programa {
  public static void main(String[] args) {
    // chamando a função exibeCabecalho
    exibeCabecalho();
    System.out.println("Recibo: R$ 545,00");

    System.out.println();

    // chamando a função exibeCabecalho
    exibeCabecalho();
    System.out.println("Atestado de Matrícula: Jonas Keizo Hirata");
  }
```

```
13
14    static void exibeCabecalho() {
15      System.out.println("------------ K19 Treinamentos ------------");
16      System.out.println("----------- contato@k19.com.br -----------");
17    }
18  }
```

```
K19$ java Programa
------------ K19 Treinamentos ------------
----------- contato@k19.com.br -----------
Recibo: R$ 545,00

------------ K19 Treinamentos ------------
----------- contato@k19.com.br -----------
Atestado de Matrícula: Jonas Keizo Hirata
```

Agora, acrescentar o telefone da empresa no cabeçalho dos documentos é muito fácil. Basta alterar o código da função **exibeCabecalho**.

```
1  static void exibeCabecalho() {
2    System.out.println("------------ K19 Treinamentos ------------");
3    System.out.println("----------- contato@k19.com.br -----------");
4    System.out.println("-------------- 11 2387-3791 --------------");
5  }
```

Parâmetros

Considere um programa de computador que realiza operações financeiras como o cálculo de juros simples por exemplo. Para evitar repetição de código, podemos definir um método para realizar esse cálculo e reutilizá-lo todo vez que for necessário.

```
1  static void calculaJurosSimples() {
2    double juros = 10000 * 0.015 * 12;
3  }
```

Observe que o método acima considera um capital fixo de R$ 10.000,00, uma taxa de juros fixa de 1,5% e um período fixo de 12 meses. De fato, esse método não é muito útil porque toda vez que ele for chamado, ele realizará o cálculo com esses valores fixos.

Para tornar o método **calculaJurosSimples** mais útil, devemos **parametrizá-lo**. Um parâmetro é basicamente um valor que um método recebe antes de ser executado.

```
1  static void calculaJurosSimples(double capital, double taxa, int periodo) {
2    double juros = capital * taxa * periodo;
3  }
```

No código acima, três parâmetros foram definidos para o método **calculaJurosSimples**. O primeiro parâmetro é do tipo **double** e será armazenado na variável **capital**. O segundo é do tipo **double** e será armazenado na variável **taxa**. O terceiro é do tipo **int** e será armazenado na variável **periodo**.

Agora, nas chamadas do método **calculaJurosSimples**, devemos passar os três parâmetros necessários para o cálculo do juros simples. No exemplo a seguir, o método **main** chama o método **calculaJurosSimples** duas vezes. Na primeira chamada, os valores passados como parâmetro são: 10000, 0.015 e 12. Na segunda chamada, os valores passados como parâmetro são: 25400, 0.02 e 30.

```
class Programa {
  public static void main(String[] args) {
    calculaJurosSimples(10000, 0.015, 12);

    calculaJurosSimples(25400, 0.02, 30);

  }

  static void calculaJurosSimples(double capital, double taxa, int periodo) {
    double juros = capital * taxa * periodo;
  }
}
```

Resposta

O valor calculado dentro do método **calculaJurosSimples** é armazenado em uma variável local. Essa variável não pode ser acessada dentro do método **main**. Em outras palavras, o método **main** não tem acesso ao juros que foi calculado dentro do método **calculaJurosSimples**.

Todo método pode, ao final do seu processamento, devolver uma resposta para que o chamou. Nas linguagens Java e C#, a instrução **return** indica o valor de resposta de um método.

```
static double calculaJurosSimples(double capital, double taxa, int periodo) {
  double juros = capital * taxa * periodo;
  return juros;
}
```

Observe as duas modificações realizadas no método **calculaJurosSimples**. A primeira alteração é a retirada da palavra reservada **void** e a inserção da palavra reservada **double** em seu lugar. A palavra **void** indicava que o método não devolvia nenhuma resposta ao final do seu processamento. A palavra **double** indica que o método devolverá um valor do tipo **double** ao final do seu processamento. A segunda modificação é a utilização do comando **return** para devolver como resposta o valor do juros que é um valor do tipo **double**.

Agora, a resposta pode ser recuperada no método **main** é armazenada em uma variável.

```
class Programa {
  public static void main(String[] args) {
    double resposta1 = calculaJurosSimples(10000, 0.015, 12);

    double resposta2 = calculaJurosSimples(25400, 0.02, 30);

    System.out.println("Juros: " + resposta1);
    System.out.println("Juros: " + resposta2);
  }

  static double calculaJurosSimples(double capital, double taxa, int periodo) {
    double juros = capital * taxa * periodo;
    return juros;
  }
}
```

Um método pode devolver outros tipos de valores. Para isso, basta modificar a marcação de retorno definindo o tipo de valor que o método devolverá. Veja alguns exemplos.

```
static int metodo() {
```

```
2   // corpo de um método que devolve int
3 }
```

```
1 static char metodo() {
2   // corpo de um método que devolve char
3 }
```

```
1 static float metodo() {
2   // corpo de um método que devolve float
3 }
```

Exercícios de Fixação Com Java

1 Abra um terminal; Entre na pasta dos seus exercícios e crie uma pasta chamada **funcoes-ou-metodos** para os arquivos desenvolvidos nesse capítulo.

```
K19/rafael$ mkdir funcoes-ou-metodos

K19/rafael$ cd funcoes-ou-metodos

K19/rafael/funcoes-ou-metodos$
```

Terminal 7.2: Criando a pasta `funcoes-ou-metodos` *no Linux*

```
C:\Users\K19\rafael> md funcoes-ou-metodos

C:\Users\K19\rafael> cd funcoes-ou-metodos

C:\Users\K19\rafael\funcoes-ou-metodos>
```

Terminal 7.3: Criando a pasta `funcoes-ou-metodos` *no Windows*

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao1.zip

2 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **ConsumoDeCombustivel.java**. Implemente um programa em Java definindo um método que realize o cálculo do consumo de combustível de um veículo.

```
1  class ConsumoDeCombustivel {
2    public static void main(String[] args) {
3      double reposta1 = calculaConsumoLitroKilometro(131.679, 13.5);
4      double reposta2 = calculaConsumoLitroKilometro(251.856, 21.6);
5
6      System.out.println("Consumo: " + reposta1);
7      System.out.println("Consumo: " + reposta2);
8    }
9
10   static double calculaConsumoLitroKilometro(double distancia, double combustivel) {
11     double consumo = distancia/combustivel;
12     return consumo;
13   }
14 }
```

Código Java 7.13: ConsumoDeCombustivel.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao2.zip

3 Compile e execute a classe **ConsumoDeCombustivel**.

```
K19/rafael/funcoes-ou-metodos$ javac ConsumoDeCombustivel.java

K19/rafael/funcoes-ou-metodos$ java ConsumoDeCombustivel
Consumo: 9.754
Consumo: 11.659999999999998
```

Terminal 7.4: Compilando e executando a classe `ConsumoDeCombustivel`

4 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **JurosComposto.java**. Implemente um programa em Java definindo um método que realize o cálculo do montante obtido com a aplicação de juros compostos.

```
1  class JurosComposto {
2    public static void main(String[] args) {
3      double reposta1 = calculaJurosComposto(10000, 0.1, 6);
4      double reposta2 = calculaJurosComposto(20000, 0.05, 6);
5
6      System.out.println("Montante: " + reposta1);
7      System.out.println("Montante: " + reposta2);
8    }
9
10   static double calculaJurosComposto(double capital, double taxa, int periodo) {
11     double montante = capital * Math.pow(1 + taxa, periodo);
12     return montante;
13   }
14 }
```

Código Java 7.14: JurosComposto.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao4.zip

5 Compile e execute a classe **JurosComposto**.

```
K19/rafael/funcoes-ou-metodos$ javac JurosComposto.java

K19/rafael/funcoes-ou-metodos$ java JurosComposto
Montante: 17715.610000000008
Montante: 26801.91281250001
```

Terminal 7.5: Compilando e executando a classe `JurosComposto`

6 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **IRPF.java**. Implemente um programa em Java definindo um método que realize o cálculo do imposto de renda pessoa física.

```
1  class IRPF {
2    public static void main(String[] args) {
3      double reposta1 = calculaIRPF(1350.57);
4      double reposta2 = calculaIRPF(2150.37);
5      double reposta3 = calculaIRPF(3378.98);
6      double reposta4 = calculaIRPF(3956.12);
7      double reposta5 = calculaIRPF(6200.15);
8
9      System.out.println("IRPF 1: " + reposta1);
10     System.out.println("IRPF 2: " + reposta2);
11     System.out.println("IRPF 3: " + reposta3);
12     System.out.println("IRPF 4: " + reposta4);
13     System.out.println("IRPF 5: " + reposta5);
14   }
15
```

```
16    static double calculaIRPF(double renda) {
17      if(renda <= 1710.18) {
18        return 0;
19      } else if(renda <= 2563.91) {
20        return renda * 0.075 - 128.31;
21      } else if(renda <= 3418.59) {
22        return renda * 0.15 - 320.60;
23      } else if(renda <= 4271.59) {
24        return renda * 0.225 - 577.00;
25      } else {
26        return renda * 0.275 - 790.58;
27      }
28    }
29  }
```

Código Java 7.15: IRPFjava

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao6.zip

7 Compile e execute a classe **IRPF**.

```
K19/rafael/funcoes-ou-metodos$ javac IRPF.java

K19/rafael/funcoes-ou-metodos$ java IRPF
IRPF 1: 0.0
IRPF 2: 32.967749999999995
IRPF 3: 186.24699999999996
IRPF 4: 313.12699999999995
IRPF 5: 914.46125
```

Terminal 7.6: Compilando e executando a classe `IRPF`

8 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **IMC.java**. Implemente um programa em Java definindo um método que realize o cálculo do índice de massa corporal e outro que determina a situação da pessoa a partir desse índice.

```
1   class IMC {
2     public static void main(String[] args) {
3       double amandaIMC = calculaIMC(52.6, 1.61);
4       double joyceIMC = calculaIMC(54.1, 1.59);
5
6       String amandaSituacao = calculaResultadoIMC(amandaIMC);
7       String joyceSituacao = calculaResultadoIMC(joyceIMC);
8
9       System.out.println("Amanda IMC: " + amandaIMC + " - " + amandaSituacao);
10      System.out.println("Joyce IMC: " + joyceIMC + " - " + joyceSituacao);
11    }
12
13    static double calculaIMC(double peso, double altura) {
14      return peso / (altura * altura);
15    }
16
17    static String calculaResultadoIMC(double imc) {
18      if(imc < 17) {
19        return "Muito abaixo do peso";
20      } else if(imc < 18.5) {
21        return "Abaixo do peso";
22      } else if(imc < 25) {
23        return "Peso normal";
24      } else if(imc < 30) {
25        return "Acima do peso";
26      } else if(imc < 35) {
27        return "Obsesidade I";
28      } else if(imc < 40) {
```

```
29        return "Obsesidade II - severa";
30      } else {
31        return "Obsesidade III - mórbida";
32      }
33    }
34  }
```

Código Java 7.16: IMC.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao8.zip

9 Compile e execute a classe **IMC**.

```
K19/rafael/funcoes-ou-metodos$ javac IMC.java

K19/rafael/funcoes-ou-metodos$ java IMC
Amanda IMC: 20.292426989699468 - Peso normal
Joyce IMC: 21.399469957675723 - Peso normal
```

Terminal 7.7: Compilando e executando a classe `IMC`

10 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **Arrays.java**. Implemente um programa em Java definindo um método que preencha um array com números inteiros aleatórios no intervalo de 0 a 99 e outro que exiba no terminal os valores armazenados em um array.

```
1  class Arrays {
2    public static void main(String[] args) {
3      int[] array1 = new int[5];
4      int[] array2 = new int[10];
5
6      preencheArray(array1);
7      preencheArray(array2);
8
9      exibeArray(array1);
10     exibeArray(array2);
11   }
12
13   static void preencheArray(int[] array) {
14     for(int i = 0; i < array.length; i++) {
15       array[i] = (int)(Math.random() * 100);
16     }
17   }
18
19   static void exibeArray(int[] array) {
20     System.out.println("Array: ");
21     for(int i = 0; i < array.length; i++) {
22       System.out.println("array[" + i + "] = " + array[i]);
23     }
24     System.out.println("-------------------------------------");
25   }
26 }
```

Código Java 7.17: Arrays.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao10.zip

11 Compile e execute a classe **Arrays**.

```
K19/rafael/funcoes-ou-metodos$ javac Arrays.java

K19/rafael/funcoes-ou-metodos$ java Arrays
```

```
Array:
array[0] = 8
array[1] = 74
array[2] = 26
array[3] = 30
array[4] = 80
-----------------------------------
Array:
array[0] = 92
array[1] = 63
array[2] = 79
array[3] = 88
array[4] = 19
array[5] = 44
array[6] = 4
array[7] = 36
array[8] = 85
array[9] = 23
-----------------------------------
```

Terminal 7.8: Compilando e executando a classe `Arrays`

12 Acrescente um método na classe **Arrays** para contar a quantidade números pares de um array.

```
1  class Arrays {
2    public static void main(String[] args) {
3      int[] array1 = new int[5];
4      int[] array2 = new int[10];
5
6      preencheArray(array1);
7      preencheArray(array2);
8
9      exibeArray(array1);
10     exibeArray(array2);
11
12     int pares1 = contaPar(array1);
13     int pares2 = contaPar(array2);
14
15     System.out.println("Quantidade de pares do primeiro array: " + pares1);
16     System.out.println("Quantidade de pares do segundo array: " + pares2);
17   }
18
19   static int contaPar(int[] array) {
20     int pares = 0;
21     for(int i = 0; i < array.length; i++) {
22       if(array[i] % 2 == 0) {
23         pares++;
24       }
25     }
26     return pares;
27   }
28
29   static void preencheArray(int[] array) {
30     for(int i = 0; i < array.length; i++) {
31       array[i] = (int)(Math.random() * 100);
32     }
33   }
34
35   static void exibeArray(int[] array) {
36     System.out.println("Array: ");
37     for(int i = 0; i < array.length; i++) {
38       System.out.println("array[" + i + "] = " + array[i]);
39     }
40     System.out.println("-----------------------------------");
41   }
42 }
```

Código Java 7.18: Arrays.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao12.zip

13 Compile e execute a classe **Arrays**.

```
K19/rafael/funcoes-ou-metodos$ javac Arrays.java

K19/rafael/funcoes-ou-metodos$ java Arrays
Array:
array[0] = 95
array[1] = 16
array[2] = 65
array[3] = 2
array[4] = 20
-----------------------------------
Array:
array[0] = 9
array[1] = 13
array[2] = 32
array[3] = 16
array[4] = 54
array[5] = 56
array[6] = 53
array[7] = 66
array[8] = 13
array[9] = 8
-----------------------------------
Quantidade de pares do primeiro array: 3
Quantidade de pares do segundo array: 6
```

Terminal 7.9: Compilando e executando a classe `Arrays`

Erro: Parâmetros incompatíveis

Um erro de **execução** comum em Java ou C# ocorre quando um método é chamado com parâmetros incompatíveis.

```
1  class Programa {
2    public static void main(String[] args) {
3      metodo();
4      metodo(10.1, 10.1, "k19");
5      metodo("10", "10.1", "k19");
6    }
7
8    static void metodo(int a, double b, String c) {
9      return a + b + c;
10   }
11 }
```

Código Java 7.19: Programa.java

A mensagem de erro de **compilação** seria semelhante a apresenta abaixo.

```
Programa.java:3: error: method metodo in class Programa cannot be applied to given types;
    metodo();
    ^
  required: int,double,String
  found: no arguments
  reason: actual and formal argument lists differ in length
Programa.java:4: error: method metodo in class Programa cannot be applied to given types;
    metodo(10.1, 10.1, "k19");
    ^
  required: int,double,String
  found: double,double,String
  reason: actual argument double cannot be converted to int by method invocation conversion
Programa.java:5: error: method metodo in class Programa cannot be applied to given types;
    metodo("10", "10.1", "k19");
    ^
```

```
  required: int,double,String
  found: String,String,String
  reason: actual argument String cannot be converted to int by method invocation conversion
Programa.java:9: error: cannot return a value from method whose result type is void
    return a + b + c;
               ^
4 errors
```

Terminal 7.10: Erro de execução

Agora, veja um exemplo de programa em C# com esse problema.

```
1  class Programa
2  {
3    static void Main()
4    {
5      metodo();
6      metodo(10.1, 10.1, "k19");
7      metodo("10.1", "10.1", "k19");
8    }
9
10   static void metodo(int a, double b, string c) {
11     return a + b + c;
12   }
13 }
```

Código C# 7.1: Programa.cs

A mensagem de erro de **execução** seria semelhante a apresenta abaixo.

```
Programa.cs(5,3): error CS1501: No overload for method 'metodo' takes 0
        arguments
Programa.cs(10,14): (Location of symbol related to previous error)
Programa.cs(6,3): error CS1502: The best overloaded method match for
        'Programa.metodo(int, double, string)' has some invalid arguments
Programa.cs(6,10): error CS1503: Argument 1: cannot convert from 'double' to
        'int'
Programa.cs(7,3): error CS1502: The best overloaded method match for
        'Programa.metodo(int, double, string)' has some invalid arguments
Programa.cs(7,10): error CS1503: Argument 1: cannot convert from 'string' to
        'int'
Programa.cs(7,18): error CS1503: Argument 2: cannot convert from 'string' to
        'double'
Programa.cs(11,3): error CS0127: Since 'Programa.metodo(int, double, string)'
        returns void, a return keyword must not be followed by an object
        expression
```

Terminal 7.11: Erro de execução

Erro: Resposta incompatível

Um erro de **execução** comum em Java ou C# ocorre quando armazenamos a resposta de um método em variáveis de tipos incompatíveis.

```
1  class Programa {
2    public static void main(String[] args) {
3      int a = metodo();
4      double d = metodo();
5      boolean a = metodo();
6    }
7
8    static string metodo() {
9      return "k19";
10   }
11 }
```

Código Java 7.20: Programa.java

A mensagem de erro de **compilação** seria semelhante a apresenta abaixo.

```
Programa.java:8: error: cannot find symbol
  static string metodo() {
         ^
  symbol:   class string
  location: class Programa
Programa.java:5: error: variable a is already defined in method main(String[])
    boolean a = metodo();
            ^
2 errors
```

Terminal 7.12: Erro de execução

Agora, veja um exemplo de programa em C# com esse problema.

```
1  class Programa
2  {
3    static void Main()
4    {
5      int a = metodo();
6      double d = metodo();
7      bool b = metodo();
8    }
9
10   static string metodo() {
11     return "k19";
12   }
13 }
```

Código C# 7.2: Programa.cs

A mensagem de erro de **execução** seria semelhante a apresenta abaixo.

```
Programa.cs(5,11): error CS0029: Cannot implicitly convert type 'string' to
        'int'
Programa.cs(6,14): error CS0029: Cannot implicitly convert type 'string' to
        'double'
Programa.cs(7,12): error CS0029: Cannot implicitly convert type 'string' to
        'bool'
```

Terminal 7.13: Erro de execução

Exercícios de Fixação Com C#

14 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **ConsumoDeCombustivel.cs**. Implemente um programa em Java definindo um método que realize o cálculo do consumo de combustível de um veículo.

```
1  class ConsumoDeCombustivel
2  {
3    static void Main()
4    {
5      double reposta1 = calculaConsumoLitroKilometro(131.679, 13.5);
6      double reposta2 = calculaConsumoLitroKilometro(251.856, 21.6);
7
```

```
8       System.Console.WriteLine("Consumo: " + reposta1);
9       System.Console.WriteLine("Consumo: " + reposta2);
10    }
11
12    static double calculaConsumoLitroKilometro(double distancia, double combustivel)
13    {
14      double consumo = distancia/combustivel;
15      return consumo;
16    }
17  }
```

Código C# 7.3: ConsumoDeCombustivel.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao14.zip

15 Compile e execute a classe **ConsumoDeCombustivel**.

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc ConsumoDeCombustivel.cs

C:\Users\K19\rafael\funcoes-ou-metodos> ConsumoDeCombustivel.exe
Consumo: 9.754
Consumo: 11.66
```

Terminal 7.14: Compilando e executando a classe `ConsumoDeCombustivel`

16 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **JurosComposto.cs**. Implemente um programa em Java definindo um método que realize o cálculo do montante obtido com a aplicação de juros compostos.

```
1   class JurosComposto
2   {
3     static void Main()
4     {
5       double reposta1 = calculaJurosComposto(10000, 0.1, 6);
6       double reposta2 = calculaJurosComposto(20000, 0.05, 6);
7
8       System.Console.WriteLine("Montante: " + reposta1);
9       System.Console.WriteLine("Montante: " + reposta2);
10    }
11
12    static double calculaJurosComposto(double capital, double taxa, int periodo)
13    {
14      double montante = capital * System.Math.Pow(1 + taxa, periodo);
15      return montante;
16    }
17  }
```

Código C# 7.4: JurosComposto.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao16.zip

17 Compile e execute a classe **JurosComposto**.

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc JurosComposto.cs

C:\Users\K19\rafael\funcoes-ou-metodos> JurosComposto.exe
Montante: 17715.61
Montante: 26801.9128125
```

Terminal 7.15: Compilando e executando a classe `JurosComposto`

18 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **IRPF.cs**. Implemente um programa em Java definindo um método que realize o cálculo do imposto de renda pessoa física.

```
class IRPF
{
  static void Main()
  {
    double reposta1 = calculaIRPF(1350.57);
    double reposta2 = calculaIRPF(2150.37);
    double reposta3 = calculaIRPF(3378.98);
    double reposta4 = calculaIRPF(3956.12);
    double reposta5 = calculaIRPF(6200.15);

    System.Console.WriteLine("IRPF 1: " + reposta1);
    System.Console.WriteLine("IRPF 2: " + reposta2);
    System.Console.WriteLine("IRPF 3: " + reposta3);
    System.Console.WriteLine("IRPF 4: " + reposta4);
    System.Console.WriteLine("IRPF 5: " + reposta5);
  }

  static double calculaIRPF(double renda)
  {
    if(renda <= 1710.18)
    {
      return 0;
    }
    else if(renda <= 2563.91)
    {
      return renda * 0.075 - 128.31;
    }
    else if(renda <= 3418.59)
    {
      return renda * 0.15 - 320.60;
    }
    else if(renda <= 4271.59)
    {
      return renda * 0.225 - 577.00;
    }
    else
    {
      return renda * 0.275 - 790.58;
    }
  }
}
```

Código C# 7.5: IRPF.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao18.zip

19 Compile e execute a classe **IRPF**.

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc IRPF.cs

C:\Users\K19\rafael\funcoes-ou-metodos> IRPF.exe
IRPF 1: 0
IRPF 2: 32.96775
IRPF 3: 186.247
IRPF 4: 313.127
IRPF 5: 914.46125
```

Terminal 7.16: Compilando e executando a classe IRPF

20 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **IMC.cs**. Implemente um programa em Java definindo um método que realize o cálculo do índice de massa corporal e outro que determina

a situação da pessoa a partir desse índice.

```
class IMC
{
  static void Main()
  {
    double amandaIMC = calculaIMC(52.6, 1.61);
    double joyceIMC = calculaIMC(54.1, 1.59);

    string amandaSituacao = calculaResultadoIMC(amandaIMC);
    string joyceSituacao = calculaResultadoIMC(joyceIMC);

    System.Console.WriteLine("Amanda IMC: " + amandaIMC + " - " + amandaSituacao);
    System.Console.WriteLine("Joyce IMC: " + joyceIMC + " - " + joyceSituacao);
  }

  static double calculaIMC(double peso, double altura)
  {
    return peso / (altura * altura);
  }

  static string calculaResultadoIMC(double imc)
  {
    if(imc < 17)
    {
      return "Muito abaixo do peso";
    }
    else if(imc < 18.5)
    {
      return "Abaixo do peso";
    }
    else if(imc < 25)
    {
      return "Peso normal";
    }
    else if(imc < 30)
    {
      return "Acima do peso";
    }
    else if(imc < 35)
    {
      return "Obsesidade I";
    }
    else if(imc < 40)
    {
      return "Obsesidade II - severa";
    }
    else
    {
      return "Obsesidade III - mórbida";
    }
  }
}
```

Código C# 7.6: IMC.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao20.zip

21 Compile e execute a classe **IMC**.

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc IMC.cs

C:\Users\K19\rafael\funcoes-ou-metodos> IMC.exe
Amanda IMC: 20.2924269896995 - Peso normal
Joyce IMC: 21.3994699576757 - Peso normal
```

Terminal 7.17: Compilando e executando a classe IMC

22 Na pasta **funcoes-ou-metodos**, crie um arquivo chamado **Arrays.cs**. Implemente um programa em Java definindo um método que preencha um array com números inteiros aleatórios no intervalo de 0 a 99 e outro que exiba no terminal os valores armazenados em um array.

```
class Arrays
{
  static void Main()
  {
    int[] array1 = new int[5];
    int[] array2 = new int[10];

    preencheArray(array1);
    preencheArray(array2);

    exibeArray(array1);
    exibeArray(array2);
  }

  static void preencheArray(int[] array)
  {
    System.Random gerador = new System.Random();
    for(int i = 0; i < array.Length; i++)
    {
      array[i] = (int)(gerador.NextDouble() * 100);
    }
  }

  static void exibeArray(int[] array)
  {
    System.Console.WriteLine("Array: ");
    for(int i = 0; i < array.Length; i++)
    {
      System.Console.WriteLine("array[" + i + "] = " + array[i]);
    }
    System.Console.WriteLine("-----------------------------------");
  }
}
```

Código C# 7.7: Arrays.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao22.zip

23 Compile e execute a classe **Arrays**.

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc Arrays.cs

C:\Users\K19\rafael\funcoes-ou-metodos> Arrays.exe
Array:
array[0] = 8
array[1] = 74
array[2] = 26
array[3] = 30
array[4] = 80
-----------------------------------
Array:
array[0] = 92
array[1] = 63
array[2] = 79
array[3] = 88
array[4] = 19
array[5] = 44
array[6] = 4
array[7] = 36
array[8] = 85
array[9] = 23
-----------------------------------
```

Terminal 7.18: Compilando e executando a classe `Arrays`

24 Acrescente um método na classe **Arrays** para contar a quantidade números pares de um array.

```
class Arrays
{
  static void Main()
  {
    int[] array1 = new int[5];
    int[] array2 = new int[10];

    preencheArray(array1);
    preencheArray(array2);

    exibeArray(array1);
    exibeArray(array2);

    int pares1 = contaPar(array1);
    int pares2 = contaPar(array2);

    System.Console.WriteLine("Quantidade de pares do primeiro array: " + pares1);
    System.Console.WriteLine("Quantidade de pares do segundo array: " + pares2);
  }

  static int contaPar(int[] array)
  {
    int pares = 0;
    for(int i = 0; i < array.Length; i++)
    {
      if(array[i] % 2 == 0)
      {
        pares++;
      }
    }
    return pares;
  }

  static void preencheArray(int[] array)
  {
    System.Random gerador = new System.Random();
    for(int i = 0; i < array.Length; i++)
    {
      array[i] = (int)(gerador.NextDouble() * 100);
    }
  }

  static void exibeArray(int[] array)
  {
    System.Console.WriteLine("Array: ");
    for(int i = 0; i < array.Length; i++)
    {
      System.Console.WriteLine("array[" + i + "] = " + array[i]);
    }
    System.Console.WriteLine("-----------------------------------");
  }
}
```

Código C# 7.8: ArraysPreencheContaPar.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-fixacao24.zip

25 Compile e execute a classe **Arrays**.

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc Arrays.cs

C:\Users\K19\rafael\funcoes-ou-metodos> Arrays.exe
Array:
array[0] = 95
array[1] = 16
array[2] = 65
```

```
array[3] = 2
array[4] = 20
-----------------------------------
Array:
array[0] = 9
array[1] = 13
array[2] = 32
array[3] = 16
array[4] = 54
array[5] = 56
array[6] = 53
array[7] = 66
array[8] = 13
array[9] = 8
-----------------------------------
Quantidade de pares do primeiro array: 3
Quantidade de pares do segundo array: 6
```

Terminal 7.19: Compilando e executando a classe `Arrays`

Exercícios Complementares

1 Implemente um método que determina se um ano é ou não é bissexto. As regras para determinar se um ano é bissexto são:

1. Anos múltiplos de 4 são bissextos.
2. Anos múltiplos de 100 não são bissextos.
3. Anos múltiplos de 400 são bissextos.
4. As últimas regras prevalecem sobre as primeiras.

Complete o código a seguir.

```
1  class AnoBissexto {
2    public static void main(String[] args) {
3      boolean b = bissexto(2000);
4      System.out.println("2000 " + b);
5
6      b = bissexto(2012);
7      System.out.println("2012 " + b);
8
9      b = bissexto(2025);
10     System.out.println("2025 " + b);
11
12     b = bissexto(2100);
13     System.out.println("2100 " + b);
14   }
15
16 }
```

Código Java 7.21: AnoBissexto.java

2 Implemente um método que verifica se uma determinada data é válida ou não. Lembre-se que Janeiro, Março, Maio, Julho, Agosto, Outubro e Dezembro possuem 31 dias; Abril, Junho, Setembro e Novembro possuem 30 dias; e Fevereiro possui 28 dias em anos não bissextos e 29 dias em anos bissextos. Complete o código a seguir.

```
class VerificaDatas {
  public static void main(String[] args) {
    boolean b = verificaData(29, 2, 2000);

    System.out.println("29/02/2000 - " + b);

    b = verificaData(29, 2, 2004);

    System.out.println("29/02/2004 - " + b);

    b = verificaData(31, 4, 2000);

    System.out.println("31/04/2000 - " + b);
  }

  static boolean bissexto(int ano){
    return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
  }

}
```

Código Java 7.23: VerificaDatas.java

3 A Páscoa é um evento religioso que ocorre todo ano e pode cair em uma data entre 22 de Março e 25 de Abril. O astrônomo **Jean Baptiste Joseph Delambre** desenvolveu um algoritmo para calcular a data da Páscoa para qualquer ano após 1583. Veja as operações necessárias para realizar esse cálculo.

```
a = ano MOD 19
b = ano / 100
c = ano MOD 100
d = b / 4
e = b MOD 4
f = (b + 8) / 25
g = (b - f + 1) / 3
h = (19 * a + b -d - g + 15) MOD 30
i = c / 4
k = c MOD 4
l = (32 + 2 * e + 2 * i - h - k) MOD 7
m = (a + 11 * h + 22 * l) / 451

mes = (h + 1 - 7 * m + 114) / 31
dia ((h + 1 - 7 * m + 114) MOD 31) + 1
```

Implemente um método que determina a data da Páscoa de um determinado ano posterior a 1583. Complete o código a seguir.

```
class Pascoa {
  public static void main(String[] args) {
    String s = pascoa(2000);
    System.out.println("Páscoa " + s);

    s = pascoa(2012);
    System.out.println("Páscoa " + s);

    s = pascoa(2025);
    System.out.println("Páscoa " + s);

    s = pascoa(2100);
    System.out.println("Páscoa " + s);
  }

}
```

Código Java 7.26: Pascoa.java

4 Implemente um método que determina a partir de uma data (dia, mês e ano) qual será o dia da semana (domingo, segunda, terça, quarta, quinta, sexta e sábado). Há diversos algoritmos para resolver esse problema. Claus Tondering descreve em http://www.faqs.org/faqs/calendars/faq/part1/ uma forma bem simples de solucionar esse problema. A solução utiliza operações matemáticas básicas.

```
a = (14 - mes) / 12;
y = ano - a;
m = mes + 12 * a - 2;
q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
d = q MOD 7;
```

O valor **d** indica o dia da semana de acordo com a seguinte correspondência.

d	dia da semana
0	Domingo
1	Segunda
2	Terça
3	Quarta
4	Quinta
5	Sexta
6	Sábado

Complete o código a seguir.

```
class DiaDaSemana {
  public static void main(String[] args) {
    int d1 = diaDaSemana(30, 10, 1984);

    int d2 = diaDaSemana(2, 4, 1985);

    int d3 = diaDaSemana(12, 12, 1982);

    String[] dias = {
      "domingo",
      "segunda",
      "terça",
      "quarta",
      "quinta",
      "sexta",
      "sábado"
    };

    System.out.println("30/10/1984 foi " + dias[d1]);

    System.out.println("2/4/1985 foi " + dias[d2]);

    System.out.println("12/12/1982 foi " + dias[d3]);
  }

}
```

Código Java 7.29: DiaDaSemana.java

5 Implemente um método que dado um mês e um ano exiba no terminal o calendário de forma semelhante ao exemplo a seguir.

```
Dom Seg Ter Qua Qui Sex Sab
                01  02  03  04
 05  06  07  08  09  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28  29  30  31
```

Complete o código a seguir.

```
1  class ExibeCalendario {
2    public static void main(String[] args) {
3
4      exibeCalendario(10, 1984);
5
6      exibeCalendario(4, 1985);
7
8      exibeCalendario(12, 1982);
9
10     exibeCalendario(2, 2000);
11
12   }
13
14   static boolean bissexto(int ano){
15     return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
16   }
17
18   static int diaDaSemana(int dia, int mes, int ano) {
19     int a = (14 - mes) / 12;
20     int y = ano - a;
21     int m = mes + 12 * a - 2;
22     int q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
23     int d = q % 7;
24
25     return d;
26   }
27
28 }
```

Código Java 7.31: ExibeCalendario.java

6 Implemente um método que determina se um ano é ou não é bissexto. As regras para determinar se um ano é bissexto são:

1. Anos múltiplos de 4 são bissextos.
2. Anos múltiplos de 100 não são bissextos.
3. Anos múltiplos de 400 são bissextos.
4. As últimas regras prevalecem sobre as primeiras.

Complete o código a seguir.

```
1  class AnoBissexto
2  {
3    static void Main()
4    {
5      bool b = bissexto(2000);
```

```
6       System.Console.WriteLine("2000 " + b);
7
8       b = bissexto(2012);
9       System.Console.WriteLine("2012 " + b);
10
11      b = bissexto(2025);
12      System.Console.WriteLine("2025 " + b);
13
14      b = bissexto(2100);
15      System.Console.WriteLine("2100 " + b);
16    }
17
18  }
```

Código C# 7.9: AnoBissexto.cs

7 Implemente um método que verifica se uma determinada data é válida ou não. Lembre-se que Janeiro, Março, Maio, Julho, Agosto, Outubro e Dezembro possuem 31 dias; Abril, Junho, Setembro e Novembro possuem 30 dias; e Fevereiro possui 28 dias em anos não bissextos e 29 dias em anos bissextos. Complete o código a seguir.

```
1   class VerificaDatas
2   {
3     static void Main()
4     {
5       bool b = verificaData(29, 2, 2000);
6
7       System.Console.WriteLine("29/02/2000 - " + b);
8
9       b = verificaData(29, 2, 2004);
10
11      System.Console.WriteLine("29/02/2004 - " + b);
12
13      b = verificaData(31, 4, 2000);
14
15      System.Console.WriteLine("31/04/2000 - " + b);
16    }
17
18    static bool bissexto(int ano)
19    {
20      return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
21    }
22
23
24  }
```

Código C# 7.11: VerificaDatas.cs

8 A Páscoa é um evento religioso que ocorre todo ano e pode cair em uma data entre 22 de Março e 25 de Abril. O astrônomo **Jean Baptiste Joseph Delambre** desenvolveu um algoritmo para calcular a data da Páscoa para qualquer ano após 1583. Veja as operações necessárias para realizar esse cálculo.

```
1   a = ano MOD 19
2   b = ano / 100
3   c = ano MOD 100
4   d = b / 4
5   e = b MOD 4
6   f = (b + 8) / 25
7   g = (b - f + 1) / 3
8   h = (19 * a + b -d - g + 15) MOD 30
9   i = c / 4
10  k = c MOD 4
```

```
l = (32 + 2 * e + 2 * i - h - k) MOD 7
m = (a + 11 * h + 22 * l) / 451

mes = (h + 1 - 7 * m + 114) / 31
dia ((h + 1 - 7 * m + 114) MOD 31) + 1
```

Implemente um método que determina a data da Páscoa de um determinado ano posterior a 1583. Complete o código a seguir.

```
class Pascoa
{
  static void Main()
  {
    String s = pascoa(2000);
    System.Console.WriteLine("Páscoa " + s);

    s = pascoa(2012);
    System.Console.WriteLine("Páscoa " + s);

    s = pascoa(2025);
    System.Console.WriteLine("Páscoa " + s);

    s = pascoa(2100);
    System.Console.WriteLine("Páscoa " + s);
  }

}
```

Código C# 7.14: Pascoa.cs

9 Implemente um método que determina a partir de uma data (dia, mês e ano) qual será o dia da semana (domingo, segunda, terça, quarta, quinta, sexta e sábado). Há diversos algoritmos para resolver esse problema. Claus Tondering descreve em http://www.faqs.org/faqs/calendars/faq/part1/ uma forma bem simples de solucionar esse problema. A solução utiliza operações matemáticas básicas.

```
a = (14 - mes) / 12;
y = ano - a;
m = mes + 12 * a - 2;
q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
d = q MOD 7;
```

O valor **d** indica o dia da semana de acordo com a seguinte correspondência.

d	dia da semana
0	Domingo
1	Segunda
2	Terça
3	Quarta
4	Quinta
5	Sexta
6	Sábado

Complete o código a seguir.

```
class DiaDaSemana
{
```

```
3   static void Main()
4   {
5     int d1 = diaDaSemana(30, 10, 1984);
6
7     int d2 = diaDaSemana(2, 4, 1985);
8
9     int d3 = diaDaSemana(12, 12, 1982);
10
11    String[] dias =
12    {
13      "domingo",
14      "segunda",
15      "terça",
16      "quarta",
17      "quinta",
18      "sexta",
19      "sábado"
20    };
21
22    System.Console.WriteLine("30/10/1984 foi " + dias[d1]);
23
24    System.Console.WriteLine("2/4/1985 foi " + dias[d2]);
25
26    System.Console.WriteLine("12/12/1982 foi " + dias[d3]);
27  }
28
29 }
```

Código C# 7.17: DiaDaSemana.cs

10 Implemente um método que dado um mês e um ano exiba no terminal o calendário de forma semelhante ao exemplo a seguir.

```
Dom Seg Ter Qua Qui Sex Sab
                01  02  03  04
 05  06  07  08  09  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28  29  30  31
```

Complete o código a seguir.

```
1  class ExibeCalendario
2  {
3    static void Main()
4    {
5
6      exibeCalendario(10, 1984);
7
8      exibeCalendario(4, 1985);
9
10     exibeCalendario(12, 1982);
11
12     exibeCalendario(2, 2000);
13
14   }
15
16   static bool bissexto(int ano)
17   {
18     return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
19   }
20
21   static int diaDaSemana(int dia, int mes, int ano)
22   {
23     int a = (14 - mes) / 12;
24     int y = ano - a;
```

```
25      int m = mes + 12 * a - 2;
26      int q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
27      int d = q % 7;
28
29      return d;
30    }
31
32  }
```

Código C# 7.19: ExibeCalendario.cs

Resumo do Capítulo

1 Para evitar a repetição de um determinado trecho de código, podemos criar uma **função** ou um **método**.

2 Em Java ou C#, a palavra reservada **void** é utilizada em métodos que **não devolvem resposta**.

3 Devemos definir um tipo de retorno para criar funções ou métodos que devolvem respostas.

4 Em Java ou C#, a palavra reservada **return** é utilizada, no corpo dos métodos, para devolver uma resposta.

5 Uma função ou um método pode ter **0 ou mais parâmetros**.

6 Um **parâmetro** é uma **variável local** de uma função ou de um método.

Prova

1 Qual alternativa está correta?

a) Uma função pode ter vários tipos de retorno.

b) Obrigatoriamente, os métodos precisam ter parâmetros.

c) Um método definido com **void** não devolve resposta.

d) Em linguagens orientadas a objetos, utilizamos o termo função e não o termo método.

e) A palavra **return** é utilizada para guardar a resposta de uma função.

2 Considere o seguinte método.

```
1 static void teste() {
2   int a = 1;
3 }
```

Qual alternativa está correta?

a) O método **teste** não devolve resposta.

b) O método **teste** sempre devolve o valor **1**.

c) O método **teste** recebe um valor do tipo **int** como parâmetro.

d) O método **teste** não compila pois faltou o comando **return**.

e) O método **teste** devolve valores do tipo **int**.

3 Considere o seguinte método.

```
1 static double teste(int a, double b) {
2   return a + b;
3 }
```

Qual chamada a esse método está correta?

```
1 // I
2 int a = teste(1.0, 1.0);
```

```
1 // II
2 double a = teste(1.0, 1.0);
```

```
1 // III
2 int a = teste(1, 1.0);
```

```
1 // IV
2 double a = teste(1.0, 1);
```

```
1 // V
2 double a = teste(1, 1);
```

a) I

b) II

c) III

d) IV

e) V

4 Considere o seguinte método.

```
1 static int teste(int a, int b) {
2   return a + b;
3 }
```

Quais chamadas a esse método estão corretas?

```
1 // I
2 int a = teste(1.0, 1.0);
```

```
1 // II
2 double a = teste(1, 1);
```

```
1 // III
2 int a = teste(1, 1);
```

```
1 // IV
2 int a = teste(1);
```

```
1 // V
2 int a = teste(1; 1);
```

a) Todas

b) I, II e III

c) II e IV

d) II e III

e) Nenhuma

5 Considere o seguinte método.

```
1 static double teste(int a, int b) {
2   return a + b;
3 }
```

Qual alternativa está correta?

a) O método **teste** não devolve resposta.

b) O método **teste** não compila pois métodos que devolvem **double** não podem devolver valores do tipo **int**.

c) O método **teste** não compila pois métodos não podem receber dois parâmetros.

d) O método **teste** não compila pois ele deveria ser **void**.

e) O método **teste** não possui nenhum problema.

6 Considere o seguinte método em Java.

```
1 static String teste(int a, int b) {
2   return a + b;
3 }
```

Qual alternativa está correta?

a) O método **teste** não devolve resposta.

b) O método **teste** não compila pois métodos que devolvem **String** não podem devolver valores do tipo **int**.

c) O método **teste** não compila pois ele deveria receber duas strings como parâmetro.

d) O método **teste** compilaria com uma operação de casting.

e) O método **teste** não possui nenhum problema.

Minha Pontuação [ ] Pontuação Mínima: [4] Pontuação Máxima: [6]

PROBLEMAS

APÊNDICE A

Para exercitar todo o conteúdo desse treinamento, resolveremos alguns problemas desenvolvendo programas de computador.

Encontrar o maior ou o menor elemento de um array

Considere um array que armazena os preços dos produtos de uma loja. A nossa tarefa é desenvolver um método que encontre a posição do maior elemento desse array e outro método para encontrar a posição do menor. Vamos considerar apenas arrays com pelo menos um elemento.

Vamos começar definido o esqueleto desses métodos. Podemos focar apenas no método que encontra o maior elemento de um array pois o outro método será praticamente igual. Esse método deve receber como parâmetro um array de números e deve devolver como resposta a posição do maior número armazenado nesse array.

```
1 public static int maior(double[] array) {
2
3 }
```

Na nossa estratégia para encontrar o maior elemento de um array de números, o ponto de partida é assumir que o primeiro elemento do array é o maior. Observe no código a seguir que a variável **posicaoDoMaior** é inicializada com o valor **0**. Esse valor indica justamente que estamos assumindo que o primeiro elemento do array é o maior.

```
1 public static int maior(double[] array) {
2   int posicaoDoMaior = 0;
3 }
```

Depois, devemos comparar o elemento que assumimos ser o maior com os outros valores armazenados no array. Observe que o laço, no código abaixo, inicia na posição **1**, ou seja, na segunda posição do array e vai até a última posição. A cada iteração, o elemento que supostamente é o maior elemento do array é comparado com um dos outros valores do array.

```
1 public static int maior(double[] array) {
2   int posicaoDoMaior = 0;
3
4   for(int i = 1; i < array.length; i++) {
5     if(array[posicaoDoMaior] < array[i]) {
6     }
7   }
8 }
```

Durante o laço, se um valor superior ao valor que consideramos ser o maior for encontrado assumiremos esse novo valor como sendo o maior e descartaremos o anterior. Observe a atualização da

variável **posicaoDoMaior** quando um valor "melhor" é encontrado.

```
1 public static int maior(double[] array) {
2   int posicaoDoMaior = 0;
3
4   for(int i = 1; i < array.length; i++) {
5     if(array[posicaoDoMaior] < array[i]) {
6       posicaoDoMaior = i;
7     }
8   }
9 }
```

Ao final do laço, todos os valores foram comparados e a variável **posicaoDoMaior** armazena a posição do maior elemento do array. Então, basta devolver como resposta esse valor.

```
1  public static int maior(double[] array) {
2    int posicaoDoMaior = 0;
3
4    for(int i = 1; i < array.length; i++) {
5      if(array[posicaoDoMaior] < array[i]) {
6        posicaoDoMaior = i;
7      }
8    }
9    return posicaoDoMaior;
10 }
```

Para encontrar o menor elemento, basta inverter a comparação realizada na condição da instrução **if**.

Exercícios de Fixação Com Java

1 Abra um terminal; Entre na pasta dos seus exercícios e crie uma pasta chamada **problemas** para os arquivos desenvolvidos nesse capítulo.

```
K19/rafael$ mkdir problemas

K19/rafael$ cd problemas

K19/rafael/problemas$
```

Terminal A.1: Criando a pasta `problemas` *no Linux*

```
C:\Users\K19\rafael> md problemas

C:\Users\K19\rafael> cd problemas

C:\Users\K19\rafael\problemas>
```

Terminal A.2: Criando a pasta `problemas` *no Windows*

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao1.zip

2 Na pasta **problemas**, crie um arquivo chamado **AchaMaiorOuMenor.java**.

```
1 class AchaMaiorOuMenor {
2   public static void main(String[] args) {
3     double[] array = {-10.7, 37.8, 101.1, 28, -4.9};
```

```
4
5       int posicaoDoMaior = maior(array);
6       System.out.println("O maior valor do array é: " + array[posicaoDoMaior]);
7       System.out.println("Esse valor está na posição: " + posicaoDoMaior);
8
9       int posicaoDoMenor = menor(array);
10      System.out.println("O menor valor do array é: " + array[posicaoDoMenor]);
11      System.out.println("Esse valor está na posição: " + posicaoDoMenor);
12    }
13
14    public static int maior(double[] array) {
15      int posicaoDoMaior = 0;
16
17      for(int i = 1; i < array.length; i++) {
18        if(array[posicaoDoMaior] < array[i]) {
19          posicaoDoMaior = i;
20        }
21      }
22
23      return posicaoDoMaior;
24    }
25
26    public static int menor(double[] array) {
27      int posicaoDoMenor = 0;
28
29      for(int i = 1; i < array.length; i++) {
30        if(array[posicaoDoMenor] > array[i]) {
31          posicaoDoMenor = i;
32        }
33      }
34
35      return posicaoDoMenor;
36    }
37  }
```

Código Java A.6: AchaMaiorOuMenor.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao2.zip

3 Compile e execute a classe **AchaMaiorOuMenor**.

```
K19/rafael/problemas$ javac AchaMaiorOuMenor.java

K19/rafael/problemas$ java AchaMaiorOuMenor
O maior valor do array é: 101.1
Esse valor está na posição: 2
O menor valor do array é: -10.7
Esse valor está na posição: 0
```

Terminal A.3: Compilando e executando a classe `AchaMaiorOuMenor`

Calcular a soma dos elementos de um array

Considere um array que armazena a quantidade de pontos que uma equipe de basquete efetuou nos jogos de um campeonato. A primeira posição do array armazena a quantidade de pontos efetuados no primeiro jogo. A segunda posição armazena a quantidade de pontos efetuados no segundo jogo. E assim por diante. A nossa tarefa é calcular quantos pontos essa equipe fez no campeonato todo.

Então, vamos implementar um método que calcule a soma dos elementos de um array. Esse método pode receber como parâmetro os elementos que devem ser somados e devolver como resposta

a soma propriamente.

```
public static double soma(double[] array) {

}
```

A ideia é somar um elemento de cada vez. O primeiro passo da nossa estratégia é declarar uma variável para acumular os valores armazenados no array. Essa variável deve ser inicializada com o número **0** para não interferir no resultado final.

```
public static double soma(double[] array) {
  double soma = 0;
}
```

Na sequência, os elementos serão adicionados na variável **soma** um a um.

```
public static double soma(double[] array) {
  double soma = 0;

  for(int i = 0; i < array.length; i++) {
    soma += array[i];
  }
}
```

Ao final do laço, a valor da variável **soma** é justamente a somatória dos valores contidos no array. Então, para finalizar, basta devolver esse valor.

```
public static double soma(double[] array) {
  double soma = 0;

  for(int i = 0; i < array.length; i++) {
    soma += array[i];
  }

  return soma;
}
```

Exercícios de Fixação Com Java

4 Na pasta **problemas**, crie um arquivo chamado **Soma.java**.

```
class Soma {
  public static void main(String[] args) {
    double[] array = {-10.7, 37.8, 101.1, 28, -4.9};

    double soma = soma(array);

    System.out.println("A soma dos elementos do array é: " + soma);
  }

  public static double soma(double[] array) {
    double soma = 0;

    for(int i = 0; i < array.length; i++) {
      soma += array[i];
    }

```

```
17      return soma;
18    }
19  }
```

Código Java A.11: Soma.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao4.zip

5 Compile e execute a classe **Soma**.

```
K19/rafael/problemas$ javac Soma.java

K19/rafael/problemas$ java Soma
A soma dos elementos do array é: 151.29999999999998
```

Terminal A.4: Compilando e executando a classe Soma

Calcular a média dos elementos de um array

Considere um array que armazena a quantidade de pontos que uma equipe de basquete efetuou nos jogos de um campeonato. A primeira posição do array armazena a quantidade de pontos efetuados no primeiro jogo. A segunda posição armazena a quantidade de pontos efetuados no segundo jogo. E assim por diante. A nossa tarefa é calcular quantos pontos em média essa equipe efetua por partida.

O cálculo da média pode ser realizado em duas etapas. Na primeira, somamos os valores armazenados no array. Na segunda, dividimos essa soma pela quantidade de elementos do array. A soma pode ser realizada com o método criado anteriormente.

O método que efetuará o cálculo da média deverá receber como parâmetro o array com os valores que devem ser considerados nesse cálculo.

```
1  public static double media(double[] array) {
2
3  }
```

O método **media** pode chamar o método **soma** para obter a somatória dos elementos do array.

```
1  public static double media(double[] array) {
2    double soma = soma(array);
3  }
```

Com o valor da somatória armazenado na variável **soma**, a média é obtida com uma divisão.

```
1  public static double media(double[] array) {
2    double soma = soma(array);
3    double media = soma / array.length;
4  }
```

Por fim, basta devolver o valor armazenado na variável **media**.

```
1  public static double media(double[] array) {
2    double soma = soma(array);
3    double media = soma / array.length;
```

```
4     return media;
5   }
```

Exercícios de Fixação Com Java

6 Na pasta **problemas**, crie um arquivo chamado **Media.java**.

```
1  class Media {
2    public static void main(String[] args) {
3      double[] array = {-10.7, 37.8, 101.1, 28, -4.9};
4
5      double media = media(array);
6
7      System.out.println("A média dos elementos do array é: " + media);
8    }
9
10   public static double media(double[] array) {
11     double soma = soma(array);
12     double media = soma / array.length;
13     return media;
14   }
15
16   public static double soma(double[] array) {
17     double soma = 0;
18
19     for(int i = 0; i < array.length; i++) {
20       soma += array[i];
21     }
22
23     return soma;
24   }
25 }
```

Código Java A.16: Media.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao6.zip

7 Compile e execute a classe **Media**.

```
K19/rafael/problemas$ javac Media.java

K19/rafael/problemas$ java Media
A média dos elementos do array é: 30.259999999999998
```

Terminal A.5: Compilando e executando a classe `Media`

Trocar as posições de dois elementos de um array

Considere a distribuição das vagas da garagem de um condomínio. Essas vagas foram numeradas de 1 a 100 e cada vaga está associada ao número do apartamento que a utilizará.

Podemos utilizar um array de números inteiros para armazenar essas associações. Vamos assumir que a posição 0 desse array corresponde à vaga número 1, a posição 1 à vaga número 2 e assim por diante. O número do apartamento que utilizará uma determinada vaga deve ser armazenado na

posição correspondente a essa vaga.

Por exemplo, considere o array {12, 34, 11, 22}. De acordo com as informações contidas nesse array, podemos deduzir que a vaga 1 pertence ao apartamento 12, a vaga 2 ao apartamento 34, a vaga 3 ao apartamento 11 e a vaga 4 ao apartamento 22. De acordo com o interesse dos moradores, as vagas podem ser trocadas entre os apartamentos. implemente um método que realize a troca dos valores contidos em duas posições de um array.

Vamos começar definindo o esqueleto do método que realizará essas trocas. Podemos definir três parâmetros para esse método. O primeiro é o array que contém os elementos que serão trocados. O segundo é a posição de um dos dois elementos que serão trocados e o terceiro é a posição do outro elemento.

```
1 public static void troca(int[] array, int i, int j) {
2
3 }
```

Basicamente, o elemento da posição **j** deve ser armazenado na posição **i** e vice versa. Considere o seguinte código.

```
1 public static void troca(int[] array, int i, int j) {
2   array[i] = array[j];
3 }
```

Na atribuição em destaque, guardamos o elemento da posição **j** na posição **i**. Nessa operação, perderemos o valor antigo da posição **i**. Dessa forma, não poderemos guardá-lo na posição **j**. Para solucionar esse problema, basta armazenar esse elemento em uma variável auxiliar antes dessa atribuição.

```
1 public static void troca(int[] array, int i, int j) {
2   int auxiliar = array[i];
3   array[i] = array[j];
4 }
```

Agora, o valor da posição **i** é armazenado na variável **auxiliar**. Depois, o valor da posição **j** é armazenado na posição **i**. Por fim, devemos armazenar o valor da variável **auxiliar** (antigo valor da posição **i**) na posição **j**.

```
1 public static void troca(int[] array, int i, int j) {
2   int auxiliar = array[i];
3   array[i] = array[j];
4   array[j] = auxiliar;
5 }
```

Exercícios de Fixação Com Java

8 Na pasta **problemas**, crie um arquivo chamado **Troca.java**.

```
1 class Troca {
2   public static void main(String[] args) {
3     System.out.println("Original");
```

```
4     int[] array = {-10, 37, 101, 28, -4};
5     exibeArray(array);
6
7     System.out.println("\nTroca 0 e 2");
8     troca(array, 0, 2);
9     exibeArray(array);
10
11    System.out.println("\nTroca 1 e 3");
12    troca(array, 1, 3);
13    exibeArray(array);
14  }
15
16  public static void troca(int[] array, int i, int j) {
17    int auxiliar = array[i];
18    array[i] = array[j];
19    array[j] = auxiliar;
20  }
21
22  public static void exibeArray(int[] array) {
23    System.out.print("{");
24
25    for(int i = 0; i < array.length - 1; i++) {
26      System.out.print(array[i] + ", ");
27    }
28
29    System.out.println(array[array.length - 1] + "}");
30  }
31 }
```

Código Java A.21: Troca.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao8.zip

9 Compile e execute a classe **Troca**.

```
K19/rafael/problemas$ javac Troca.java

K19/rafael/problemas$ java Troca
Original
{-10, 37, 101, 28, -4}

Troca 0 e 2
{101, 37, -10, 28, -4}

Troca 1 e 3
{101, 28, -10, 37, -4}
```

Terminal A.6: Compilando e executando a classe `Troca`

Escolher aleatoriamente um número inteiro dentro de um intervalo

Considere o sorteio de um premio entre os participantes de uma determinada palestra. Os números de 100 a 300 foram distribuídos a esses participantes na entrada da palestra. Implemente um método que escolha aleatoriamente um número inteiro dentro de um determinado intervalo.

Novamente, vamos começar definindo o esqueleto do método desejado. Os parâmetros definem o intervalo que deve ser considerado na escolha aleatória. O primeiro define o início do intervalo e o segundo o término. Vamos considerar que o primeiro parâmetro é menor do que o segundo. Além disso, a resposta desse método será número sorteado.

```
1 public static int aleatorio(int i, int j) {
```

```
2 }
```

O trecho de código "Math.random()" gera um número do tipo **double** maior ou igual a **0** e menor do que **1**. Podemos utilizá-lo como base para gerar um número inteiro em um determinado intervalo.

A quantidade de elementos no intervalo **[i, j]** é **j - i + 1**. Multiplicando esse valor pelo número gerado com o "Math.random()", obteremos um número do tipo **double** maior ou igual a **0** e menor do que **j - i + 1**.

```
1 public static int aleatorio(int i, int j) {
2   double a = Math.random() * (j - i + 1);
3 }
```

Como o objetivo é o obter um número inteiro, podemos realizar uma operação de casting. Com isso teremos um número inteiro maior ou igual a **0** e menor do que **j - i + 1**.

```
1 public static int aleatorio(int i, int j) {
2   double a = Math.random() * (j - i + 1);
3   int b = (int)a;
4 }
```

Na sequência, podemos realizar uma operação de soma para ajustar o número obtido aos limites do intervalo desejado.

```
1 public static int aleatorio(int i, int j) {
2   double a = Math.random() * (j - i + 1);
3   int b = (int)a;
4   int c = b + i;
5 }
```

Com a soma em destaque, obteremos um número inteiro maior ou igual a **i** (**0 + i**) e menor do que **j + 1** (**j - i + 1 + i**). Em outras palavras, obteremos um número inteiro maior ou igual a **i** e menor ou igual a **j**. Para finalizar, basta devolver esse valor.

```
1 public static int aleatorio(int i, int j) {
2   double a = Math.random() * (j - i + 1);
3   int b = (int)a;
4   int c = b + i;
5   return c;
6 }
```

Podemos simplificar o código do método **aleatorio**.

```
1 public static int aleatorio(int i, int j) {
2   return (int)(Math.random() * (j - i + 1)) + i;
3 }
```

Exercícios de Fixação Com Java

10 Na pasta **problemas**, crie um arquivo chamado **Aleatorio.java**.

```
class Aleatorio {
  public static void main(String[] args) {
    System.out.println("Sorteando no intervalo [0, 10]");
    for(int i = 0; i < 10; i++) {
      int numero = aleatorio(0, 10);
      System.out.println(numero);
    }

    System.out.println("\nSorteando no intervalo [-25, 10]");
    for(int i = 0; i < 10; i++) {
      int numero = aleatorio(-25, 10);
      System.out.println(numero);
    }
  }

  public static int aleatorio(int i, int j) {
    return (int)(Math.random() * (j - i + 1)) + i;
  }
}
```

Código Java A.28: Aleatorio.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao10.zip

11 Compile e execute a classe **Aleatorio**.

```
K19/rafael/problemas$ javac Aleatorio.java

K19/rafael/problemas$ java Aleatorio
Sorteando no intervalo [0, 10]
6
1
4
2
3
7
9
8
10
2

Sorteando no intervalo [-25, 10]
4
3
-1
-11
-14
-7
-2
-25
9
-22
```

Terminal A.7: Compilando e executando a classe `Aleatorio`

Gerar apostas da Mega-Sena

Você não quer perder tempo escolhendo os números que utilizará para apostar na Mega-Sena. Implemente um programa que gere aleatoriamente apostas da Mega-Sena com 6 números.

Para gerar apostas da Mega-Sena, devemos escolher aleatoriamente 6 números entre 1 e 60. Considere o seguinte método.

```
public static int[] geraApostaMegaSena() {
}
```

Toda vez que for chamado, o método **geraApostaMegaSena** devolverá um array de **int** com 6 números. Esses números são justamente os que formam a aposta da Mega-Sena.

Podemos começar a implementação do método **geraApostaMegaSena**, armazenando em um array os números de 1 a 60. Esses são os números que podem ser sorteados.

```
1 public static int[] geraApostaMegaSena() {
2   int[] numeros = new int[60];
3   for(int i = 0; i < numeros.length; i++) {
4     numeros[i] = i + 1;
5   }
6 }
```

Depois, devemos escolher aleatoriamente uma das posições do array **numeros**. O valor armazenado nessa posição será o primeiro número da aposta. Para realizar essa escolha, podemos utilizar o método **aleatorio** que foi desenvolvido anteriormente.

```
1 public static int[] geraApostaMegaSena() {
2   int[] numeros = new int[60];
3   for(int i = 0; i < numeros.length; i++) {
4     numeros[i] = i + 1;
5   }
6
7   int j = aleatorio(0, numeros.length - 1);
8   aposta[0] = numeros[j];
9 }
```

Antes de escolher aleatoriamente o segundo número da aposta, para não correr o risco de sortear o mesmo número duas vezes, realizaremos uma troca entre o primeiro número sorteado e o primeiro elemento do array **numeros**. Para isso, podemos utilizar o método **troca** que foi criado anteriormente.

```
1  public static int[] geraApostaMegaSena() {
2    int[] numeros = new int[60];
3    for(int i = 0; i < numeros.length; i++) {
4      numeros[i] = i + 1;
5    }
6
7    int j = aleatorio(0, numeros.length - 1);
8    aposta[0] = numeros[j];
9    troca(numeros, 0, j);
10 }
```

Agora, o próximo passo é sortear mais um elemento do array **numeros** só que devemos desconsiderando o primeiro pois esse já foi sorteado. O número sorteado dessa vez será o segundo número da aposta que está sendo definida.

```
1  public static int[] geraApostaMegaSena() {
2    int[] numeros = new int[60];
3    for(int i = 0; i < numeros.length; i++) {
4      numeros[i] = i + 1;
5    }
6
7    int j = aleatorio(0, numeros.length - 1);
8    aposta[0] = numeros[j];
9    troca(numeros, 0, j);
10
11   j = aleatorio(1, numeros.length - 1);
12   aposta[1] = numeros[j];
13 }
```

Novamente, para não ser escolhido duas vezes, o último elemento sorteado será trocado com o segundo elemento do array **numeros**.

```
public static int[] geraApostaMegaSena() {
  int[] numeros = new int[60];
  for(int i = 0; i < numeros.length; i++) {
    numeros[i] = i + 1;
  }

  int j = aleatorio(0, numeros.length - 1);
  aposta[0] = numeros[j];
  troca(numeros, 0, j);

  j = aleatorio(1, numeros.length - 1);
  aposta[1] = numeros[j];
  troca(numeros, 1, j);
}
```

Analogamente, os outros 4 números que formam a aposta podem ser definidos.

```
public static int[] geraApostaMegaSena() {
  int[] numeros = new int[60];
  for(int i = 0; i < numeros.length; i++) {
    numeros[i] = i + 1;
  }

  int j = aleatorio(0, numeros.length - 1);
  aposta[0] = numeros[j];
  troca(numeros, 0, j);

  j = aleatorio(1, numeros.length - 1);
  aposta[1] = numeros[j];
  troca(numeros, 1, j);

  j = aleatorio(2, numeros.length - 1);
  aposta[2] = numeros[j];
  troca(numeros, 2, j);

  j = aleatorio(3, numeros.length - 1);
  aposta[3] = numeros[j];
  troca(numeros, 3, j);

  j = aleatorio(4, numeros.length - 1);
  aposta[4] = numeros[j];
  troca(numeros, 4, j);

  j = aleatorio(5, numeros.length - 1);
  aposta[5] = numeros[j];
  troca(numeros, 5, j);
}
```

Observe a formação de um padrão no código. Sorteamos um número; armazenamos no array **aposta**; e realizamos uma troca. Dado esse padrão, podemos utilizar um laço para sortear os 6 números da aposta.

```
public static int[] geraApostaMegaSena() {
  int[] numeros = new int[60];
  for(int i = 0; i < numeros.length; i++) {
    numeros[i] = i + 1;
  }

  for(int i = 0; i < aposta.length; i++) {
    int j = aleatorio(i, numeros.length - 1);
    aposta[i] = numeros[j];
    troca(numeros, i, j);
```

```
  }
}
```

Para finalizar, o método **geraApostaMegaSena** deve devolver a aposta gerada.

```
public static int[] geraApostaMegaSena() {
  int[] numeros = new int[60];
  for(int i = 0; i < numeros.length; i++) {
    numeros[i] = i + 1;
  }

  for(int i = 0; i < aposta.length; i++) {
    int j = aleatorio(i, numeros.length - 1);
    aposta[i] = numeros[j];
    troca(numeros, i, j);
  }

  return aposta;
}
```

Exercícios de Fixação Com Java

12 Na pasta **problemas**, crie um arquivo chamado **GeraApostaMegaSena.java**.

```
class GeraApostaMegaSena {
  public static void main(String[] args) {
    for(int i = 0; i < 10; i++) {
      int[] array = geraApostaMegaSena();
      System.out.print("Aposta " + (i + 1) + ": ");
      exibeArray(array);
    }
  }

  public static int[] geraApostaMegaSena() {
    int[] numeros = new int[60];
    for(int i = 0; i < numeros.length; i++) {
      numeros[i] = i + 1;
    }

    int[] aposta = new int[6];
    for(int i = 0; i < aposta.length; i++) {
      int j = aleatorio(i, numeros.length - 1);
      aposta[i] = numeros[j];
      troca(numeros, i, j);
    }

    return aposta;
  }

  public static int aleatorio(int i, int j) {
    return (int)(Math.random() * (j - i + 1)) + i;
  }

  public static void troca(int[] array, int i, int j) {
    int auxiliar = array[i];
    array[i] = array[j];
    array[j] = auxiliar;
  }

  public static void exibeArray(int[] array) {
    System.out.print("{");

```

```
39      for(int i = 0; i < array.length - 1; i++) {
40        System.out.print(array[i] + ", ");
41      }
42
43      System.out.println(array[array.length - 1] + "}");
44    }
45  }
```

Código Java A.38: GeraApostaMegaSena.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao12.zip

13 Compile e execute a classe **GeraApostaMegaSena**.

```
K19/rafael/problemas$ javac GeraApostaMegaSena.java

K19/rafael/problemas$ java GeraApostaMegaSena
Aposta 1: {18, 46, 55, 42, 22, 34}
Aposta 2: {26, 38, 10, 28, 25, 49}
Aposta 3: {2, 35, 59, 42, 45, 51}
Aposta 4: {22, 21, 7, 15, 34, 27}
Aposta 5: {11, 4, 17, 21, 32, 48}
Aposta 6: {32, 60, 7, 25, 40, 16}
Aposta 7: {40, 5, 39, 49, 43, 35}
Aposta 8: {16, 59, 15, 27, 13, 43}
Aposta 9: {40, 28, 7, 10, 29, 59}
Aposta 10: {10, 29, 20, 17, 43, 53}
```

Terminal A.8: Compilando e executando a classe `GeraApostaMegaSena`

Embaralhar os elementos de um array

Considere a distribuição das vagas da garagem de um condomínio. Essas vagas foram numeradas de 1 a 100 e cada vaga está associada ao número do apartamento que a utilizará.

Podemos utilizar um array de números inteiros para armazenar essas associações. Vamos assumir que a posição 0 desse array corresponde à vaga número 1, a posição 1 à vaga número 2 e assim por diante. O número do apartamento que utilizará uma determinada vaga deve ser armazenado na posição correspondente a essa vaga.

Por exemplo, considere o array 12, 34, 11, 22. De acordo com as informações contidas nesse array, podemos deduzir que a vaga 1 pertence ao apartamento 12, a vaga 2 ao apartamento 34, a vaga 3 ao apartamento 11 e a vaga 4 ao apartamento 22.

No final de cada semestre, as vagas são redistribuídas de forma aleatória. Implemente um método que embaralhe os elementos de um array para simular essa redistribuição.

Podemos utilizar uma estratégia muito parecida com a utilizada para gerar apostas de 6 números da Mega-Sena. Basicamente, temos que continuar o processo de sorteio até o último elemento do array. Outra diferença é que o array pode possuir elementos fora do intervalo de 1 a 60, mas essa diferença não atrapalha a nossa estratégia.

```
1  public static void embaralha(int[] array) {
2    for(int i = 0; i < array.length; i++) {
3      int j = aleatorio(i, array.length - 1);
4      troca(array, i, j);
5    }
6  }
```

Exercícios de Fixação Com Java

14 Na pasta **problemas**, crie um arquivo chamado **Embaralha.java**.

```
class Embaralha {
  public static void main(String[] args) {
    System.out.println("Original");
    int[] array = {-10, 37, 101, 28, -4};
    exibeArray(array);

    System.out.println("\nEmbaralhando...");
    embaralha(array);
    exibeArray(array);

    System.out.println("\nEmbaralhando...");
    embaralha(array);
    exibeArray(array);

    System.out.println("\nEmbaralhando...");
    embaralha(array);
    exibeArray(array);
  }

  public static void embaralha(int[] array) {
    for(int i = 0; i < array.length; i++) {
      int j = aleatorio(i, array.length - 1);
      troca(array, i, j);
    }
  }

  public static int aleatorio(int i, int j) {
    return (int)(Math.random() * (j - i + 1)) + i;
  }

  public static void troca(int[] array, int i, int j) {
    int auxiliar = array[i];
    array[i] = array[j];
    array[j] = auxiliar;
  }

  public static void exibeArray(int[] array) {
    System.out.print("{");

    for(int i = 0; i < array.length - 1; i++) {
      System.out.print(array[i] + ", ");
    }

    System.out.println(array[array.length - 1] + "}");
  }
}
```

Código Java A.40: Embaralha.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao14.zip

15 Compile e execute a classe **Embaralha**.

```
K19/rafael/problemas$ javac Embaralha.java

K19/rafael/problemas$ java Embaralha
Original
{-10, 37, 101, 28, -4}
```

```
Embaralhando...
{-4, 28, 101, 37, -10}

Embaralhando...
{37, 101, 28, -10, -4}

Embaralhando...
{-4, 101, 37, -10, 28}
```

Terminal A.9: Compilando e executando a classe `Embaralha`

Ordenar os elementos de um array

Considere um array contendo os preços dos produtos de uma loja. Esse valores não estão ordenados. Implemente um método para ordenar esse elementos do menor para o maior.

Por mais contraditório que pareça, podemos ordenar um array com uma pequena alteração no método que embaralha. A alteração consiste em selecionar a cada iteração o menor elemento do array ao invés de um elemento aleatório.

```
1 public static void ordena(int[] array) {
2   for(int i = 0; i < array.length; i++) {
3     int menor = menor(array, i);
4     troca(array, i, menor);
5   }
6 }
```

Exercícios de Fixação Com Java

16 Na pasta **problemas**, crie um arquivo chamado **Ordena.java**.

```
1  class Ordena {
2    public static void main(String[] args) {
3      System.out.println("Original");
4      int[] array = {-10, 37, 101, 28, -4};
5      exibeArray(array);
6
7      System.out.println("\nOrdenando...");
8      ordena(array);
9      exibeArray(array);
10   }
11
12   public static void ordena(int[] array) {
13     for(int i = 0; i < array.length; i++) {
14       int menor = menor(array, i);
15       troca(array, i, menor);
16     }
17   }
18
19   public static int menor(int[] array, int inicio) {
20     int menor = inicio;
21
22     for(int i = inicio + 1; i < array.length; i++) {
23       if(array[menor] > array[i]) {
24         menor = i;
25       }
26     }
27
```

```
28      return menor;
29    }
30
31    public static void troca(int[] array, int i, int j) {
32      int auxiliar = array[i];
33      array[i] = array[j];
34      array[j] = auxiliar;
35    }
36
37    public static void exibeArray(int[] array) {
38      System.out.print("{");
39
40      for(int i = 0; i < array.length - 1; i++) {
41        System.out.print(array[i] + ", ");
42      }
43
44      System.out.println(array[array.length - 1] + "}");
45    }
46  }
```

Código Java A.42: Ordena.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao16.zip

17 Compile e execute a classe **Ordena**.

```
K19/rafael/problemas$ javac Ordena.java

K19/rafael/problemas$ java Ordena
Original
{-10, 37, 101, 28, -4}

Ordenando...
{-10, -4, 28, 37, 101}
```

Terminal A.10: Compilando e executando a classe Ordena

Inverter o posicionamento dos elementos de um array

Considere um jogo de cartas no qual o posicionamento das cartas deve ser invertido. Implemente um método que inverta os elementos de um array de números. Depois da inversão, o antigo primeiro deve ser o novo último, o antigo segundo deve se o novo penúltimo e assim por diante.

Para começar, vamos definir o esqueleto do método que realizará a inversão dos elementos de um array. Esse método deve receber como parâmetro o array que armazena os elementos que serão invertidos.

```
1  public static void inverte(double[] array) {
2  }
```

O primeiro passo da nossa estratégia para inverter os elementos de um array é realizar uma troca entre o primeiro e o último elemento.

```
1  public static void inverte(double[] array) {
2    troca(array, 0, array.length - 1);
3  }
```

O próximo passo é realizar a troca do segundo e do penúltimo elemento.

```
public static void inverte(double[] array) {
  troca(array, 0, array.length - 1);
  troca(array, 1, array.length - 2);
}
```

Analogamente, no próximo passo, devemos trocar o terceiro e o antepenúltimo.

```
public static void inverte(double[] array) {
  troca(array, 0, array.length - 1);
  troca(array, 1, array.length - 2);
  troca(array, 2, array.length - 3);
}
```

Você já pode deduzir quais seriam os próximos passos. Observe a existência de uma padrão nesse código. Dessa forma, podemos utilizar um laço.

```
public static void inverte(double[] array) {
  int i = 0;
  int j = array.length - 1;

  while(i < j) {
    troca(array, i, j);
    i++;
    j--;
  }
}
```

Exercícios de Fixação Com Java

18 Na pasta **problemas**, crie um arquivo chamado **Inverte.java**.

```
class Inverte {
  public static void main(String[] args) {
    System.out.println("Original");
    int[] array = {-10, 37, 101, 28, -4};
    exibeArray(array);

    System.out.println("\nInvertendo...");
    inverte(array);
    exibeArray(array);
  }

  public static void inverte(int[] array) {
    int i = 0;
    int j = array.length - 1;

    while(i < j) {
      troca(array, i, j);
      i++;
      j--;
    }
  }

  public static void troca(int[] array, int i, int j) {
    int auxiliar = array[i];
    array[i] = array[j];
    array[j] = auxiliar;
  }

```

```
29    public static void exibeArray(int[] array) {
30      System.out.print("{");
31
32      for(int i = 0; i < array.length - 1; i++) {
33        System.out.print(array[i] + ", ");
34      }
35
36      System.out.println(array[array.length - 1] + "}");
37    }
38  }
```

Código Java A.48: Inverte.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao18.zip

19 Compile e execute a classe **Inverte**.

```
K19/rafael/problemas$ javac Inverte.java

K19/rafael/problemas$ java Inverte
Original
{-10, 37, 101, 28, -4}

Invertendo...
{-4, 28, 101, 37, -10}
```

Terminal A.11: Compilando e executando a classe `Inverte`

Números em formato binário

Estamos acostumados a lidar com os números em formato decimal. Os computadores trabalham com números em formato binário. Implemente um método que devolva um número inteiro em formato binário.

O algoritmo para gerar o formato binário de um número inteiro é muito simples. Devemos realizar sucessivas divisões inteiras por 2 sempre armazenando o resto dessas operações. O número em binário é formado pela concatenação desses restos. O último resto é o primeiro dígito do número; o penúltimo resto é o segundo dígito; o antepenúltimo resto é o terceiro dígito; e assim por diante.

Vamos definir o esqueleto do método que gera a representação binária dos números inteiros. Esse método deve receber como parâmetro o número inteiro e devolver como resposta a representação binária desse número.

```
1  public static String decimalParaBinario(int x) {
2  }
```

Aplicando o algoritmo descrito acima, podemos definir o seguinte código.

```
1  public static String binario(int x) {
2    String y = "";
3
4    // último dígito
5    y = x % 2 + y;
6    x = x / 2;
7
8    // penúltimo dígito
9    y = x % 2 + y;
10   x = x / 2;
```

```
11
12    // antepenúltimo dígito
13    y = x % 2 + y;
14    x = x / 2;
15
16    ...
17    return y;
18  }
```

Você pode observar um padrão ocorrendo na implementação acima. Nesse caso, podemos utilizar um laço com a condição de parada é "x > 0".

```
1  public static String binario(int x) {
2    String y = "";
3    while(x > 0) {
4      y = x % 2 + y;
5      x = x / 2;
6    }
7    return y;
8  }
```

Exercícios de Fixação Com Java

20 Na pasta **problemas**, crie um arquivo chamado **Binario.java**.

```
1  class Binario {
2    public static void main(String[] args) {
3      for(int i = 1; i < 100; i++) {
4        String s = binario(i);
5        System.out.println(i + " em binário " + s);
6      }
7    }
8
9    public static String binario(int x) {
10     String y = "";
11     while(x > 0) {
12       y = x % 2 + y;
13       x = x / 2;
14     }
15     return y;
16   }
17 }
```

Código Java A.52: Binario.java

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao20.zip

21 Compile e execute a classe **Binario**.

```
K19/rafael/problemas$ javac Binario.java

K19/rafael/problemas$ java Binario
1 em binário 1
2 em binário 10
3 em binário 11
...
97 em binário 1100001
98 em binário 1100010
99 em binário 1100011
```

Terminal A.12: Compilando e executando a classe `Binario`

Exercícios de Fixação Com C#

22 Na pasta **problemas**, crie um arquivo chamado **AchaMaiorOuMenor.cs**.

```
class AchaMaiorOuMenor
{
  static void Main()
  {
    double[] array = {-10.7, 37.8, 101.1, 28, -4.9};

    int posicaoDoMaior = maior(array);
    System.Console.WriteLine("O maior valor do array é: " + array[posicaoDoMaior]);
    System.Console.WriteLine("Esse valor está na posição: " + posicaoDoMaior);

    int posicaoDoMenor = menor(array);
    System.Console.WriteLine("O menor valor do array é: " + array[posicaoDoMenor]);
    System.Console.WriteLine("Esse valor está na posição: " + posicaoDoMenor);
  }

  static int maior(double[] array)
  {
    int posicaoDoMaior = 0;

    for(int i = 1; i < array.Length; i++)
    {
      if(array[posicaoDoMaior] < array[i])
      {
        posicaoDoMaior = i;
      }
    }

    return posicaoDoMaior;
  }

  static int menor(double[] array)
  {
    int posicaoDoMenor = 0;

    for(int i = 1; i < array.Length; i++)
    {
      if(array[posicaoDoMenor] > array[i])
      {
        posicaoDoMenor = i;
      }
    }

    return posicaoDoMenor;
  }
}
```

Código C# A.1: AchaMaiorOuMenor.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao22.zip

23 Compile e execute a classe **AchaMaiorOuMenor**.

```
C:\Users\K19\rafael\problemas> csc AchaMaiorOuMenor.cs
```

```
C:\Users\K19\rafael\problemas> AchaMaiorOuMenor.exe
O maior valor do array é: 101.1
Esse valor está na posição: 2
O menor valor do array é: -10.7
Esse valor está na posição: 0
```

Terminal A.13: Compilando e executando a classe `AchaMaiorOuMenor`

24 Na pasta **problemas**, crie um arquivo chamado **Soma.cs**.

```
1  class Soma
2  {
3    static void Main()
4    {
5      double[] array = {-10.7, 37.8, 101.1, 28, -4.9};
6
7      double valor = soma(array);
8
9      System.Console.WriteLine("A soma dos elementos do array é: " + valor);
10   }
11
12   static double soma(double[] array)
13   {
14     double soma = 0;
15
16     for(int i = 0; i < array.Length; i++)
17     {
18       soma += array[i];
19     }
20
21     return soma;
22   }
23 }
```

Código C# A.2: Soma.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao24.zip

25 Compile e execute a classe **Soma**.

```
C:\Users\K19\rafael\problemas> csc Soma.cs

C:\Users\K19\rafael\problemas> Soma.exe
A soma dos elementos do array é: 151.3
```

Terminal A.14: Compilando e executando a classe `Soma`

26 Na pasta **problemas**, crie um arquivo chamado **Media.cs**.

```
1  class Media
2  {
3    static void Main()
4    {
5      double[] array = {-10.7, 37.8, 101.1, 28, -4.9};
6
7      double valor = media(array);
8
9      System.Console.WriteLine("A média dos elementos do array é: " + valor);
10   }
11
12   static double media(double[] array)
```

```
13    {
14      double valor = soma(array);
15      double media = valor / array.Length;
16      return media;
17    }
18
19    static double soma(double[] array)
20    {
21      double soma = 0;
22
23      for(int i = 0; i < array.Length; i++)
24      {
25        soma += array[i];
26      }
27
28      return soma;
29    }
30  }
```

Código C# A.3: Media.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao26.zip

27 Compile e execute a classe **Media**.

```
C:\Users\K19\rafael\problemas> csc Media.cs

C:\Users\K19\rafael\problemas> Media.exe
A média dos elementos do array é: 30.26
```

Terminal A.15: Compilando e executando a classe `Media`

28 Na pasta **problemas**, crie um arquivo chamado **Troca.cs**.

```
1   class Troca
2   {
3     static void Main()
4     {
5       System.Console.WriteLine("Original");
6       int[] array = {-10, 37, 101, 28, -4};
7       exibeArray(array);
8
9       System.Console.WriteLine("\nTroca 0 e 2");
10      troca(array, 0, 2);
11      exibeArray(array);
12
13      System.Console.WriteLine("\nTroca 1 e 3");
14      troca(array, 1, 3);
15      exibeArray(array);
16    }
17
18    static void troca(int[] array, int i, int j)
19    {
20      int auxiliar = array[i];
21      array[i] = array[j];
22      array[j] = auxiliar;
23    }
24
25    static void exibeArray(int[] array)
26    {
27      System.Console.Write("{");
28
29      for(int i = 0; i < array.Length - 1; i++)
30      {
31        System.Console.Write(array[i] + ", ");
```

```
32      }
33
34      System.Console.WriteLine(array[array.Length - 1] + "}");
35    }
36  }
```

Código C# A.4: Troca.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao28.zip

29 Compile e execute a classe **Troca**.

```
C:\Users\K19\rafael\problemas> csc Troca.cs

C:\Users\K19\rafael\problemas> Troca.exe
Original
{-10, 37, 101, 28, -4}

Troca 0 e 2
{101, 37, -10, 28, -4}

Troca 1 e 3
{101, 28, -10, 37, -4}
```

Terminal A.16: Compilando e executando a classe `Troca`

30 Na pasta **problemas**, crie um arquivo chamado **Aleatorio.cs**.

```
1  class Aleatorio
2  {
3    static void Main()
4    {
5      System.Console.WriteLine("Sorteando no intervalo [0, 10]");
6      for(int i = 0; i < 10; i++)
7      {
8        int numero = aleatorio(0, 10);
9        System.Console.WriteLine(numero);
10     }
11
12     System.Console.WriteLine("\nSorteando no intervalo [-25, 10]");
13     for(int i = 0; i < 10; i++)
14     {
15       int numero = aleatorio(-25, 10);
16       System.Console.WriteLine(numero);
17     }
18   }
19
20   static int aleatorio(int i, int j)
21   {
22     System.Random gerador = new System.Random();
23     return (int)(gerador.NextDouble() * (j - i + 1)) + i;
24   }
25 }
```

Código C# A.5: Aleatorio.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao30.zip

31 Compile e execute a classe **Aleatorio**.

```
C:\Users\K19\rafael\problemas> csc Aleatorio.cs

C:\Users\K19\rafael\problemas> Aleatorio.exe
```

```
Sorteando no intervalo [0, 10]
6
1
4
2
3
7
9
8
10
2

Sorteando no intervalo [-25, 10]
4
3
-1
-11
-14
-7
-2
-25
9
-22
```

Terminal A.17: Compilando e executando a classe `Aleatorio`

32 Na pasta **problemas**, crie um arquivo chamado **GeraApostaMegaSena.cs**.

```
1  class GeraApostaMegaSena
2  {
3    static void Main()
4    {
5      for(int i = 0; i < 10; i++)
6      {
7        int[] array = geraApostaMegaSena();
8        System.Console.Write("Aposta " + (i + 1) + ": ");
9        exibeArray(array);
10     }
11   }
12
13   static int[] geraApostaMegaSena()
14   {
15     int[] numeros = new int[60];
16     for(int i = 0; i < numeros.Length; i++)
17     {
18       numeros[i] = i + 1;
19     }
20
21     int[] aposta = new int[6];
22     for(int i = 0; i < aposta.Length; i++)
23     {
24       int j = aleatorio(i, numeros.Length - 1);
25       aposta[i] = numeros[j];
26       troca(numeros, i, j);
27     }
28
29     return aposta;
30   }
31
32   static int aleatorio(int i, int j)
33   {
34     System.Random gerador = new System.Random();
35     return (int)(gerador.NextDouble() * (j - i + 1)) + i;
36   }
37
38   static void troca(int[] array, int i, int j)
39   {
40     int auxiliar = array[i];
41     array[i] = array[j];
42     array[j] = auxiliar;
43   }
```

```
44
45    static void exibeArray(int[] array)
46    {
47      System.Console.Write("{");
48
49      for(int i = 0; i < array.Length - 1; i++)
50      {
51        System.Console.Write(array[i] + ", ");
52      }
53
54      System.Console.WriteLine(array[array.Length - 1] + "}");
55    }
56  }
```

Código C# A.6: GeraApostaMegaSena.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao32.zip

33 Compile e execute a classe **GeraApostaMegaSena**.

```
C:\Users\K19\rafael\problemas> csc GeraApostaMegaSena.cs

C:\Users\K19\rafael\problemas> GeraApostaMegaSena.exe
Aposta 1: {18, 46, 55, 42, 22, 34}
Aposta 2: {26, 38, 10, 28, 25, 49}
Aposta 3: {2, 35, 59, 42, 45, 51}
Aposta 4: {22, 21, 7, 15, 34, 27}
Aposta 5: {11, 4, 17, 21, 32, 48}
Aposta 6: {32, 60, 7, 25, 40, 16}
Aposta 7: {40, 5, 39, 49, 43, 35}
Aposta 8: {16, 59, 15, 27, 13, 43}
Aposta 9: {40, 28, 7, 10, 29, 59}
Aposta 10: {10, 29, 20, 17, 43, 53}
```

Terminal A.18: Compilando e executando a classe GeraApostaMegaSena

34 Na pasta **problemas**, crie um arquivo chamado **Embaralha.cs**.

```
1  class Embaralha
2  {
3    static void Main()
4    {
5      System.Console.WriteLine("Original");
6      int[] array = {-10, 37, 101, 28, -4};
7      exibeArray(array);
8
9      System.Console.WriteLine("\nEmbaralhando...");
10     embaralha(array);
11     exibeArray(array);
12
13     System.Console.WriteLine("\nEmbaralhando...");
14     embaralha(array);
15     exibeArray(array);
16
17     System.Console.WriteLine("\nEmbaralhando...");
18     embaralha(array);
19     exibeArray(array);
20   }
21
22   static void embaralha(int[] array)
23   {
24     for(int i = 0; i < array.Length; i++)
25     {
26       int j = aleatorio(i, array.Length - 1);
27       troca(array, i, j);
28     }
```

```
29    }
30
31    static int aleatorio(int i, int j)
32    {
33      System.Random gerador = new System.Random();
34      return (int)(gerador.NextDouble() * (j - i + 1)) + i;
35    }
36
37    static void troca(int[] array, int i, int j)
38    {
39      int auxiliar = array[i];
40      array[i] = array[j];
41      array[j] = auxiliar;
42    }
43
44    static void exibeArray(int[] array)
45    {
46      System.Console.Write("{");
47
48      for(int i = 0; i < array.Length - 1; i++)
49      {
50        System.Console.Write(array[i] + ", ");
51      }
52
53      System.Console.WriteLine(array[array.Length - 1] + "}");
54    }
55  }
```

Código C# A.7: Embaralha.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao34.zip

35 Compile e execute a classe **Embaralha**.

```
C:\Users\K19\rafael\problemas> csc Embaralha.cs

C:\Users\K19\rafael\problemas> Embaralha.exe
Original
{-10, 37, 101, 28, -4}

Embaralhando...
{-4, 28, 101, 37, -10}

Embaralhando...
{37, 101, 28, -10, -4}

Embaralhando...
{-4, 101, 37, -10, 28}
```

Terminal A.19: Compilando e executando a classe `Embaralha`

36 Na pasta **problemas**, crie um arquivo chamado **Ordena.cs**.

```
1   class Ordena
2   {
3     static void Main()
4     {
5       System.Console.WriteLine("Original");
6       int[] array = {-10, 37, 101, 28, -4};
7       exibeArray(array);
8
9       System.Console.WriteLine("\nOrdenando...");
10      ordena(array);
11      exibeArray(array);
12    }
13
```

```
14    static void ordena(int[] array)
15    {
16      for(int i = 0; i < array.Length; i++)
17      {
18        int posicaoDoMenor = menor(array, i);
19        troca(array, i, posicaoDoMenor);
20      }
21    }
22
23    static int menor(int[] array, int inicio)
24    {
25      int menor = inicio;
26
27      for(int i = inicio + 1; i < array.Length; i++)
28      {
29        if(array[menor] > array[i])
30        {
31          menor = i;
32        }
33      }
34
35      return menor;
36    }
37
38    static void troca(int[] array, int i, int j)
39    {
40      int auxiliar = array[i];
41      array[i] = array[j];
42      array[j] = auxiliar;
43    }
44
45    static void exibeArray(int[] array)
46    {
47      System.Console.Write("{");
48
49      for(int i = 0; i < array.Length - 1; i++)
50      {
51        System.Console.Write(array[i] + ", ");
52      }
53
54      System.Console.WriteLine(array[array.Length - 1] + "}");
55    }
56  }
```

Código C# A.8: Ordena.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao36.zip

37 Compile e execute a classe **Ordena**.

```
C:\Users\K19\rafael\problemas> csc Ordena.cs

C:\Users\K19\rafael\problemas> java Ordena.exe
Original
{-10, 37, 101, 28, -4}

Ordenando...
{-10, -4, 28, 37, 101}
```

Terminal A.20: Compilando e executando a classe `Ordena`

38 Na pasta **problemas**, crie um arquivo chamado **Inverte.cs**.

```
1   class Inverte
2   {
```

```
3   static void Main()
4   {
5     System.Console.WriteLine("Original");
6     int[] array = {-10, 37, 101, 28, -4};
7     exibeArray(array);
8
9     System.Console.WriteLine("\nInvertendo...");
10    inverte(array);
11    exibeArray(array);
12  }
13
14  static void inverte(int[] array)
15  {
16    int i = 0;
17    int j = array.Length - 1;
18
19    while(i < j)
20    {
21      troca(array, i, j);
22      i++;
23      j--;
24    }
25  }
26
27  static void troca(int[] array, int i, int j)
28  {
29    int auxiliar = array[i];
30    array[i] = array[j];
31    array[j] = auxiliar;
32  }
33
34  static void exibeArray(int[] array)
35  {
36    System.Console.Write("{");
37
38    for(int i = 0; i < array.Length - 1; i++)
39    {
40      System.Console.Write(array[i] + ", ");
41    }
42
43    System.Console.WriteLine(array[array.Length - 1] + "}");
44  }
45}
```

Código C# A.9: Inverte.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao38.zip

39 Compile e execute a classe **Inverte**.

```
C:\Users\K19\rafael\problemas> csc Inverte.cs

C:\Users\K19\rafael\problemas> Inverte.exe
Original
{-10, 37, 101, 28, -4}

Invertendo...
{-4, 28, 101, 37, -10}
```

Terminal A.21: Compilando e executando a classe `Inverte`

40 Na pasta **problemas**, crie um arquivo chamado **Binario.cs**.

```
1  class Binario
2  {
```

```
3   static void Main()
4   {
5     for(int i = 1; i < 100; i++)
6     {
7       string s = binario(i);
8       System.Console.WriteLine(i + " em binário " + s);
9     }
10  }
11
12  static string binario(int x)
13  {
14    string y = "";
15    while(x > 0)
16    {
17      y = x % 2 + y;
18      x = x / 2;
19    }
20    return y;
21  }
22 }
```

Código C# A.10: Binario.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-problemas-fixacao40.zip

41 Compile e execute a classe **Binario**.

```
C:\Users\K19\rafael\problemas> csc Binario.cs

C:\Users\K19\rafael\problemas> Binario.exe
1 em binário 1
2 em binário 10
3 em binário 11
...
97 em binário 1100001
98 em binário 1100010
99 em binário 1100011
```

Terminal A.22: Compilando e executando a classe `Binario`

RESPOSTAS

APÊNDICE B

Exercício Complementar 1.1

Crie um arquivo chamado **DuasMensagens.java** na pasta **introducao**. Depois utilize o terminal para compilar e executar.

```
1 class DuasMensagens {
2   public static void main(String[] args) {
3     System.out.println("Hello World 1");
4     System.out.println("Hello World 2");
5   }
6 }
```

Código Java 1.19: DuasMensagens.java

```
K19/rafael/introducao$ javac DuasMensagens.java

K19/rafael/introducao$ java DuasMensagens
Hello World 1
Hello World 2
```

Terminal 1.27: Compilando e Executando

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-complementar1.zip

Exercício Complementar 1.2

Crie um arquivo chamado **DuasMensagens.cs** na pasta **introducao**. Depois utilize o terminal para compilar e executar.

```
1 class DuasMensagens
2 {
3   static void Main()
4   {
5     System.Console.WriteLine("Hello World 1");
6     System.Console.WriteLine("Hello World 2");
7   }
8 }
```

Código C# 1.16: DuasMensagens.cs

```
C:\Users\K19\rafael\introducao> csc HelloWorld2.java

C:\Users\K19\rafael\introducao> HelloWorld2.exe
Hello World 1
Hello World 2
```

Terminal 1.28: Compilando e Executando

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-complementar2.zip

Exercício Complementar 1.3

Crie um arquivo chamado **FrasePreferida.java** na pasta **introducao**. Depois utilize o terminal para compilar e executar.

```
1 class FrasePreferida {
2   public static void main(String[] args) {
3     System.out.println("Lorem ipsum dolor sit amet");
4   }
5 }
```

Código Java 1.20: FrasePreferida.java

```
K19/rafael/introducao$ javac FrasePreferida.java

K19/rafael/introducao$ java FrasePreferida
Lorem ipsum dolor sit amet
```

Terminal 1.29: Compilando e executando a classe `FrasePreferida`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-complementar3.zip

Exercício Complementar 1.4

Crie um arquivo chamado **FrasePreferida.cs** na pasta **introducao**. Depois utilize o terminal para compilar e executar.

```
1 class FrasePreferida
2 {
3   static void Main()
4   {
5     System.Console.WriteLine("Lorem ipsum dolor sit amet");
6   }
7 }
```

Código C# 1.17: FrasePreferida.cs

```
K19/rafael/introducao$ mcs FrasePreferida.cs

K19/rafael/introducao$ mono FrasePreferida.exe
Lorem ipsum dolor sit amet
```

Terminal 1.30: Compilando e executando a classe `FrasePreferida`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-complementar4.zip

Exercício Complementar 1.5

Crie um arquivo chamado **K19.java** na pasta **introducao**. Depois utilize o terminal para compilar e executar.

```
1 class K19 {
2   public static void main(String[] args) {
3     System.out.println("#    #   #    #####");
4     System.out.println("#   #   ##   #     #");
5     System.out.println("#  #   # #   #     #");
6     System.out.println("###      #    ######");
```

```
7       System.out.println("#  #      #          #");
8       System.out.println("#   #     #    #     #");
9       System.out.println("#    # #####   #####");
10    }
11  }
```

Código Java 1.21: K19.java

```
K19/rafael/introducao$ javac K19.java

K19/rafael/introducao$ java K19
```

Terminal 1.32: Compilando e executando a classe K19

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-complementar5.zip

Exercício Complementar 1.6

Crie um arquivo chamado **K19.cs** na pasta **introducao**. Depois utilize o terminal para compilar e executar.

```
1   class K19
2   {
3     static void Main()
4     {
5       System.Console.WriteLine("#    #   #    #####");
6       System.Console.WriteLine("#   #   ##   #     #");
7       System.Console.WriteLine("#  #   # #   #     #");
8       System.Console.WriteLine("###      #    ######");
9       System.Console.WriteLine("#  #     #         #");
10      System.Console.WriteLine("#   #    #   #     #");
11      System.Console.WriteLine("#    # #####  #####");
12    }
13  }
```

Código C# 1.18: K19.cs

```
K19/rafael/introducao$ mcs K19.cs

K19/rafael/introducao$ mono K19.exe
```

Terminal 1.34: Compilando e executando a classe K19

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-introducao-complementar6.zip

Questão 1.1

d

Questão 1.2

d

Questão 1.3

b

Questão 1.4

e

Questão 1.5

c

Questão 1.6

a

Questão 1.7

e

Questão 1.8

a

Questão 1.9

b

Questão 1.10

a

Questão 1.11

e

Questão 1.12

d

Questão 1.13

b

Questão 1.14

c

Questão 1.15

c

Exercício de Fixação 2.1

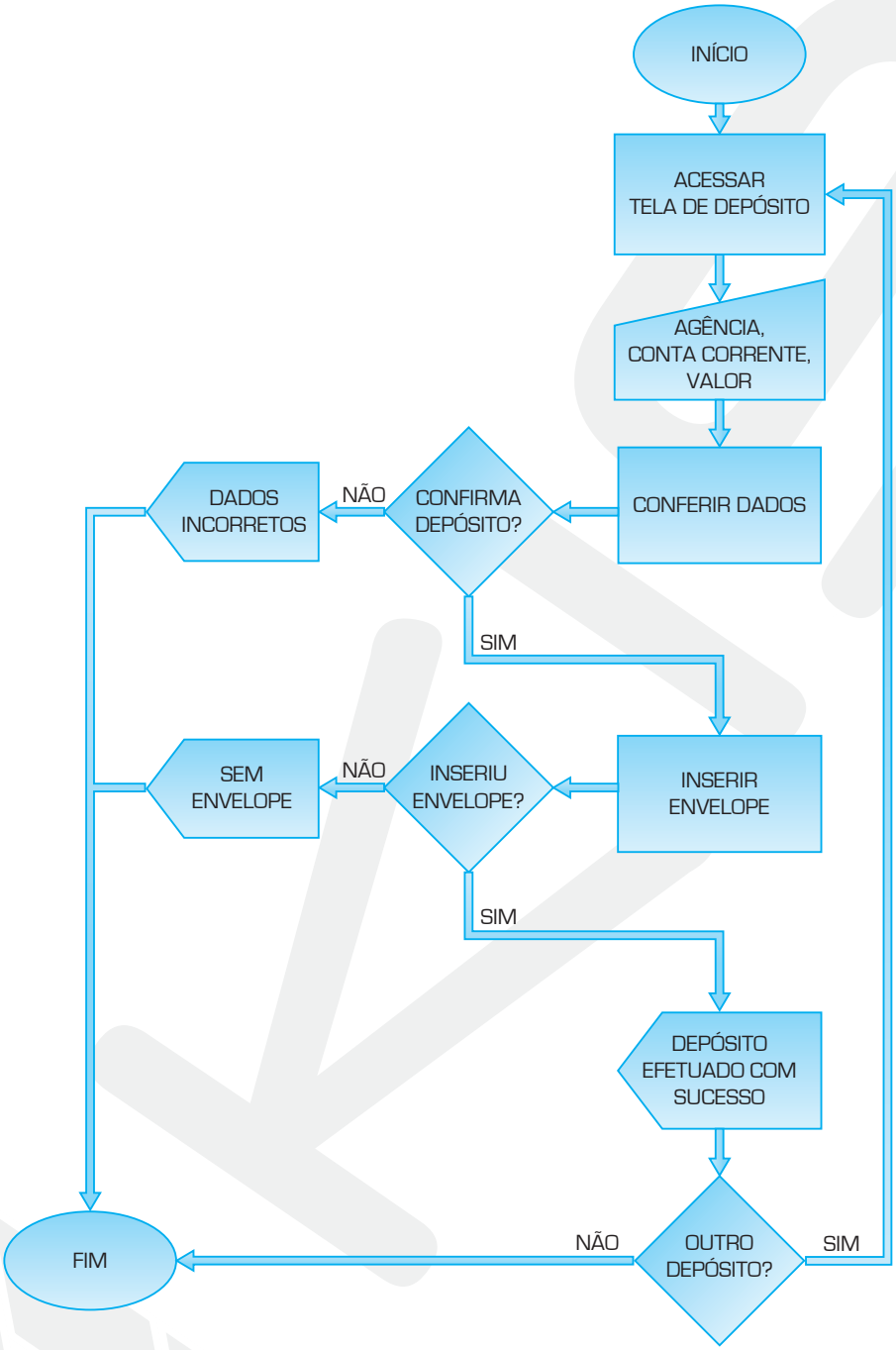

Figura 2.4: Resolução do exercício.

Exercício de Fixação 2.2

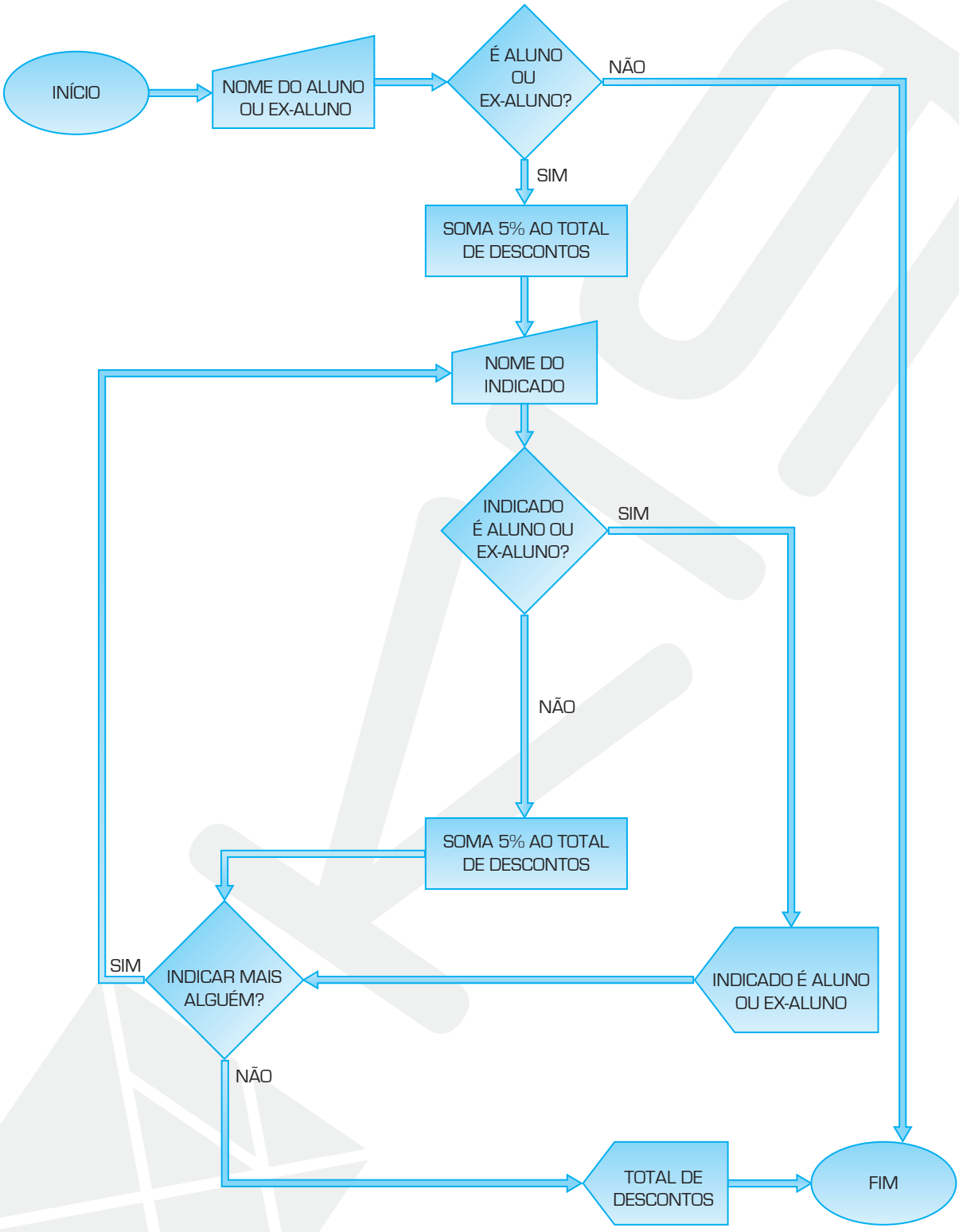

Figura 2.5: Resolução do exercício.

Desafio 2.1

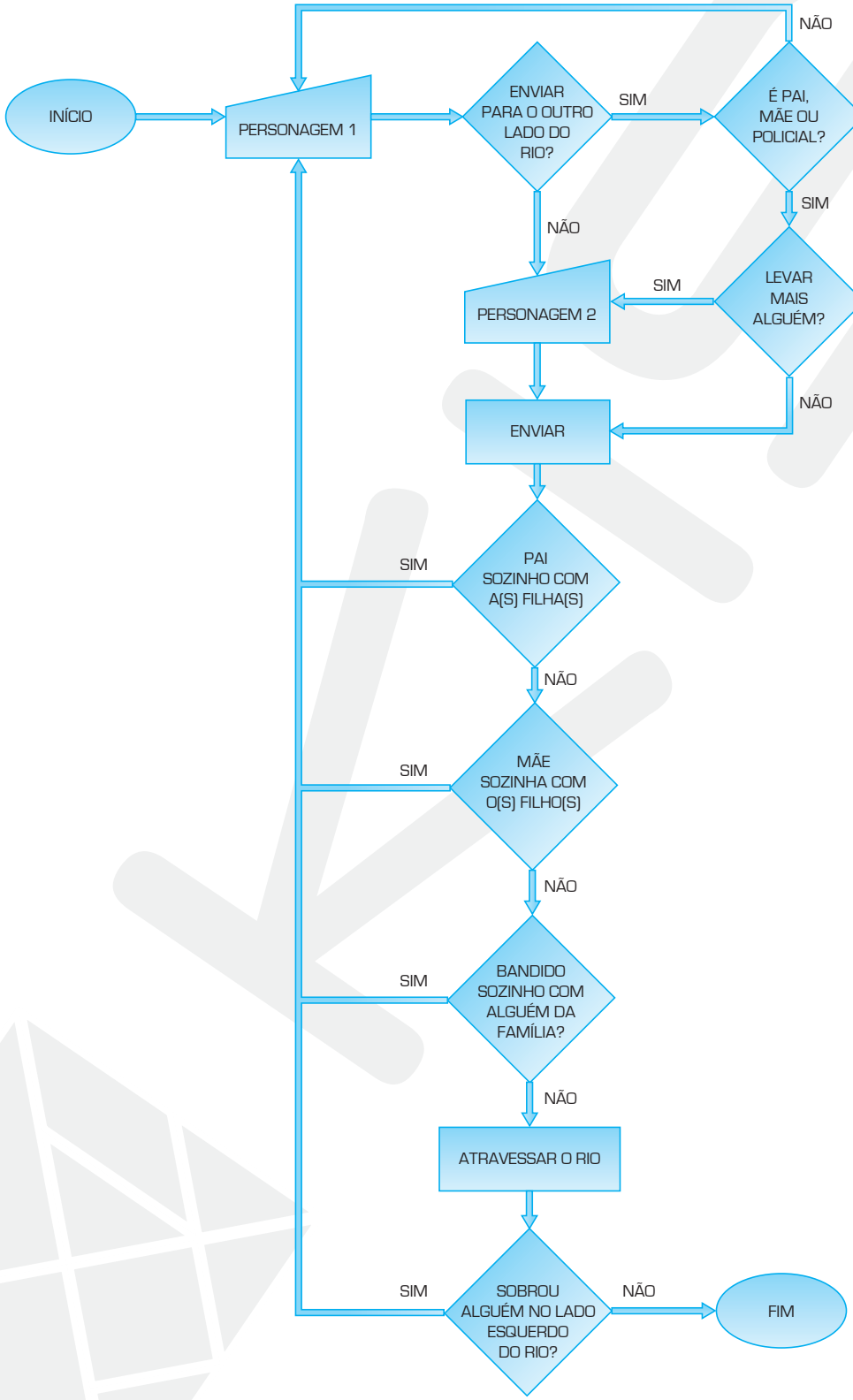

Figura 2.6: Resolução do desafio.

Exercício Complementar 3.1

Em Java:

1. “Bom dia” -> `String`
2. 3 -> `byte`, `short`, `int` ou `long`
3. 235.13 -> `double`
4. true -> `boolean`
5. -135 -> `short`, `int` ou `long`
6. 256.23F -> `float`
7. ‘A’ -> `char`
8. 6463275245745L -> `long`

Em C#:

1. “Bom dia” -> `string`
2. 3 -> `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long` ou `ulong`
3. 235.13 -> `double`
4. true -> `bool`
5. -135 -> `short`, `int` ou `long`
6. 256.23F -> `float`
7. ‘A’ -> `char`
8. 6463275245745L -> `long`

Exercício Complementar 3.2

```
1 class TestaVariavelPeso {
2   public static void main(String[] args) {
3     double peso;
4
5     peso = 88.20;
6
7     System.out.println(peso);
8   }
9 }
```

Código Java 3.51: TestaVariavelPeso.java

Compile e execute a classe **TestaVariavelPeso**

```
K19/rafael/variaveis$ javac TestaVariavelPeso.java

K19/rafael/variaveis$ java TestaVariavelPeso
88.20
```

Terminal 3.27: Compilando e executando a classe `TestaVariavelPeso`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar2.zip

Exercício Complementar 3.3

```
1  class TestaTiposBasicos {
2    public static void main(String[] args) {
3      byte b = 1;
4      short s = 2;
5      int i = 3;
6      long l = 4;
7      float f = 5.5F;
8      double d = 6.6;
9      char c = 'K';
10     boolean v = true;
11
12     System.out.println(b);
13     System.out.println(s);
14     System.out.println(i);
15     System.out.println(l);
16     System.out.println(f);
17     System.out.println(d);
18     System.out.println(c);
19     System.out.println(v);
20   }
21 }
```

Código Java 3.52: TestaTiposBasicos.java

Compile e execute a classe **TestaTiposBasicos**

```
K19/rafael/variaveis$ javac TestaTiposBasicos.java

K19/rafael/variaveis$ java TestaTiposBasicos
1
2
3
4
5.5
6.6
K
true
```

Terminal 3.28: Compilando e executando a classe `TestaTiposBasicos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar3.zip

Exercício Complementar 3.4

```
1  class TestaConversaoDouble {
```

```
2   public static void main(String[] args) {
3     String s = "1571.11";
4
5     double d = Double.parseDouble(s);
6
7     System.out.println(d);
8   }
9 }
```

Código Java 3.53: TestaConversaoDouble.java

Compile e execute a classe **TestaConversaoDouble**

```
K19/rafael/variaveis$ javac TestaConversaoDouble.java

K19/rafael/variaveis$ java TestaConversaoDouble
1571.11
```

Terminal 3.29: Compilando e executando a classe `TestaConversaoDouble`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar4.zip

Exercício Complementar 3.5

```
1  class TestaCalendar {
2    public static void main(String[] args) {
3      java.util.Calendar exatamenteAgora = java.util.Calendar.getInstance();
4      java.util.Calendar fundacaoK19 =
5        new java.util.GregorianCalendar(2010, 7, 27, 10, 32, 15);
6
7      java.text.SimpleDateFormat sdf =
8        new java.text.SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
9
10     String exatamenteAgoraFormatada = sdf.format(exatamenteAgora.getTime());
11     String fundacaoK19Formatada = sdf.format(fundacaoK19.getTime());
12
13     System.out.println(exatamenteAgoraFormatada);
14     System.out.println(fundacaoK19Formatada);
15   }
16 }
```

Código Java 3.54: TestaCalendar.java

Compile e execute a classe **TestaCalendar**

```
K19/rafael/variaveis$ javac TestaCalendar.java

K19/rafael/variaveis$ java TestaCalendar
03/08/2013 14:38:21
27/08/2010 10:32:15
```

Terminal 3.30: Compilando e executando a classe `TestaCalendar`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar5.zip

Exercício Complementar 3.6

```
1  class TestaVariavelPeso
2  {
3    static void Main()
4    {
5      double peso;
6
7      peso = 88.20;
8
9      System.Console.WriteLine(peso);
10   }
11 }
```

Código C# 3.41: TestaVariavelPeso.cs

```
C:\Users\K19\rafael\variaveis> csc TestaVariavelPeso.cs

C:\Users\K19\rafael\variaveis> TestaVariavelPeso.exe
88.2
```

Terminal 3.31: Compilando e executando a classe `TestaVariavelPeso`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar6.zip

Exercício Complementar 3.7

```
1  class TestaTiposBasicos
2  {
3    static void Main()
4    {
5      sbyte sb = 1;
6      byte  b = 2;
7      short s = 3;
8      ushort us = 4;
9      int i = 5;
10     uint ui = 6;
11     long l = 7;
12     ulong ul = 8;
13     float f = 5.5F;
14     double d1 = 6.6;
15     decimal d2 = 7.7M;
16     char c = 'K';
17     bool v = true;
18
19     System.Console.WriteLine(sb);
20     System.Console.WriteLine(b);
21     System.Console.WriteLine(s);
22     System.Console.WriteLine(us);
23     System.Console.WriteLine(i);
24     System.Console.WriteLine(ui);
25     System.Console.WriteLine(l);
26     System.Console.WriteLine(ul);
27     System.Console.WriteLine(f);
28     System.Console.WriteLine(d1);
29     System.Console.WriteLine(d2);
30     System.Console.WriteLine(c);
31     System.Console.WriteLine(v);
32   }
33 }
```

Código C# 3.42: TestaTiposBasicos.cs

Compile e execute a classe **TestaTiposBasicos**

```
C:\Users\K19\rafael\variaveis> csc TestaTiposBasicos.cs

C:\Users\K19\rafael\variaveis> TestaTiposBasicos
1
2
3
4
5
6
7
8
5.5
6.6
7.7
K
True
```

Terminal 3.32: Compilando e executando a classe `TestaTiposBasicos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar7.zip

Exercício Complementar 3.8

```
1  class TestaConversaoDouble
2  {
3    static void Main()
4    {
5      string s = "1571.11";
6
7      double d = System.Convert.ToDouble(s);
8
9      System.Console.WriteLine(d);
10   }
11 }
```

Código C# 3.43: TestaConversaoDouble.cs

Compile e execute a classe **TestaConversaoDouble**

```
C:\Users\K19\rafael\variaveis> csc TestaConversaoDouble.cs

C:\Users\K19\rafael\variaveis> TestaConversaoDouble.exe
1571.11
```

Terminal 3.33: Compilando e executando a classe `TestaConversaoDouble`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar8.zip

Exercício Complementar 3.9

```
1  class TestaDateTime
2  {
3    static void Main()
4    {
5      System.DateTime exatamenteAgora = System.DateTime.Now;
6      System.DateTime fundacaoK19 =
7        new System.DateTime(2010, 7, 27, 10, 32, 15);
8
```

```
9       string exatamenteAgoraFormatada = exatamenteAgora.ToString("dd/MM/yyyy HH:mm:ss")↩
            ;
10      string fundacaoK19Formatada = fundacaoK19.ToString("dd/MM/yyyy HH:mm:ss");
11
12      System.Console.WriteLine(exatamenteAgoraFormatada);
13      System.Console.WriteLine(fundacaoK19Formatada);
14    }
15  }
```

Código C# 3.44: TestaDateTime.cs

Compile e execute a classe **TestaDateTime**

```
C:\Users\K19\rafael\variaveis> csc TestaDateTime.cs

C:\Users\K19\rafael\variaveis> TestaDateTime.exe
03/08/2013 14:38:21
27/08/2010 10:32:15
```

Terminal 3.34: Compilando e executando a classe `TestaDateTime`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar9.zip

Exercício Complementar 3.10

Em Java:

```
1  class SistemaMercadorias {
2    public static void main(String[] args) {
3      int numeroDoPedido = 1523;
4      int codigoDoProduto = 845732;
5      short quantidade = 200;
6      double valorTotalDaCompra = 62373.5;
7
8      System.out.println(numeroDoPedido);
9      System.out.println(codigoDoProduto);
10     System.out.println(quantidade);
11     System.out.println(valorTotalDaCompra);
12   }
13 }
```

Código Java 3.55: SistemaMercadorias.java

Em C#:

```
1  class SistemaMercadorias
2  {
3    static void Main()
4    {
5      int numeroDoPedido = 1523;
6      int codigoDoProduto = 845732;
7      short quantidade = 200;
8      double valorTotalDaCompra = 62373.5;
9
10     System.Console.WriteLine(numeroDoPedido);
11     System.Console.WriteLine(codigoDoProduto);
12     System.Console.WriteLine(quantidade);
13     System.Console.WriteLine(valorTotalDaCompra);
14   }
15 }
```

Código C# 3.45: SistemaMercadorias.cs

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-variaveis-complementar10.zip

Desafio 3.1

Se estivéssemos trabalhando com uma loja bem pequena, com um baixo volume de vendas, assim como uma pequena variedade de produtos, poderíamos alterar as variáveis `numeroDoPedido` e `codigoDoProduto` para o tipo `short`. Dessa forma reduziríamos em 50% a quantidade de memória necessária para armazenarmos essas variáveis.

Caso estivéssemos trabalhando com uma grande rede de lojas, o tipo mais apropriado seria `long`. Consequentemente estaríamos aumentando em 50% a quantidade de memória necessária para armazenarmos essas variáveis.

Questão 3.1

d

Questão 3.2

a

Questão 3.3

c

Questão 3.4

b

Questão 3.5

a

Questão 3.6

e

Questão 3.7

e

Questão 3.8

a

Questão 3.9

b

Questão 3.10

c

Questão 3.11

a

Questão 3.12

c

Exercício Complementar 4.1

```
1  class UseOperadoresAritmeticos {
2    public static void main(String[] args) {
3      int x = 3 + 8;
4      int y = 7 - 3;
5      int z = 4 * 3;
6      int q = 8 / 2;
7      int w = 9 % 4;
8
9      System.out.println(x);
10     System.out.println(y);
11     System.out.println(z);
12     System.out.println(q);
13     System.out.println(w);
14   }
15 }
```

Código Java 4.53: UseOperadoresAritmeticos.java

Compile e execute a classe `UseOperadoresAritmeticos`

```
K19/rafael/operadores$ javac UseOperadoresAritmeticos.java

K19/rafael/operadores$ java UseOperadoresAritmeticos
11
4
12
4
1
```

Terminal 4.29: Compilando e executando a classe `UseOperadoresAritmeticos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar1.zip

Exercício Complementar 4.2

```
1  class IdadeMedia {
2    public static void main(String[] args) {
3      double idadeDoRafael = 27;
4      double idadeDoJonas = 29;
5      double idadeDoMarcelo = 27;
6
7      double idadeMedia = (idadeDoRafael + idadeDoJonas + idadeDoMarcelo)/3;
8
9      System.out.println("Idade Média: " + idadeMedia);
10   }
11 }
```

Código Java 4.54: IdadeMedia.java

Compile e execute a classe `IdadeMedia`

```
K19/rafael/operadores$ javac IdadeMedia.java

K19/rafael/operadores$ java IdadeMedia
Idade Média: 27.666666666666668
```

Terminal 4.30: Compilando e executando a classe `IdadeMedia`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar2.zip

Exercício Complementar 4.3

```
1  class UseDivisaoCasting {
2    public static void main(String[] args) {
3      int x = 41;
4      int y = 2;
5
6      System.out.println(x / y);
7      System.out.println((double)x / y);
8    }
9  }
```

Código Java 4.56: UseDivisaoCasting.java

Compile e execute a classe `UseDivisaoCasting`

```
K19/rafael/operadores$ javac UseDivisaoCasting.java

K19/rafael/operadores$ java UseDivisaoCasting
20
20.5
```

Terminal 4.31: Compilando e executando a classe `UseDivisaoCasting`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar3.zip

Exercício Complementar 4.4

```
1  class UseConcatenacao {
2    public static void main(String[] args) {
3      String s1 = "Rafael";
4      String s2 = "Jonas";
5      String s3 = "Marcelo";
6      String s4 = "Cosentino";
7      String s5 = "Hirata";
8      String s6 = "Martins";
9
10     System.out.println(s1 + " " + s4);
11     System.out.println(s2 + " " + s5);
12     System.out.println(s3 + " " + s6);
13   }
14 }
```

Código Java 4.58: UseConcatenacao.java

Compile e execute a classe `UseConcatenacao`

```
K19/rafael/operadores$ javac UseConcatenacao.java

K19/rafael/operadores$ java UseConcatenacao
Rafael Cosentino
Jonas Hirata
Marcelo Martins
```

Terminal 4.32: Compilando e executando a classe `UseConcatenacao`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar4.zip

Exercício Complementar 4.5

```
1  class UseOperadoresAtribuicao {
2    public static void main(String[] args) {
3      int x = 5;
4      System.out.println(x);
5      x += 10;
6      System.out.println(x);
7      x -= 3;
8      System.out.println(x);
9      x *= 4;
10     System.out.println(x);
11     x /= 8;
12     System.out.println(x);
13     x %= 5;
14     System.out.println(x);
15     x++;
16     System.out.println(x);
17     x--;
18     System.out.println(x);
19   }
20 }
```

Código Java 4.60: UseOperadoresAtribuicao.java

Compile e execute a classe `UseOperadoresAtribuicao`

```
K19/rafael/operadores$ javac UseOperadoresAtribuicao.java

K19/rafael/operadores$ java UseOperadoresAtribuicao
5
15
12
48
6
1
2
1
```

Terminal 4.33: Compilando e executando a classe `UseOperadoresAtribuicao`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar5.zip

Exercício Complementar 4.6

```
1  class NumeroTelefone {
2    public static void main(String [] args) {
3      int x = 23;
4      x += 8;
5      System.out.println(x);
6      x -= 7;
7      System.out.println(x);
8      x /= 3;
9      System.out.println(x);
10     x *= 7;
11     System.out.println(x);
12     x %= 9;
13     System.out.println(x);
14     x += 1;
15     System.out.println(x);
16   }
17 }
```

Código Java 4.61: NumeroTelefone.java

Compile e execute a classe `NumeroTelefone`

```
K19/rafael/operadores$ javac NumeroTelefone.java

K19/rafael/operadores$ java NumeroTelefone
31
24
8
56
2
3
```

Terminal 4.34: Compilando e executando a classe `NumeroTelefone`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar6.zip

Exercício Complementar 4.7

```
1  class UseOperadoresRelacionais {
2    public static void main(String[] args) {
```

```
3       int x = 20;
4       int y = 15;
5
6       System.out.println (x > y);
7       System.out.println (x >= y);
8       System.out.println (x < y);
9       System.out.println (x <= y);
10      System.out.println (x == y);
11      System.out.println (x != y);
12    }
13  }
```

Código Java 4.63: UseOperadoresRelacionais.java

Compile e execute a classe `UseOperadoresRelacionais`

```
K19/rafael/operadores$ javac UseOperadoresRelacionais.java

K19/rafael/operadores$ java UseOperadoresRelacionais
true
true
false
false
false
true
```

Terminal 4.35: Compilando e executando a classe `UseOperadoresRelacionais`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar7.zip

Exercício Complementar 4.8

```
1   class VerificaValores {
2     public static void main(String [] args) {
3       int anoTorreGemeas = 2001;
4       int anoMichaelJackson = 2009;
5
6       System.out.println((anoTorreGemeas+19)/4 >= (anoMichaelJackson+129)/5);
7     }
8   }
```

Código Java 4.64: VerificaValores.java

Compile e execute a classe `VerificaValores`

```
K19/rafael/operadores$ javac VerificaValores.java

K19/rafael/operadores$ java VerificaValores
true
```

Terminal 4.36: Compilando e executando a classe `VerificaValores`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar8.zip

Exercício Complementar 4.9

```
class UseOperadoresLogicos {
  public static void main(String[] args) {
    int q = 10;
    int w = 5;
    int e = 8;
    int r = 11;

    System.out.println(q > w | e < r);
    System.out.println(q > r || e < w);
    System.out.println(q > e & w < r);
    System.out.println(q > w && r < e);
    System.out.println(q > w ^ e < r);
  }
}
```

Código Java 4.66: UseOperadoresLogicos.java

Compile e execute a classe `UseOperadoresLogicos`

```
K19/rafael/operadores$ javac UseOperadoresLogicos.java

K19/rafael/operadores$ java UseOperadoresLogicos
true
false
true
false
false
```

Terminal 4.37: Compilando e executando a classe `UseOperadoresLogicos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar9.zip

Exercício Complementar 4.10

```
class UseTernarioNegacaoIncrementoDecremento {
  public static void main(String[] args) {
    int a = 10;
    int b = 8;

    System.out.println((a < b) ? a : b);
    System.out.println(!(a < b) ? "Marcelo" : "Jonas");
    System.out.println((a < b) ? a : ++b);
    System.out.println(!(--a == b) ? a : b + 1);
  }
}
```

Código Java 4.68: UseTernarioNegacaoIncrementoDecremento.java

Compile e execute a classe `UseTernarioNegacaoIncrementoDecremento`

```
K19/rafael/operadores$ javac UseTernarioNegacaoIncrementoDecremento.java

K19/rafael/operadores$ java UseTernarioNegacaoIncrementoDecremento
8
Marcelo
9
10
```

Terminal 4.38: Compilando e executando a classe `UseTernarioNegacaoIncrementoDecremento`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar10.zip

Exercício Complementar 4.11

```
class GeraParcelas {
  public static void main(String[] args) {
    java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat("dd/MM/yyyy");

    java.util.Calendar p = new java.util.GregorianCalendar(2015 , 7 , 15) ;
    System.out.println(sdf.format(p.getTime()));

    p.add(java.util.Calendar.DAY_OF_MONTH, 30);
    System.out.println(sdf.format(p.getTime()));

    p.add(java.util.Calendar.DAY_OF_MONTH, 30);
    System.out.println(sdf.format(p.getTime()));

    p.add(java.util.Calendar.DAY_OF_MONTH, 30);
    System.out.println(sdf.format(p.getTime()));
  }
}
```

Código Java 4.69: GeraParcelas.java

Compile e execute a classe `GeraParcelas`

```
K19/rafael/operadores$ javac GeraParcelas.java

K19/rafael/operadores$ java GeraParcelas
15/08/2015
14/09/2015
14/10/2015
13/11/2015
```

Terminal 4.39: Compilando e executando a classe `GeraParcelas`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar11.zip

Exercício Complementar 4.12

```
class UseOperadoresAritmeticos
{
  static void Main()
  {
    int x = 3 + 8;
    int y = 7 - 3;
    int z = 4 * 3;
    int q = 8 / 2;
    int w = 9 % 4;

    System.Console.WriteLine(x);
    System.Console.WriteLine(y);
    System.Console.WriteLine(z);
    System.Console.WriteLine(q);
    System.Console.WriteLine(w);
  }
}
```

Código C# 4.43: UseOperadoresAritmeticos.cs

Compile e execute a classe `UseOperadoresAritmeticos`

```
C:\Users\K19\rafael\operadores> csc UseOperadoresAritmeticos.cs

C:\Users\K19\rafael\operadores> UseOperadoresAritmeticos.exe
11
4
12
4
1
```

Terminal 4.40: Compilando e executando a classe `UseOperadoresAritmeticos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar12.zip

Exercício Complementar 4.13

```
1  class IdadeMedia
2  {
3    static void Main()
4    {
5      double idadeDoRafael = 27;
6      double idadeDoJonas = 29;
7      double idadeDoMarcelo = 27;
8
9      double idadeMedia = (idadeDoRafael + idadeDoJonas + idadeDoMarcelo)/3;
10
11     System.Console.WriteLine("Idade Média: " + idadeMedia);
12   }
13 }
```

Código C# 4.44: IdadeMedia.cs

Compile e execute a classe `IdadeMedia`

```
C:\Users\K19\rafael\operadores> csc IdadeMedia.cs

C:\Users\K19\rafael\operadores> IdadeMedia.exe
Idade Média: 27.6666666666667
```

Terminal 4.41: Compilando e executando a classe `IdadeMedia`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar13.zip

Exercício Complementar 4.14

```
1  class UseDivisaoCasting
2  {
3    static void Main()
4    {
5      int x = 41;
6      int y = 2;
7
8      System.Console.WriteLine(x / y);
9      System.Console.WriteLine((double)x / y);
10   }
11 }
```

Código C# 4.46: UseDivisaoCasting.cs

Compile e execute a classe `UseDivisaoCasting`

```
C:\Users\K19\rafael\operadores> csc UseDivisaoCasting.cs

C:\Users\K19\rafael\operadores> UseDivisaoCasting.exe
20
20.5
```

Terminal 4.42: Compilando e executando a classe `UseDivisaoCasting`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar14.zip

Exercício Complementar 4.15

```
1  class UseConcatenacao
2  {
3    static void Main()
4    {
5      string s1 = "Rafael";
6      string s2 = "Jonas";
7      string s3 = "Marcelo";
8      string s4 = "Cosentino";
9      string s5 = "Hirata";
10     string s6 = "Martins";
11
12     System.Console.WriteLine(s1 + " " + s4);
13     System.Console.WriteLine(s2 + " " + s5);
14     System.Console.WriteLine(s3 + " " + s6);
15   }
16 }
```

Código Java 4.70: UseConcatenacao.java

Compile e execute a classe `UseConcatenacao`

```
C:\Users\K19\rafael\operadores> csc UseConcatenacao.cs

C:\Users\K19\rafael\operadores> UseConcatenacao.exe
Rafael Cosentino
Jonas Hirata
Marcelo Martins
```

Terminal 4.43: Compilando e executando a classe `UseConcatenacao`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar15.zip

Exercício Complementar 4.16

```
1  class UseOperadoresAtribuicao
2  {
3    static void Main()
4    {
5      int x = 5;
6      System.Console.WriteLine(x);
7      x += 10;
8      System.Console.WriteLine(x);
9      x -= 3;
```

```
10      System.Console.WriteLine(x);
11      x *= 4;
12      System.Console.WriteLine(x);
13      x /= 8;
14      System.Console.WriteLine(x);
15      x %= 5;
16      System.Console.WriteLine(x);
17      x++;
18      System.Console.WriteLine(x);
19      x--;
20      System.Console.WriteLine(x);
21    }
22  }
```

Código C# 4.49: UseOperadoresAtribuicao.cs

Compile e execute a classe `UseOperadoresAtribuicao`

```
C:\Users\K19\rafael\operadores> csc UseOperadoresAtribuicao.cs

C:\Users\K19\rafael\operadores> UseOperadoresAtribuicao.exe
5
15
12
48
6
1
2
1
```

Terminal 4.44: Compilando e executando a classe `UseOperadoresAtribuicao`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar16.zip

Exercício Complementar 4.17

```
1  class NumeroTelefone
2  {
3    static void Main()
4    {
5      int x = 23;
6      x += 8;
7      System.Console.WriteLine(x);
8      x -= 7;
9      System.Console.WriteLine(x);
10     x /= 3;
11     System.Console.WriteLine(x);
12     x *= 7;
13     System.Console.WriteLine(x);
14     x %= 9;
15     System.Console.WriteLine(x);
16     x += 1;
17     System.Console.WriteLine(x);
18   }
19 }
```

Código C# 4.50: NumeroTelefone.cs

Compile e execute a classe `NumeroTelefone`

```
C:\Users\K19\rafael\operadores> csc NumeroTelefone.cs

C:\Users\K19\rafael\operadores> NumeroTelefone.exe
```

```
31
24
8
56
2
3
```

Terminal 4.45: Compilando e executando a classe `NumeroTelefone`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar17.zip

Exercício Complementar 4.18

```
1  class UseOperadoresRelacionais
2  {
3    static void Main()
4    {
5      int x = 20;
6      int y = 15;
7
8      System.Console.WriteLine(x > y);
9      System.Console.WriteLine(x >= y);
10     System.Console.WriteLine(x < y);
11     System.Console.WriteLine(x <= y);
12     System.Console.WriteLine(x == y);
13     System.Console.WriteLine(x != y);
14   }
15 }
```

Código C# 4.52: UseOperadoresRelacionais.cs

Compile e execute a classe `UseOperadoresRelacionais`

```
C:\Users\K19\rafael\operadores> csc UseOperadoresRelacionais.cs

C:\Users\K19\rafael\operadores> UseOperadoresRelacionais.exe
True
True
False
False
False
True
```

Terminal 4.46: Compilando e executando a classe `UseOperadoresRelacionais`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar18.zip

Exercício Complementar 4.19

```
1  class VerificaValores
2  {
3    static void Main()
4    {
5      int anoTorreGemeas = 2001;
6      int anoMichaelJackson = 2009;
7
8      System.Console.WriteLine((anoTorreGemeas+19)/4 >= (anoMichaelJackson+129)/5);
9    }
```

```
10 }
```

Código C# 4.53: VerificaValores.cs

Compile e execute a classe `VerificaValores`

```
C:\Users\K19\rafael\operadores> csc VerificaValores.cs

C:\Users\K19\rafael\operadores> VerificaValores.exe
True
```

Terminal 4.47: Compilando e executando a classe `VerificaValores`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar19.zip

Exercício Complementar 4.20

```
1  class UseOperadoresLogicos
2  {
3    static void Main()
4    {
5      int q = 10;
6      int w = 5;
7      int e = 8;
8      int r = 11;
9
10     System.Console.WriteLine(q > w | e < r);
11     System.Console.WriteLine(q > r || e < w);
12     System.Console.WriteLine(q > e & w < r);
13     System.Console.WriteLine(q > w && r < e);
14     System.Console.WriteLine(q > w ^ e < r);
15   }
16 }
```

Código C# 4.55: UseOperadoresLogicos.cs

Compile e execute a classe `UseOperadoresLogicos`

```
C:\Users\K19\rafael\operadores> csc UseOperadoresLogicos.cs

C:\Users\K19\rafael\operadores> UseOperadoresLogicos.exe
True
False
True
False
False
```

Terminal 4.48: Compilando e executando a classe `UseOperadoresLogicos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar20.zip

Exercício Complementar 4.21

```
1  class UseTernarioNegacaoIncrementoDecremento
2  {
3    static void Main()
```

```
4   {
5     int a = 10;
6     int b = 8;
7
8     System.Console.WriteLine((a < b) ? a : b);
9     System.Console.WriteLine(!(a < b) ? "Marcelo" : "Jonas");
10    System.Console.WriteLine((a < b) ? a : ++b);
11    System.Console.WriteLine(!(--a == b) ? a : b + 1);
12  }
13 }
```

Código C# 4.57: UseTernarioNegacaoIncrementoDecremento.cs

Compile e execute a classe `UseTernarioNegacaoIncrementoDecremento`

```
C:\Users\K19\rafael\operadores> csc UseTernarioNegacaoIncrementoDecremento.cs

C:\Users\K19\rafael\operadores> UseTernarioNegacaoIncrementoDecremento.exe
8
Marcelo
9
10
```

Terminal 4.49: Compilando e executando a classe `UseTernarioNegacaoIncrementoDecremento`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar21.zip

Exercício Complementar 4.22

```
1  class GeraParcelas
2  {
3    static void Main()
4    {
5      System.DateTime p = new System.DateTime(2015 , 8 , 15);
6      System.Console.WriteLine(p.ToString("dd/MM/yyyy"));
7
8      p = p.AddDays(30);
9      System.Console.WriteLine(p.ToString("dd/MM/yyyy"));
10
11     p = p.AddDays(30);
12     System.Console.WriteLine(p.ToString("dd/MM/yyyy"));
13
14     p = p.AddDays(30);
15     System.Console.WriteLine(p.ToString("dd/MM/yyyy"));
16   }
17 }
```

Código C# 4.58: GeraParcelas.cs

Compile e execute a classe `GeraParcelas`

```
C:\Users\K19\rafael\operadores> csc GeraParcelas.cs

C:\Users\K19\rafael\operadores> GeraParcelas.exe
15/08/2015
14/09/2015
14/10/2015
13/11/2015
```

Terminal 4.50: Compilando e executando a classe `GeraParcelas`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-operadores-complementar22.zip

Questão 4.1

d

Questão 4.2

c

Questão 4.3

e

Questão 4.4

b

Questão 4.5

a

Questão 4.6

e

Questão 4.7

d

Questão 4.8

b

Questão 4.9

e

Questão 4.10

c

Questão 4.11

a

Exercício Complementar 5.1

```
1  class ComparaValores {
2    public static void main(String[] args) {
3      double primeiro = Math.random();
4      double segundo = Math.random();
5
6      System.out.println("Primeiro: " + primeiro);
7      System.out.println("Segundo: " + segundo);
8
9      if(primeiro < segundo) {
10       System.out.println("Primeiro > Segundo");
11     } else if(primeiro > segundo) {
12       System.out.println("Segundo > Primeiro");
13     } else {
14       System.out.println("Primeiro = Segundo");
15     }
16   }
17 }
```

Código Java 5.119: ComparaValores.java

Compile e execute a classe `ComparaValores`

```
K19/rafael/controle-de-fluxo$ javac ComparaValores.java

K19/rafael/controle-de-fluxo$ java ComparaValores
Primeiro: 0.129763492443489
Segundo: 0.00506741041553552
Primeiro > Segundo

K19/rafael/controle-de-fluxo$ java ComparaValores
Primeiro: 0.16979575246475298
Segundo: 0.7230291214946017
Segundo > Primeiro

K19/rafael/controle-de-fluxo$ java ComparaValores
Primeiro: 0.8885443635299185
Segundo: 0.8885443635299185
Primeiro = Segundo
```

Terminal 5.91: Compilando e executando a classe `ComparaValores`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar1.zip

Exercício Complementar 5.2

```
1  class BlocoDeAstericos {
2    public static void main(String[] args) {
3      for(int i = 0; i < 5; i++) {
4        System.out.println("******************");
5      }
6    }
7  }
```

Código Java 5.121: BlocoDeAstericos.java

Compile e execute a classe `BlocoDeAstericos`

```
K19/rafael/controle-de-fluxo$ javac BlocoDeAstericos.java

K19/rafael/controle-de-fluxo$ java BlocoDeAstericos
**********
**********
**********
**********
**********
```

Terminal 5.92: Compilando e executando a classe `BlocoDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar2.zip

Exercício Complementar 5.3

```
1 class TrianguloDeAstericos {
2   public static void main(String[] args) {
3     String s = "*";
4     for(int i = 0; i < 6; i++) {
5       System.out.println(s);
6       s += "*";
7     }
8   }
9 }
```

Código Java 5.123: TrianguloDeAstericos.java

Compile e execute a classe `TrianguloDeAstericos`

```
K19/rafael/controle-de-fluxo$ javac TrianguloDeAstericos.java

K19/rafael/controle-de-fluxo$ java TrianguloDeAstericos
*
**
***
****
*****
******
```

Terminal 5.93: Compilando e executando a classe `TrianguloDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar3.zip

Exercício Complementar 5.4

```
1  class TresTriangulosDeAstericos {
2    public static void main(String[] args) {
3      for(int i = 0; i < 3; i++) {
4        String s = "*";
5        for(int j = 0; j < 6; j++) {
6          System.out.println(s);
7          s += "*";
8        }
9      }
10   }
11 }
```

Código Java 5.125: TresTriangulosDeAstericos.java

Compile e execute a classe `TresTriangulosDeAstericos`

```
K19/rafael/controle-de-fluxo$ javac TresTriangulosDeAstericos.java

K19/rafael/controle-de-fluxo$ java TresTriangulosDeAstericos
*
**
***
****
*****
******
*
**
***
****
*****
******
*
**
***
****
*****
******
```

Terminal 5.94: Compilando e executando a classe `TresTriangulosDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar4.zip

Exercício Complementar 5.5

```
1  class LosangoDeAstericos {
2    public static void main(String[] args) {
3      String s1 = "";
4      String s2 = "**********";
5      for(int i = 0; i < 5; i++) {
6        System.out.print(s1);
7        System.out.println(s2);
8        s1 += " ";
9      }
10   }
11 }
```

Código Java 5.127: LosangoDeAstericos.java

Compile e execute a classe `LosangoDeAstericos`

```
K19/rafael/controle-de-fluxo$ javac LosangoDeAstericos.java

K19/rafael/controle-de-fluxo$ java LosangoDeAstericos
**********
 **********
  **********
   **********
    **********
```

Terminal 5.95: Compilando e executando a classe `LosangoDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar5.zip

Exercício Complementar 5.6

```
class TresLosangosDeAstericos {
  public static void main(String[] args) {
    for(int i = 0; i < 3; i++) {
      String s1 = "";
      String s2 = "**********";
      for(int j = 0; j < 5; j++) {
        System.out.print(s1);
        System.out.println(s2);
        s1 += " ";
      }
    }
  }
}
```

Código Java 5.129: TresLosangosDeAstericos.java

Compile e execute a classe `TresLosangosDeAstericos`

```
K19/rafael/controle-de-fluxo$ javac TresLosangosDeAstericos.java

K19/rafael/controle-de-fluxo$ java TresLosangosDeAstericos
**********
 **********
  **********
   **********
    **********
**********
 **********
  **********
   **********
    **********
**********
 **********
  **********
   **********
    **********
```

Terminal 5.96: Compilando e executando a classe `TresLosangosDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar6.zip

Exercício Complementar 5.7

```
class CartoesDeEstacionamento {
  public static void main(String[] args) {
    for(int i = 1; i <= 3; i++) {
      for(int j = 1; j <= 9; j++) {
        for(int k = 1; k <= 4; k++) {
          System.out.println("BLOCO: " + i + " APTO: " + (j * 10 + k));
        }
      }
    }
  }
}
```

Código Java 5.131: CartoesDeEstacionamento.java

Compile e execute a classe `CartoesDeEstacionamento`

```
K19/rafael/controle-de-fluxo$ javac CartoesDeEstacionamento.java

K19/rafael/controle-de-fluxo$ java CartoesDeEstacionamento
BLOCO: 1 APTO: 11
BLOCO: 1 APTO: 12
BLOCO: 1 APTO: 13
BLOCO: 1 APTO: 14
BLOCO: 1 APTO: 21
BLOCO: 1 APTO: 22
BLOCO: 1 APTO: 23
BLOCO: 1 APTO: 24
...
```

Terminal 5.97: Compilando e executando a classe `CartoesDeEstacionamento`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar7.zip

Exercício Complementar 5.8

```
1 class Tabuada {
2   public static void main(String[] args) {
3     for (int i = 1; i <= 10; i++) {
4       for (int j = 1; j <= 10; j++) {
5         System.out.println(i + "x" + j + " = " + i * j);
6       }
7     }
8   }
9 }
```

Código Java 5.133: Tabuada.java

Compile e execute a classe Tabuada

```
K19/rafael/controle-de-fluxo$ javac Tabuada.java

K19/rafael/controle-de-fluxo$ java Tabuada
1x1 = 1
1x2 = 2
1x3 = 3
...
10x8 = 80
10x9 = 90
10x10 = 100
```

Terminal 5.98: Compilando e executando a classe Tabuada

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar8.zip

Exercício Complementar 5.9

```
1 class Piramide {
2   public static void main(String[] args) {
3     int baseMaior = 7;
4
5     for (int i = 1; i <= baseMaior; i += 2) {
6       int espacos = (baseMaior - i) / 2;
7       String linha = "";
8
9       for (int j = 0; j < espacos; j++) {
```

```
10          linha += " ";
11        }
12
13        for (int k = 0; k < i; k++) {
14          linha += "*";
15        }
16
17        System.out.println(linha);
18      }
19    }
20  }
```

Código Java 5.135: Piramide.java

Compile e execute a classe `Piramide`

```
K19/rafael/controle-de-fluxo$ javac Piramide.java

K19/rafael/controle-de-fluxo$ java Piramide
   *
  ***
 *****
*******
```

Terminal 5.99: Compilando e executando a classe `Piramide`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar9.zip

Exercício Complementar 5.10

```
1  class ArvoreNatal {
2    public static void main(String[] args) {
3      int baseMaior = 15;
4
5      for (int m = 7; m <= 15; m += 4) {
6
7        for (int i = m - 6; i <= m; i += 2) {
8          int espacos = (baseMaior - i) / 2;
9          String linha = "";
10
11         for (int j = 0; j < espacos; j++) {
12           linha += " ";
13         }
14
15         for (int k = 0; k < i; k++) {
16           linha += "*";
17         }
18
19         System.out.println(linha);
20       }
21     }
22   }
23 }
```

Código Java 5.137: ArvoreNatal.java

Compile e execute a classe `ArvoreNatal`

```
K19/rafael/controle-de-fluxo$ javac ArvoreNatal.java

K19/rafael/controle-de-fluxo$ java ArvoreNatal
       *
```

```
      ***
     *****
    *******
     *****
    *******
   *********
  ***********
   *********
  ***********
 *************
***************
```

Terminal 5.100: Compilando e executando a classe `ArvoreNatal`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar10.zip

Exercício Complementar 5.11

```
1  class ContaUns {
2    public static void main(String[] args) {
3      double numero = Math.random();
4      System.out.println(numero);
5
6      String s = "" + numero;
7      int resposta = 0;
8
9      for(int i = 0; i < s.length(); i++) {
10       if(s.charAt(i) == '1') {
11         resposta++;
12       }
13     }
14
15     System.out.println(resposta);
16   }
17 }
```

Código Java 5.139: ContaUns.java

Compile e execute a classe `ContaUns`

```
K19/rafael/controle-de-fluxo$ javac ContaUns.java

K19/rafael/controle-de-fluxo$ java ContaUns
0.46029982416581405
2

K19/rafael/controle-de-fluxo$ java ContaUns
0.2402150675471108
4

K19/rafael/controle-de-fluxo$ java ContaUns
0.9189156985811225
4
```

Terminal 5.101: Compilando e executando a classe `ContaUns`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar11.zip

Exercício Complementar 5.12

```
class JogoDaSomaImpar {
  public static void main(String[] args) {
    int soma = 0;
    int quantidadeDeSeis = 0;

    for(int i = 0; i < 10; i++) {
      int numero = (int)(Math.random() * 6 + 1);

      System.out.println("Número: " + numero);

      if(numero == 1) {
        continue;
      }

      if(numero == 6) {
        quantidadeDeSeis++;
      }

      if(quantidadeDeSeis == 2) {
        System.out.println("Dois seis! Você perdeu!");
        break;
      }

      soma += numero;
    }

    if(quantidadeDeSeis != 2) {
      System.out.println("Soma: " + soma);
      if(soma % 2 != 0) {
        System.out.println("Soma ímpar! Você ganhou");
      } else {
        System.out.println("Soma par! Você perdeu");
      }
    }
  }
}
```

Código Java 5.141: JogoDaSomaImpar.java

Compile e execute a classe `JogoDaSomaImpar`

```
K19/rafael/controle-de-fluxo$ javac JogoDaSomaImpar.java

K19/rafael/controle-de-fluxo$ java JogoDaSomaImpar
Número: 5
Número: 3
Número: 2
Número: 5
Número: 2
Número: 6
Número: 5
Número: 6
Dois seis! Você perdeu!

K19/rafael/controle-de-fluxo$ java JogoDaSomaImpar
Número: 1
Número: 2
Número: 2
Número: 1
Número: 6
Número: 1
Número: 4
Número: 3
Número: 2
Número: 2
Soma: 21
Soma ímpar! Você ganhou

K19/rafael/controle-de-fluxo$ java JogoDaSomaImpar
Número: 4
Número: 4
Número: 2
Número: 4
```

```
Número: 3
Número: 5
Número: 5
Número: 3
Número: 6
Número: 2
Soma: 38
Soma par! Você perdeu
```

Terminal 5.102: Compilando e executando a classe `JogoDaSomaImpar`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar12.zip

Exercício Complementar 5.13

```
1  class ComparaValores
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6
7      double primeiro = gerador.NextDouble();
8      double segundo = gerador.NextDouble();
9
10     System.Console.WriteLine("Primeiro: " + primeiro);
11     System.Console.WriteLine("Segundo: " + segundo);
12
13     if(primeiro > segundo)
14     {
15       System.Console.WriteLine("Primeiro > Segundo");
16     }
17     else if(primeiro < segundo)
18     {
19       System.Console.WriteLine("Segundo > Primeiro");
20     }
21     else
22     {
23       System.Console.WriteLine("Primeiro = Segundo");
24     }
25   }
26 }
```

Código C# 5.21: ComparaValores.cs

Compile e execute a classe `ComparaValores`

```
C:\Users\K19\rafael\controle-de-fluxo> csc ComparaValores.cs

C:\Users\K19\rafael\controle-de-fluxo> ComparaValores.exe
Primeiro: 0.129763492443489
Segundo: 0.00506741041553552
Primeiro > Segundo

C:\Users\K19\rafael\controle-de-fluxo> ComparaValores.exe
Primeiro: 0.363589867653134
Segundo: 0.388261379389214
Segundo > Primeiro

C:\Users\K19\rafael\controle-de-fluxo> ComparaValores.exe
Primeiro: 0.661000854177867
Segundo: 0.661000854177867
Primeiro = Segundo
```

Terminal 5.103: Compilando e executando a classe `ComparaValores`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar13.zip

Exercício Complementar 5.14

```
1  class BlocoDeAstericos
2  {
3    static void Main()
4    {
5      for(int i = 0; i < 5; i++)
6      {
7        System.Console.WriteLine("*****************");
8      }
9    }
10 }
```

Código C# 5.23: BlocoDeAstericos.cs

Compile e execute a classe `BlocoDeAstericos`

```
C:\Users\K19\rafael\controle-de-fluxo> csc BlocoDeAstericos.cs

C:\Users\K19\rafael\controle-de-fluxo> BlocoDeAstericos.exe
*****************
*****************
*****************
*****************
*****************
```

Terminal 5.104: Compilando e executando a classe `BlocoDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar14.zip

Exercício Complementar 5.15

```
1  class TrianguloDeAstericos
2  {
3    static void Main()
4    {
5      string s = "*";
6      for(int i = 0; i < 6; i++)
7      {
8        System.Console.WriteLine(s);
9        s += "*";
10     }
11   }
12 }
```

Código C# 5.25: TrianguloDeAstericos.cs

Compile e execute a classe `TrianguloDeAstericos`

```
C:\Users\K19\rafael\controle-de-fluxo> csc TrianguloDeAstericos.cs

C:\Users\K19\rafael\controle-de-fluxo> TrianguloDeAstericos.exe
*
**
***
```

```
****
*****
******
```

Terminal 5.105: Compilando e executando a classe `TrianguloDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar15.zip

Exercício Complementar 5.16

```
1  class TresTriangulosDeAstericos
2  {
3    static void Main()
4  {
5      for(int i = 0; i < 3; i++)
6      {
7        string s = "*";
8        for(int j = 0; j < 6; j++)
9        {
10         System.Console.WriteLine(s);
11         s += "*";
12       }
13     }
14   }
15 }
```

Código C# 5.27: TresTriangulosDeAstericos.cs

Compile e execute a classe `TresTriangulosDeAstericos`

```
C:\Users\K19\rafael\controle-de-fluxo> csc TresTriangulosDeAstericos.cs

C:\Users\K19\rafael\controle-de-fluxo> TresTriangulosDeAstericos.exe
*
**
***
****
*****
******
*
**
***
****
*****
******
*
**
***
****
*****
******
```

Terminal 5.106: Compilando e executando a classe `TresTriangulosDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar16.zip

Exercício Complementar 5.17

```
1  class LosangoDeAstericos
2  {
```

```
3   static void Main()
4   {
5     string s1 = "";
6     string s2 = "**********";
7     for(int i = 0; i < 5; i++)
8     {
9       System.Console.Write(s1);
10      System.Console.WriteLine(s2);
11      s1 += " ";
12    }
13  }
14 }
```

Código C# 5.29: LosangoDeAstericos.cs

Compile e execute a classe `LosangoDeAstericos`

```
C:\Users\K19\rafael\controle-de-fluxo> csc LosangoDeAstericos.cs

C:\Users\K19\rafael\controle-de-fluxo> LosangoDeAstericos.exe
**********
 **********
  **********
   **********
    **********
```

Terminal 5.107: Compilando e executando a classe `LosangoDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar17.zip

Exercício Complementar 5.18

```
1  class TresLosangosDeAstericos
2  {
3    static void Main()
4    {
5      for(int i = 0; i < 3; i++)
6      {
7        string s1 = "";
8        string s2 = "**********";
9        for(int j = 0; j < 5; j++)
10       {
11         System.Console.Write(s1);
12         System.Console.WriteLine(s2);
13         s1 += " ";
14       }
15     }
16   }
17 }
```

Código C# 5.31: TresLosangosDeAstericos.cs

Compile e execute a classe `TresLosangosDeAstericos`

```
C:\Users\K19\rafael\controle-de-fluxo> csc TresTriangulosDeAstericos.cs

C:\Users\K19\rafael\controle-de-fluxo> TresTriangulosDeAstericos.exe
**********
 **********
  **********
   **********
    **********
**********
```

```
 **********
  **********
   **********
    **********
**********
 **********
  **********
   **********
    **********
```

Terminal 5.108: Compilando e executando a classe `TresLosangosDeAstericos`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar18.zip

Exercício Complementar 5.19

```
1  class CartoesDeEstacionamento
2  {
3    static void Main()
4    {
5      for(int i = 1; i <= 3; i++)
6      {
7        for(int j = 1; j <= 9; j++)
8        {
9          for(int k = 1; k <= 4; k++)
10         {
11           System.Console.WriteLine("BLOCO: " + i + " APTO: " + (j * 10 + k));
12         }
13       }
14     }
15   }
16 }
```

Código C# 5.33: CartoesDeEstacionamento.cs

Compile e execute a classe `CartoesDeEstacionamento`

```
C:\Users\K19\rafael\controle-de-fluxo> csc CartoesDeEstacionamento.cs

C:\Users\K19\rafael\controle-de-fluxo> CartoesDeEstacionamento.exe
BLOCO: 1 APTO: 11
BLOCO: 1 APTO: 12
BLOCO: 1 APTO: 13
BLOCO: 1 APTO: 14
BLOCO: 1 APTO: 21
BLOCO: 1 APTO: 22
BLOCO: 1 APTO: 23
BLOCO: 1 APTO: 24
...
```

Terminal 5.109: Compilando e executando a classe `CartoesDeEstacionamento`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar19.zip

Exercício Complementar 5.20

```
1  class Tabuada
2  {
3    static void Main()
```

```
4   {
5     for (int i = 1; i <= 10; i++)
6     {
7       for (int j = 1; j <= 10; j++)
8       {
9         System.Console.WriteLine(i + "x" + j + " = " + i * j);
10      }
11    }
12  }
13 }
```

Código C# 5.35: Tabuada.cs

Compile e execute a classe Tabuada

```
C:\Users\K19\rafael\controle-de-fluxo> csc Tabuada.cs

C:\Users\K19\rafael\controle-de-fluxo> Tabuada.exe
1x1 = 1
1x2 = 2
1x3 = 3
...
10x8 = 80
10x9 = 90
10x10 = 100
```

Terminal 5.110: Compilando e executando a classe Tabuada

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar20.zip

Exercício Complementar 5.21

```
1  class Piramide
2  {
3    static void Main()
4    {
5      int baseMaior = 7;
6
7      for (int i = 1; i <= baseMaior; i += 2)
8      {
9        int espacos = (baseMaior - i) / 2;
10       string linha = "";
11
12       for (int j = 0; j < espacos; j++)
13       {
14         linha += " ";
15       }
16
17       for (int k = 0; k < i; k++)
18       {
19         linha += "*";
20       }
21
22       System.Console.WriteLine(linha);
23     }
24   }
25 }
```

Código C# 5.37: Piramide.cs

Compile e execute a classe Piramide

```
C:\Users\K19\rafael\controle-de-fluxo> csc Piramide.cs

C:\Users\K19\rafael\controle-de-fluxo> Piramide.exe
   *
  ***
 *****
*******
```

Terminal 5.111: Compilando e executando a classe `Piramide`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar21.zip

Exercício Complementar 5.22

```
1  class ArvoreNatal
2  {
3    static void Main()
4  {
5      int baseMaior = 15;
6
7      for (int m = 7; m <= 15; m += 4)
8      {
9
10       for (int i = m - 6; i <= m; i += 2)
11       {
12         int espacos = (baseMaior - i) / 2;
13         string linha = "";
14
15         for (int j = 0; j < espacos; j++)
16         {
17           linha += " ";
18         }
19
20         for (int k = 0; k < i; k++)
21         {
22           linha += "*";
23         }
24
25         System.Console.WriteLine(linha);
26       }
27     }
28   }
29 }
```

Código C# 5.39: ArvoreNatal.cs

Compile e execute a classe `ArvoreNatal`

```
C:\Users\K19\rafael\controle-de-fluxo> csc ArvoreNatal.cs

C:\Users\K19\rafael\controle-de-fluxo> ArvoreNatal.exe
       *
      ***
     *****
    *******
     *****
    *******
   *********
  ***********
   *********
  ***********
 *************
***************
```

Terminal 5.112: Compilando e executando a classe `ArvoreNatal`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar22.zip

Exercício Complementar 5.23

```
class ContaUns
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    double numero = gerador.NextDouble();
    System.Console.WriteLine(numero);

    string s = "" + numero;
    int resposta = 0;

    for(int i = 0; i < s.Length; i++)
    {
      if(s[i] == '1')
      {
        resposta++;
      }
    }

    System.Console.WriteLine(resposta);
  }
}
```

Código C# 5.41: ContaUns.cs

Compile e execute a classe `ContaUns`

```
C:\Users\K19\rafael\controle-de-fluxo> csc ContaUns.cs

C:\Users\K19\rafael\controle-de-fluxo> ContaUns.exe
0.46029982416581405
2

C:\Users\K19\rafael\controle-de-fluxo> ContaUns.exe
0.24021506754711108
4

C:\Users\K19\rafael\controle-de-fluxo> ContaUns.exe
0.9189156985811225
4
```

Terminal 5.113: Compilando e executando a classe `ContaUns`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar23.zip

Exercício Complementar 5.24

```
class JogoDaSomaImpar
{
  static void Main()
  {
    System.Random gerador = new System.Random();
    int soma = 0;
    int quantidadeDeSeis = 0;
```

```
8
9       for(int i = 0; i < 10; i++)
10      {
11        int numero = (int)(gerador.NextDouble() * 6 + 1);
12
13        System.Console.WriteLine("Número: " + numero);
14
15        if(numero == 1)
16        {
17          continue;
18        }
19
20        if(numero == 6)
21        {
22          quantidadeDeSeis++;
23        }
24
25        if(quantidadeDeSeis == 2)
26        {
27          System.Console.WriteLine("Dois seis! Você perdeu!");
28          break;
29        }
30
31        soma += numero;
32      }
33
34      if(quantidadeDeSeis != 2)
35  {
36        System.Console.WriteLine("Soma: " + soma);
37        if(soma % 2 != 0)
38        {
39          System.Console.WriteLine("Soma ímpar! Você ganhou");
40        } else
41        {
42          System.Console.WriteLine("Soma par! Você perdeu");
43        }
44      }
45    }
46  }
```

Código C# 5.43: JogoDaSomaImpar.cs

Compile e execute a classe `JogoDaSomaImpar`

```
C:\Users\K19\rafael\controle-de-fluxo> csc JogoDaSomaImpar.cs

C:\Users\K19\rafael\controle-de-fluxo> JogoDaSomaImpar.exe
Número: 5
Número: 3
Número: 2
Número: 5
Número: 2
Número: 6
Número: 5
Número: 6
Dois seis! Você perdeu!

C:\Users\K19\rafael\controle-de-fluxo> JogoDaSomaImpar.exe
Número: 1
Número: 2
Número: 2
Número: 1
Número: 6
Número: 1
Número: 4
Número: 3
Número: 2
Número: 2
Soma: 21
Soma ímpar! Você ganhou

C:\Users\K19\rafael\controle-de-fluxo> JogoDaSomaImpar.exe
Número: 4
```

```
Número: 4
Número: 2
Número: 4
Número: 3
Número: 5
Número: 5
Número: 3
Número: 6
Número: 2
Soma: 38
Soma par! Você perdeu
```

Terminal 5.114: Compilando e executando a classe `JogoDaSomaImpar`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-controle-de-fluxo-complementar24.zip

Questão 5.1

a

Questão 5.2

c

Questão 5.3

e

Questão 5.4

d

Questão 5.5

d

Questão 5.6

e

Questão 5.7

a

Questão 5.8

d

Questão 5.9

d

Questão 5.10

e

Questão 5.11

c

Exercício Complementar 6.1

```
1  class GeraradorDeGabarito {
2    public static void main(String[] args) {
3      int[] gabarito = new int[10];
4      for(int i = 0; i < gabarito.length; i++) {
5        gabarito[i] = (int)(Math.random() * 3 + 1);
6        System.out.print(gabarito[i] + " ");
7      }
8      System.out.println("gabarito");
9    }
10 }
```

Código Java 6.22: GeraradorDeGabarito.java

Compile e execute a classe `GeraradorDeGabarito`

```
K19/rafael/arrays$ javac GeraradorDeGabarito.java

K19/rafael/arrays$ java GeraradorDeGabarito
3 3 2 3 1 3 3 2 2 1 gabarito
```

Terminal 6.21: Compilando e executando a classe `GeraradorDeGabarito`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar1.zip

Exercício Complementar 6.2

```
1  class GeraradorDeRespostasAleatorias {
2    public static void main(String[] args) {
3      int[][] respostas = new int[5][10];
4      for(int i = 0; i < respostas.length; i++) {
5        for(int j = 0; j < respostas[i].length; j++) {
6          respostas[i][j] = (int)(Math.random() * 3 + 1);
7          System.out.print(respostas[i][j] + " ");
8        }
9        System.out.println("aluno " + (i + 1));
10     }
11   }
```

```
12 }
```

Código Java 6.24: GeraradorDeRespostasAleatorias.java

Compile e execute a classe `GeraradorDeRespostasAleatorias`

```
K19/rafael/arrays$ javac GeraradorDeRespostasAleatorias.java

K19/rafael/arrays$ java GeraradorDeRespostasAleatorias
1 1 1 1 3 1 3 3 3 1 aluno 1
2 3 3 1 3 2 3 1 2 1 aluno 2
1 1 3 1 3 3 3 2 1 3 aluno 3
3 2 1 2 3 1 3 3 2 1 aluno 4
2 3 2 2 3 2 3 3 2 1 aluno 5
```

Terminal 6.22: Compilando e executando a classe `GeraradorDeRespostasAleatorias`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar2.zip

Exercício Complementar 6.3

```
1  class CorretorDeProva {
2    public static void main(String[] args) {
3      int[] gabarito = new int[10];
4
5      for(int i = 0; i < gabarito.length; i++) {
6        gabarito[i] = (int)(Math.random() * 3 + 1);
7        System.out.print(gabarito[i] + " ");
8      }
9      System.out.println("gabarito");
10
11     int[][] respostas = new int[5][10];
12
13     for(int i = 0; i < respostas.length; i++) {
14       for(int j = 0; j < respostas[i].length; j++) {
15         respostas[i][j] = (int)(Math.random() * 3 + 1);
16         System.out.print(respostas[i][j] + " ");
17       }
18       System.out.println("aluno " + (i + 1));
19     }
20
21     System.out.println("Resultado:");
22     for(int i = 0; i < respostas.length; i++) {
23       int acertos = 0;
24       for(int j = 0; j < respostas[i].length; j++) {
25         if(gabarito[j] == respostas[i][j]) {
26           acertos++;
27         }
28       }
29       System.out.println("Aluno " + (i + 1) + ": " + acertos);
30     }
31   }
32 }
```

Código Java 6.26: CorretorDeProva.java

Compile e execute a classe `CorretorDeProva`

```
K19/rafael/arrays$ javac CorretorDeProva.java

K19/rafael/arrays$ java CorretorDeProva
3 3 2 3 1 3 3 2 2 1 gabarito
3 1 2 2 3 3 1 1 1 1 aluno 1
```

```
3 2 1 1 1 1 3 2 1 2 aluno 2
3 3 3 3 2 3 2 1 3 3 aluno 3
2 2 1 1 1 3 2 1 1 1 aluno 4
3 3 3 1 1 2 1 2 1 1 aluno 5
Resultado:
Aluno 1: 4
Aluno 2: 4
Aluno 3: 4
Aluno 4: 3
Aluno 5: 5
```

Terminal 6.23: Compilando e executando a classe `CorretorDeProva`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar3.zip

Exercício Complementar 6.4

```
1  class ControleDeVagas {
2    public static void main(String[] args) {
3      boolean[][] vagas = new boolean[4][10];
4      for(int i = 0; i < vagas.length; i++) {
5        for(int j = 0; j < vagas[i].length; j++) {
6          vagas[i][j] = Math.random() < 0.5;
7          System.out.print(vagas[i][j] ?  "- " : "X ");
8        }
9        System.out.println("andar " + (i + 1));
10     }
11   }
12 }
```

Código Java 6.28: ControleDeVagas.java

Compile e execute a classe `ControleDeVagas`

```
K19/rafael/arrays$ javac ControleDeVagas.java

K19/rafael/arrays$ java ControleDeVagas
X X - X - - - X X - andar 1
X X - - - - X X - - andar 2
X - - - X - - - - X andar 3
X X - X X - - X X X andar 4
```

Terminal 6.24: Compilando e executando a classe `ControleDeVagas`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar4.zip

Exercício Complementar 6.5

```
1  class Estacionamento {
2    public static void main(String[] args) {
3      boolean[][] vagas = new boolean[4][10];
4      for(int i = 0; i < vagas.length; i++) {
5        for(int j = 0; j < vagas[i].length; j++) {
6          vagas[i][j] = Math.random() < 0.5;
7          System.out.print(vagas[i][j] ?  "- " : "X ");
8        }
9        System.out.println("andar " + (i + 1));
10     }
```

```
11
12      System.out.println("Vagas Livres");
13      for(int i = 0; i < vagas.length; i++) {
14        int vagasLivres = 0;
15        for(int j = 0; j < vagas[i].length; j++) {
16          if(vagas[i][j]) {
17            vagasLivres++;
18          }
19        }
20        System.out.println("Andar " + (i + 1) + ": " + vagasLivres);
21      }
22    }
23  }
```

Código Java 6.30: Estacionamento.java

Compile e execute a classe `Estacionamento`

```
K19/rafael/arrays$ javac Estacionamento.java

K19/rafael/arrays$ java Estacionamento
X X X X X X X - X X andar 1
X - X - X - X - X X andar 2
X X X - - X - X - - andar 3
- - X X X - X - - andar 4
Vagas Livres
Andar 1: 1
Andar 2: 4
Andar 3: 5
Andar 4: 5
```

Terminal 6.25: Compilando e executando a classe `Estacionamento`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar5.zip

Exercício Complementar 6.6

```
1  class GeraradorDeGabarito
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6      int[] gabarito = new int[10];
7      for(int i = 0; i < gabarito.Length; i++)
8      {
9        gabarito[i] = (int)(gerador.NextDouble() * 3 + 1);
10       System.Console.Write(gabarito[i] + " ");
11     }
12     System.Console.WriteLine("gabarito");
13   }
14 }
```

Código C# 6.16: GeraradorDeGabarito.cs

Compile e execute a classe `GeraradorDeGabarito`

```
C:\Users\K19\rafael\arrays> csc GeraradorDeGabarito.cs

C:\Users\K19\rafael\arrays> GeraradorDeGabarito.exe
3 3 2 3 1 3 3 2 2 1 gabarito
```

Terminal 6.26: Compilando e executando a classe `GeraradorDeGabarito`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar6.zip

Exercício Complementar 6.7

```
class GeraradorDeRespostasAleatorias
{
  static void Main()
  {
    System.Random gerador = new System.Random();
    int[][] respostas = new int[5][];
    for(int i = 0; i < respostas.Length; i++)
    {
      respostas[i] = new int[10];
      for(int j = 0; j < respostas[i].Length; j++)
      {
        respostas[i][j] = (int)(gerador.NextDouble() * 3 + 1);
        System.Console.Write(respostas[i][j] + " ");
      }
      System.Console.WriteLine("aluno " + (i + 1));
    }
  }
}
```

Código C# 6.18: GeraradorDeRespostasAleatorias.cs

Compile e execute a classe `GeraradorDeRespostasAleatorias`

```
C:\Users\K19\rafael\arrays> csc GeraradorDeRespostasAleatorias.cs

C:\Users\K19\rafael\arrays> GeraradorDeRespostasAleatorias.exe
1 1 1 1 3 1 3 3 3 1 aluno 1
2 3 3 1 3 2 3 1 2 1 aluno 2
1 1 3 1 3 3 3 2 1 3 aluno 3
3 2 1 2 3 1 3 3 2 1 aluno 4
2 3 2 2 3 2 3 3 2 1 aluno 5
```

Terminal 6.27: Compilando e executando a classe `GeraradorDeRespostasAleatorias`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar7.zip

Exercício Complementar 6.8

```
class CorretorDeProva
{
  static void Main()
  {
    System.Random gerador = new System.Random();

    int[] gabarito = new int[10];

    for(int i = 0; i < gabarito.Length; i++)
    {
      gabarito[i] = (int)(gerador.NextDouble() * 3 + 1);
      System.Console.Write(gabarito[i] + " ");
    }
    System.Console.WriteLine("gabarito");

    int[][] respostas = new int[5][];
```

```
17
18      for(int i = 0; i < respostas.Length; i++)
19      {
20        respostas[i] = new int[10];
21        for(int j = 0; j < respostas[i].Length; j++)
22        {
23          respostas[i][j] = (int)(gerador.NextDouble() * 3 + 1);
24          System.Console.Write(respostas[i][j] + " ");
25        }
26        System.Console.WriteLine("aluno " + (i + 1));
27      }
28
29      System.Console.WriteLine("Resultado:");
30      for(int i = 0; i < respostas.Length; i++)
31      {
32        int acertos = 0;
33        for(int j = 0; j < respostas[i].Length; j++)
34        {
35          if(gabarito[j] == respostas[i][j])
36          {
37            acertos++;
38          }
39        }
40        System.Console.WriteLine("Aluno " + (i + 1) + ": " + acertos);
41      }
42    }
43  }
```

Código C# 6.20: CorretorDeProva.cs

Compile e execute a classe `CorretorDeProva`

```
C:\Users\K19\rafael\arrays> csc CorretorDeProva.cs

C:\Users\K19\rafael\arrays> CorretorDeProva.exe
3 3 2 3 1 3 3 2 2 1 gabarito
3 1 2 2 3 3 1 1 1 1 aluno 1
3 2 1 1 1 1 3 2 1 2 aluno 2
3 3 3 3 2 3 2 1 3 3 aluno 3
2 2 1 1 1 3 2 1 1 1 aluno 4
3 3 3 1 1 2 1 2 1 1 aluno 5
Resultado:
Aluno 1: 4
Aluno 2: 4
Aluno 3: 4
Aluno 4: 3
Aluno 5: 5
```

Terminal 6.28: Compilando e executando a classe `CorretorDeProva`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar8.zip

Exercício Complementar 6.9

```
1   class ControleDeVagas
2   {
3     static void Main()
4     {
5       System.Random gerador = new System.Random();
6       bool[][] vagas = new bool[4][];
7       for(int i = 0; i < vagas.Length; i++)
8       {
9         vagas[i] = new bool[10];
10        for(int j = 0; j < vagas[i].Length; j++)
11        {
```

```
12          vagas[i][j] = gerador.NextDouble() < 0.5;
13          System.Console.Write(vagas[i][j] ?  "- " : "X ");
14        }
15        System.Console.WriteLine("andar " + (i + 1));
16      }
17    }
18  }
```

Código C# 6.22: ControleDeVagas.cs

Compile e execute a classe `ControleDeVagas`

```
C:\Users\K19\rafael\arrays> csc ControleDeVagas.cs

C:\Users\K19\rafael\arrays> ControleDeVagas.exe
X X - X - - - X X - andar 1
X X - - - - X X - - andar 2
X - - - X - - - - X andar 3
X X - X X - - X X X andar 4
```

Terminal 6.29: Compilando e executando a classe `ControleDeVagas`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar9.zip

Exercício Complementar 6.10

```
1  class Estacionamento
2  {
3    static void Main()
4    {
5      System.Random gerador = new System.Random();
6      bool[][] vagas = new bool[4][];
7      for(int i = 0; i < vagas.Length; i++)
8      {
9        vagas[i] = new bool[10];
10       for(int j = 0; j < vagas[i].Length; j++)
11       {
12         vagas[i][j] = gerador.NextDouble() < 0.5;
13         System.Console.Write(vagas[i][j] ?  "- " : "X ");
14       }
15       System.Console.WriteLine("andar " + (i + 1));
16     }
17
18     System.Console.WriteLine("Vagas Livres");
19     for(int i = 0; i < vagas.Length; i++)
20     {
21       int vagasLivres = 0;
22       for(int j = 0; j < vagas[i].Length; j++)
23       {
24         if(vagas[i][j])
25         {
26           vagasLivres++;
27         }
28       }
29       System.Console.WriteLine("Andar " + (i + 1) + ": " + vagasLivres);
30     }
31   }
32 }
```

Código C# 6.24: Estacionamento.cs

Compile e execute a classe `Estacionamento`

```
C:\Users\K19\rafael\arrays> csc Estacionamento.cs

C:\Users\K19\rafael\arrays> Estacionamento.exe
X X X X X X - X X andar 1
X - X - X - X X andar 2
X X X - - X - X - - andar 3
- - X X X X - X - - andar 4
Vagas Livres
Andar 1: 1
Andar 2: 4
Andar 3: 5
Andar 4: 5
```

Terminal 6.30: Compilando e executando a classe `Estacionamento`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-arrays-complementar10.zip

Questão 6.1

a

Questão 6.2

b

Questão 6.3

e

Questão 6.4

a

Questão 6.5

d

Questão 6.6

e

Exercício Complementar 7.1

```
1 class AnoBissexto {
2   public static void main(String[] args) {
3     boolean b = bissexto(2000);
4     System.out.println("2000 " + b);
5
```

```
6     b = bissexto(2012);
7     System.out.println("2012 " + b);
8
9     b = bissexto(2025);
10    System.out.println("2025 " + b);
11
12    b = bissexto(2100);
13    System.out.println("2100 " + b);
14  }
15
16  static boolean bissexto(int ano){
17    return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
18  }
19 }
```

Código Java 7.22: AnoBissexto.java

Compile e execute a classe `AnoBissexto`

```
K19/rafael/funcoes-ou-metodos$ javac AnoBissexto.java

K19/rafael/funcoes-ou-metodos$ java AnoBissexto
2000 true
2012 true
2025 false
2100 false
```

Terminal 7.20: Compilando e executando a classe `AnoBissexto`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar1.zip

Exercício Complementar 7.2

```
1  class VerificaDatas {
2    public static void main(String[] args) {
3      boolean b = verificaData(29, 2, 2100);
4
5      System.out.println("29/02/2100 - " + b);
6
7      b = verificaData(29, 2, 2004);
8
9      System.out.println("29/02/2004 - " + b);
10
11     b = verificaData(31, 4, 2000);
12
13     System.out.println("31/04/2000 - " + b);
14   }
15
16   static boolean bissexto(int ano){
17     return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
18   }
19
20   static boolean verificaData(int dia, int mes, int ano) {
21     int[] dias = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
22
23     boolean b = bissexto(ano);
24
25     if(b) {
26       dias[1] = 29;
27     }
28
29     return (dia >= 1 && dia <= dias[mes - 1]) && (mes >= 1 && mes <= 12) && ano >= 1;
30   }
```

```
31 }
```

Código Java 7.24: VerificaDatas.java

Compile e execute a classe `AnoBissexto`

```
K19/rafael/funcoes-ou-metodos$ javac VerificaDatas.java

K19/rafael/funcoes-ou-metodos$ java VerificaDatas
29/02/2100 - false
29/02/2004 - true
31/04/2000 - false
```

Terminal 7.21: Compilando e executando a classe `AnoBissexto`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar2.zip

Exercício Complementar 7.3

```
1  class Pascoa {
2    public static void main(String[] args) {
3      String s = pascoa(2000);
4      System.out.println("Páscoa " + s);
5
6      s = pascoa(2012);
7      System.out.println("Páscoa " + s);
8
9      s = pascoa(2025);
10     System.out.println("Páscoa " + s);
11
12     s = pascoa(2100);
13     System.out.println("Páscoa " + s);
14   }
15
16   static String pascoa(int ano){
17     int a = ano % 19;
18     int b = ano / 100;
19     int c = ano % 100;
20     int d = b / 4;
21     int e = b % 4;
22     int f = (b + 8) / 25;
23     int g = (b - f + 1) / 3;
24     int h = (19 * a + b -d - g + 15) % 30;
25     int i = c / 4;
26     int k = c % 4;
27     int l = (32 + 2 * e + 2 * i - h - k) % 7 ;
28     int m = (a + 11 * h + 22 * l) / 451;
29
30     int mes = (h + l - 7 * m + 114) / 31;
31     int dia = ((h + l - 7 * m + 114) % 31) + 1;
32
33     return dia + "/" + mes + "/" + ano;
34   }
35 }
```

Código Java 7.27: Pascoa.java

Compile e execute a classe `Pascoa`

```
K19/rafael/funcoes-ou-metodos$ javac Pascoa.java

K19/rafael/funcoes-ou-metodos$ java Pascoa
```

```
Páscoa: 23/4/2000
Páscoa: 8/4/2012
Páscoa: 20/4/2025
Páscoa: 28/3/2100
```

Terminal 7.22: Compilando e executando a classe Pascoa

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar3.zip

Exercício Complementar 7.4

```
1  class DiaDaSemana {
2    public static void main(String[] args) {
3      int d1 = diaDaSemana(30, 10, 1984);
4
5      int d2 = diaDaSemana(2, 4, 1985);
6
7      int d3 = diaDaSemana(12, 12, 1982);
8
9      String[] dias = {
10       "domingo",
11       "segunda",
12       "terça",
13       "quarta",
14       "quinta",
15       "sexta",
16       "sábado"
17     };
18
19     System.out.println("30/10/1984 foi " + dias[d1]);
20
21     System.out.println("2/4/1985 foi " + dias[d2]);
22
23     System.out.println("12/12/1982 foi " + dias[d3]);
24   }
25
26   static int diaDaSemana(int dia, int mes, int ano) {
27     int a = (14 - mes) / 12;
28     int y = ano - a;
29     int m = mes + 12 * a - 2;
30     int q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
31     int d = q % 7;
32
33     return d;
34   }
35 }
```

Código Java 7.30: Pascoa.java

Compile e execute a classe DiaDaSemana

```
K19/rafael/funcoes-ou-metodos$ javac DiaDaSemana.java

K19/rafael/funcoes-ou-metodos$ java DiaDaSemana
30/10/1984 foi terça
2/4/1985 foi terça
12/12/1982 foi domingo
```

Terminal 7.23: Compilando e executando a classe DiaDaSemana

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar4.zip

Exercício Complementar 7.5

```
class ExibeCalendario {
  public static void main(String[] args) {

    exibeCalendario(10, 1984);

    exibeCalendario(4, 1985);

    exibeCalendario(12, 1982);

    exibeCalendario(2, 2000);

  }

  static boolean bissexto(int ano){
    return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
  }

  static int diaDaSemana(int dia, int mes, int ano) {
    int a = (14 - mes) / 12;
    int y = ano - a;
    int m = mes + 12 * a - 2;
    int q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
    int d = q % 7;

    return d;
  }

  static void exibeCalendario(int mes, int ano) {
    int[] dias = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    boolean b = bissexto(ano);

    if(b) {
      dias[1] = 29;
    }

    int d = diaDaSemana(1, mes, ano);

    System.out.println("Dom Seg Ter Qua Qui Sex Sab");

    // espaços da primeira semana
    for(int i = 0; i < d; i++) {
      System.out.print("    ");
    }

    for(int i = 1; i <= dias[mes - 1]; i++) {
      String dia = "" + i;
      if(i < 10) {
        dia = "0" + dia;
      }

      System.out.print(" " + dia + " ");

      if((i + d) % 7 == 0) {
        System.out.println();
      }
    }
    System.out.println("\n----------------------------");
  }
}
```

Código Java 7.32: ExibeCalendario.java

Compile e execute a classe `ExibeCalendario`

```
K19/rafael/funcoes-ou-metodos$ javac ExibeCalendario.java

K19/rafael/funcoes-ou-metodos$ java ExibeCalendario
Dom Seg Ter Qua Qui Sex Sab
     01  02  03  04  05  06
 07  08  09  10  11  12  13
 14  15  16  17  18  19  20
 21  22  23  24  25  26  27
 28  29  30  31
----------------------------
Dom Seg Ter Qua Qui Sex Sab
     01  02  03  04  05  06
 07  08  09  10  11  12  13
 14  15  16  17  18  19  20
 21  22  23  24  25  26  27
 28  29  30
----------------------------
Dom Seg Ter Qua Qui Sex Sab
             01  02  03  04
 05  06  07  08  09  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28  29  30  31
----------------------------
Dom Seg Ter Qua Qui Sex Sab
         01  02  03  04  05
 06  07  08  09  10  11  12
 13  14  15  16  17  18  19
 20  21  22  23  24  25  26
 27  28  29
----------------------------
```

Terminal 7.25: Compilando e executando a classe `ExibeCalendario`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar5.zip

Exercício Complementar 7.6

```
1  class AnoBissexto
2  {
3    static void Main()
4    {
5      bool b = bissexto(2000);
6      System.Console.WriteLine("2000 " + b);
7
8      b = bissexto(2012);
9      System.Console.WriteLine("2012 " + b);
10
11     b = bissexto(2025);
12     System.Console.WriteLine("2025 " + b);
13
14     b = bissexto(2100);
15     System.Console.WriteLine("2100 " + b);
16   }
17
18   static bool bissexto(int ano)
19   {
20     return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
21   }
22 }
```

Código C# 7.10: AnoBissexto.cs

Compile e execute a classe `AnoBissexto`

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc AnoBissexto.cs

C:\Users\K19\rafael\funcoes-ou-metodos> AnoBissexto.exe
2000 True
2012 True
2025 False
2100 False
```

Terminal 7.26: Compilando e executando a classe `AnoBissexto`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar6.zip

Exercício Complementar 7.7

```
1  class VerificaDatas
2  {
3    static void Main()
4    {
5      bool b = verificaData(29, 2, 2100);
6
7      System.Console.WriteLine("29/02/2100 - " + b);
8
9      b = verificaData(29, 2, 2004);
10
11     System.Console.WriteLine("29/02/2004 - " + b);
12
13     b = verificaData(31, 4, 2000);
14
15     System.Console.WriteLine("31/04/2000 - " + b);
16   }
17
18   static bool bissexto(int ano)
19   {
20     return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
21   }
22
23   static bool verificaData(int dia, int mes, int ano)
24   {
25     int[] dias = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
26
27     bool b = bissexto(ano);
28
29     if(b)
30     {
31       dias[1] = 29;
32     }
33
34     return (dia >= 1 && dia <= dias[mes - 1]) && (mes >= 1 && mes <= 12) && ano >= 1;
35   }
36 }
```

Código C# 7.12: VerificaDatas.cs

Compile e execute a classe `AnoBissexto`

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc VerificaDatas.cs

C:\Users\K19\rafael\funcoes-ou-metodos> VerificaDatas.exe
29/02/2100 - False
29/02/2004 - True
31/04/2000 - False
```

Terminal 7.27: Compilando e executando a classe `AnoBissexto`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar7.zip

Exercício Complementar 7.8

```
1  class Pascoa
2  {
3    static void Main()
4    {
5      string s = pascoa(2000);
6      System.Console.WriteLine("Páscoa " + s);
7
8      s = pascoa(2012);
9      System.Console.WriteLine("Páscoa " + s);
10
11     s = pascoa(2025);
12     System.Console.WriteLine("Páscoa " + s);
13
14     s = pascoa(2100);
15     System.Console.WriteLine("Páscoa " + s);
16   }
17
18   static string pascoa(int ano)
19   {
20     int a = ano % 19;
21     int b = ano / 100;
22     int c = ano % 100;
23     int d = b / 4;
24     int e = b % 4;
25     int f = (b + 8) / 25;
26     int g = (b - f + 1) / 3;
27     int h = (19 * a + b -d - g + 15) % 30;
28     int i = c / 4;
29     int k = c % 4;
30     int l = (32 + 2 * e + 2 * i - h - k) % 7 ;
31     int m = (a + 11 * h + 22 * l) / 451;
32
33     int mes = (h + l - 7 * m + 114) / 31;
34     int dia = ((h + l - 7 * m + 114) % 31) + 1;
35
36     return dia + "/" + mes + "/" + ano;
37   }
38 }
```

Código C# 7.15: Pascoa.cs

Compile e execute a classe Pascoa

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc Pascoa.cs

C:\Users\K19\rafael\funcoes-ou-metodos> Pascoa.exe
Páscoa: 23/4/2000
Páscoa: 8/4/2012
Páscoa: 20/4/2025
Páscoa: 28/3/2100
```

Terminal 7.28: Compilando e executando a classe Pascoa

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar8.zip

Exercício Complementar 7.9

```
class DiaDaSemana
{
  static void Main()
  {
    int d1 = diaDaSemana(30, 10, 1984);

    int d2 = diaDaSemana(2, 4, 1985);

    int d3 = diaDaSemana(12, 12, 1982);

    string[] dias =
    {
      "domingo",
      "segunda",
      "terça",
      "quarta",
      "quinta",
      "sexta",
      "sábado"
    };

    System.Console.WriteLine("30/10/1984 foi " + dias[d1]);

    System.Console.WriteLine("2/4/1985 foi " + dias[d2]);

    System.Console.WriteLine("12/12/1982 foi " + dias[d3]);
  }

  static int diaDaSemana(int dia, int mes, int ano)
  {
    int a = (14 - mes) / 12;
    int y = ano - a;
    int m = mes + 12 * a - 2;
    int q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
    int d = q % 7;

    return d;
  }
}
```

Código C# 7.18: Pascoa.cs

Compile e execute a classe `DiaDaSemana`

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc DiaDaSemana.cs

C:\Users\K19\rafael\funcoes-ou-metodos> DiaDaSemana.exe
30/10/1984 foi terça
2/4/1985 foi terça
12/12/1982 foi domingo
```

Terminal 7.29: Compilando e executando a classe `DiaDaSemana`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar9.zip

Exercício Complementar 7.10

```
class ExibeCalendario
{
  static void Main()
  {
```

```
5
6       exibeCalendario(10, 1984);
7
8       exibeCalendario(4, 1985);
9
10      exibeCalendario(12, 1982);
11
12      exibeCalendario(2, 2000);
13
14    }
15
16    static bool bissexto(int ano)
17    {
18      return ano % 400 == 0 || (ano % 100 != 0 && ano % 4 == 0);
19    }
20
21    static int diaDaSemana(int dia, int mes, int ano)
22    {
23      int a = (14 - mes) / 12;
24      int y = ano - a;
25      int m = mes + 12 * a - 2;
26      int q = dia + 31 * m / 12 + y + y / 4 - y / 100 + y / 400;
27      int d = q % 7;
28
29      return d;
30    }
31
32    static void exibeCalendario(int mes, int ano)
33    {
34      int[] dias = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
35
36      bool b = bissexto(ano);
37
38      if(b)
39      {
40        dias[1] = 29;
41      }
42
43      int d = diaDaSemana(1, mes, ano);
44
45      System.Console.WriteLine("Dom Seg Ter Qua Qui Sex Sab");
46
47      // espaços da primeira semana
48      for(int i = 0; i < d; i++)
49      {
50        System.Console.Write("    ");
51      }
52
53      for(int i = 1; i <= dias[mes - 1]; i++)
54      {
55        string dia = "" + i;
56        if(i < 10)
57        {
58          dia = "0" + dia;
59        }
60
61        System.Console.Write(" " + dia + " ");
62
63        if((i + d) % 7 == 0)
64        {
65          System.Console.WriteLine();
66        }
67      }
68      System.Console.WriteLine("\n----------------------------");
69    }
70  }
```

Código C# 7.20: ExibeCalendario.cs

Compile e execute a classe `ExibeCalendario`

```
C:\Users\K19\rafael\funcoes-ou-metodos> csc ExibeCalendario.cs

C:\Users\K19\rafael\funcoes-ou-metodos> ExibeCalendario.exe
Dom Seg Ter Qua Qui Sex Sab
     01  02  03  04  05  06
 07  08  09  10  11  12  13
 14  15  16  17  18  19  20
 21  22  23  24  25  26  27
 28  29  30  31
----------------------------
Dom Seg Ter Qua Qui Sex Sab
     01  02  03  04  05  06
 07  08  09  10  11  12  13
 14  15  16  17  18  19  20
 21  22  23  24  25  26  27
 28  29  30
----------------------------
Dom Seg Ter Qua Qui Sex Sab
             01  02  03  04
 05  06  07  08  09  10  11
 12  13  14  15  16  17  18
 19  20  21  22  23  24  25
 26  27  28  29  30  31
----------------------------
Dom Seg Ter Qua Qui Sex Sab
         01  02  03  04  05
 06  07  08  09  10  11  12
 13  14  15  16  17  18  19
 20  21  22  23  24  25  26
 27  28  29
----------------------------
```

Terminal 7.31: Compilando e executando a classe `ExibeCalendario`

Arquivo: https://github.com/K19/K19-Exercicios/archive/k01-funcoes-ou-metodos-complementar10.zip

Questão 7.1

c

Questão 7.2

a

Questão 7.3

e

Questão 7.4

d

Questão 7.5

e

Questão 7.6

b