A Linguagem de Programação Java

- **Programação orientada a objetos:** nasceu porque algumas **linguagens procedimentais** se mostraram inadequadas para a construção de programas de grande porte.
- Existem dois tipos de problemas:
  1. **Falta de correspondência entre o programa e o mundo real:** Os procedimentos implementam tarefas e estruturas de dados armazenam informação, mas a maioria dos objetos do mundo real contém as duas coisas.
  2. **Organização interna dos programas:** Não existe uma maneira flexível para dizer que determinados procedimentos poderiam acessar uma variável enquanto outros não.

A Linguagem de Programação Java

- **Programação orientada a objetos:** permite que objetos do mundo real que compartilham propriedades e comportamentos comuns sejam agrupados em classes.
- Estilo de programação diretamente suportado pelo conceito de classe em Java.
- Pode-se também impor restrições de visibilidade aos dados de um programa.
- Classes e objetos são os conceitos fundamentais nas linguagens orientadas a objeto.
- A linguagem Java possui um grau de orientação a objetos maior do que a linguagem C++.
- Java não é totalmente orientada a objetos como a linguagem **S**malltalk.
- Java não é totalmente orientada a objetos porque, por questões de eficiência, foram mantidos alguns tipos primitivos e suas operações.

Principais Componentes de um Programa Java

- Em Java, as funções e os procedimentos são chamados de **métodos.**
- Um **objeto** contém métodos e variáveis que representam seus campos de dados (atributos).
  - Ex: um objeto $painelDeControle$ deveria conter não somente os métodos $ligaForno$ e $desligaForno$, mas também as variáveis $temperaturaCorrente$ e $temperaturaDese\text{-}jada$.
- O conceito de objeto resolve bem os problemas apontados anteriormente.
  - Os métodos $ligaForno$ e $desligaForno$ podem acessar as variáveis $temperaturaCorrente$ e $temperaturaDesejada$, mas elas ficam escondidas de outros métodos que não fazem parte do objeto $painelDeControle$.

Principais Componentes de um Programa Java

- O conceito de classe nasceu da necessidade de se criar diversos objetos de um mesmo tipo.
- Dizemos que um objeto pertence a uma classe ou, mais comumente, que é uma **i**nstância

```
package cap1;
class PainelDeControle {
   private float temperaturaCorrente;
   private float temperaturaDesejada;

   public void ligaForno () {
      // código do método
   }
   public void desligaForno() {
      // código do método
   }
}
```

- A palavra chave **class** introduz a classe $PainelDeControle$.
- A palavra chave **void** é utilizada para indicar

Principais Componentes de um Programa Java

- Um objeto em Java é criado usando a palavra chave **n**ew
- É necessário armazenar uma referência para ele em uma variável do mesmo tipo da classe, como abaixo:

```
PainelDeControle painel1, painel2;
```

- Posteriormente, cria-se os objetos, como a seguir:

```
painel1 = new PainelDeControle ();
painel2 = new PainelDeControle ();
```

- Outras partes do programa interagem com os métodos dos objetos por meio do operador (**.**), o qual associa um objeto com um de seus métodos, como a seguir:

```
painel1.ligaForno ();
```

Herança e Polimorfismo

- **Herança:** criação de uma classe a partir de uma outra classe.
- A classe é estendida a partir da classe base usando a palavra chave **extends**.
- A classe estendida (**subclasse**) tem todas as características da classe base (**superclasse**) mais alguma característica adicional.
- **Polimorfismo:** tratamento de objetos de classes diferentes de uma mesma forma.
- As classes diferentes devem ser derivadas da mesma classe base.

Herança e Polimorfismo

```
package cap1;
class Empregado {
  protected float salario;
  public float salarioMensal () { return salario; }
  public void imprime () { System.out.println ("Empregado");
}
class Secretaria extends Empregado {
  private int velocidadeDeDigitacao;
  public void imprime () { System.out.println ("Secretaria");
}
class Gerente extends Empregado {
  private float bonus;
  public float salarioMensal () { return salario + bonus; }
  public void imprime () { System.out.println ("Gerente"); }
}
public class Polimorfismo {
  public static void main (String[] args) {
    Empregado empregado = new Empregado ();
    Empregado secretaria = new Secretaria ();
    Empregado gerente = new Gerente ();
    empregado.imprime (); secretaria.imprime ();
    gerente.imprime ();
  }
}
```

Objetos e Tipos Genéricos

- Uma estrutura de dados é genérica quando o tipo dos dados armazenados na estrutura é definido na aplicação que a utiliza (**objetos genéricos**).
- Um **objeto genérico** pode armazenar uma referência para um objeto de qualquer classe (classe **Object** em Java).
- Os mecanismos de **herança** e **polimorfismo** que permitem a implementação de estruturas de dados genéricas.

```
package cap1.objetogenerico;
public class Lista {
  private static class Celula {
    Object item; Celula prox;
  }
  private Celula primeiro, ultimo;
}
```

- O objeto $item$ é definido como um objeto genérico, assim $Lista$ pode ter objetos de classes distintas em cada item

Objetos e Tipos Genéricos

- Para evitar que se declare o tipo de cada objeto a ser inserido ou retirado da lista, a **Versão 5** da linguagem Java introduziu um mecanismo de definição de um tipo genérico.
- **Tipo genérico:** definição de um parâmetro de tipo que deve ser especificado na aplicação que utiliza a estrutura de dados:

```
package cap1.tipogenerico;
public class Lista<T> {
  private static class Celula<T> {
    T item;
    Celula<T> prox;
  }
  private Celula<T> primeiro, ultimo;
}
```

- O objeto $item$ tem de ser uma instância de um tipo genérico T que será fornecido quando um objeto da classe $Lista$ for instanciado.
  - Para instanciar uma lista de inteiros basta declarar o comando “Lista<Integer> lista = **n**ew Lista<Integer>();”.

Sobrecarga

- A **sobrecarga** acontece quando determinado objeto se comporta de diferentes formas.
- É um tipo de **p**olimorfismo $ad\ hoc$, no qual um identificador representa vários métodos com computações distintas.

```
public float salarioMensal (float desconto) {
  return salario + bonus − desconto;
}
```

- O programa acima apresenta um exemplo de sobrecarga do método $salarioMensal$ da classe $Gerente$ mostrada em um programa anterior, em que um $desconto$ é subtraído de $salario + bonus$.
- Note que o método $salarioMensal$ do programa acima possui uma assinatura diferente da assinatura apresentada no programa anterior.

Sobrescrita

- A ocultação de um método de uma classe mais genérica em uma classe mais específica é chamada de **s**obrescrita
- Por exemplo, o método $imprime$ da classe $Empregado$ apresentada nas parte de Herança e Polimorfismo, foi sobrescrito nas classes $Gerente$ e $Secretaria$.
- Para sobrescrever um método em uma subclasse é preciso que ele tenha a mesma assinatura na superclasse.

Programa Principal

```
package cap1;
class ContaBancaria {
  private double saldo;
  public ContaBancaria (double saldoInicial) {
    saldo = saldoInicial;
  }
  public void deposito (double valor) {
    saldo = saldo + valor;
  }
  public void saque (double valor) {
    saldo = saldo – valor;
  }
  public void imprime () {
    System.out.println ("saldo=" + saldo);
  }
}
public class AplicacaoBancaria {
  public static void main (String[] args) {
    ContaBancaria conta1 = new ContaBancaria (200.00);
    System.out.print ("Antes da movimentacao, ");
    conta1.imprime ();
    conta1.deposito (50.00); conta1.saque (70.00);
    System.out.print ("Depois da movimentacao, ");
    conta1.imprime ();
  }
```

Programa Principal

- Programa anterior modela uma conta bancária típica com as operações: cria uma conta com um saldo inicial; imprime o saldo; realiza um depósito; realiza um saque e imprime o novo saldo;
- A classe $Contabancaria$ tem um campo de dados chamado $saldo$ e três métodos chamados $deposito$, $saque$ e $imprime$.
- Para **c**ompilar o Programa acima a partir de uma linha de comando em MS-DOS ou Linux, fazemos:

  javac -d **./** $AplicacaoBancaria.java$

  e para executá-lo, fazemos:

  java $cap1.AplicacaoBancaria$
- A classe $ContaBancaria$ tem um método especial denominado **construtor**, que é chamado automaticamente sempre que um novo objeto é criado com o comando **new** e tem sempre o mesmo nome da classe.

Modificadores de Acesso

- **Modificadores de acesso:** determinam quais outros métodos podem acessar um campo de dados ou um método.
- Um campo de dados ou um método que seja precedido pelo modificador **private** pode ser acessado somente por métodos que fazem parte da mesma classe.
- Um campo de dados ou um método que seja precedido pelo modificador **public** pode ser acessado por métodos de outras classes.
  - Classe modificada com o modificador **public** indica que a classe é visível externamente ao pacote em que ela foi definida (classe $AplicacaoBancaria$, **package** cap1).
  - Em cada arquivo de um programa Java só pode existir uma classe modificada por **p**ublic, e o nome do arquivo deve ser o mesmo dado à classe.
- Os campos de dados de uma classe são geralmente feitos **private** e os métodos são tornados **public**.

Modificadores de Acesso

- Modificador protected: utilizado para permitir que somente subclasses de uma classe mais genérica possam acessar os campos de dados precedidos com **protected**.
- Um campo de dados ou um método de uma classe declarado como **static** pertence à classe e não às suas instâncias, ou seja, somente um campo de dados ou um método será criado pelo compilador para todas as instâncias.
- Os métodos de uma classe que foram declarados **static** operam somente sobre os campos da classe que também foram declarados **static**.
- Se além de **static** o método for declarado **public** será possível acessá-lo com o nome da classe e o operador (**.**).

Modificadores de Acesso

```
package cap1;
class A {
  public static int total;
  public int media;
}
public class B {
  public static void main (String[] args) {
    A a = new A(); a.total = 5; a.media = 5;
    A b = new A(); b.total = 7; b.media = 7;
  }
}
```

- No exemplo acima, o campo de dados $total$ pertence somente à classe A, enquanto o campo de dados $media$ pertence a todas as instâncias da classe A.
- Ao final da execução do método $main$, os valores de $a.total$ e $b.total$ são iguais a 7, enquanto os valores de $a.media$ e $b.media$ são iguais a 5 e 7, respectivamente.

Interfaces

- Uma **i**nterface em Java é uma **classe abstrata** que não pode ser instanciada, cujos os métodos devem ser **p**ublic e somente suas assinaturas são definidas
- Uma **i**nterface é sempre implementada por outras classes.
- Utilizada para prover a especificação de um comportamento que seja comum a um conjunto de objetos.

```
package cap1;
import java.io.*;
public class Max {
  public static Item max (Item v[], int n) {
    Item max = v[0];
    for (int i = 1; i < n; i++)
        if (max.compara (v[i]) < 0) max = v[i];
    return max;
  }
}
```

- O programa acima apresenta uma versão generalizada do programa para obter o máximo de um conjunto de inteiros.

Interfaces

- Para permitir a generalização do tipo de dados da chave é necessário criar a interface $Item$ que apresenta a assinatura do método abstrato $compara$.

```
package cap1;
public interface Item {
  public int compara (Item it);
}
```

- A classe $MeuItem$, o tipo de dados da chave é definido e o método $compara$ é implementado.

```
package cap1;
import java.io.*;
public class MeuItem implements Item {
  public int chave;
  // outros componentes do registro

  public MeuItem (int chave) { this.chave = chave; }
  public int compara (Item it) {
    MeuItem item = (MeuItem) it;
    if (this.chave < item.chave) return -1;
    else if (this.chave > item.chave) return 1;
    return 0;
  }
```

Interfaces

```
package cap1;
public class EncontraMax {
  public static void main (String[] args) {
    MeuItem v[] = new MeuItem[2];
    v[0] = new MeuItem (3); v[1] = new MeuItem (10);
    MeuItem max = (MeuItem) Max.max (v, 2);
    System.out.println ("Maior chave: " + max.chave);
  }
}
```

- O programa acima ilustra a utilização do método $compara$ apresentado.
- Note que para atribuir a um objeto da classe $MeuItem$ o valor máximo retornado pelo método max é necessário fazer uma conversão do tipo $Item$ para o tipo $MeuItem$, conforme ilustra a penúltima linha do método $main$.

Pacotes

- A linguagem Java permite agrupar as classes e as interfaces em pacotes(do inglês, **p**ackage.
- Convenientes para organizar e separar as classes de um conjunto de programas de outras bibliotecas de classes, evitando colisões entre nomes de classes desenvolvidas por uma equipe composta por muitos programadores.
- Deve ser realizada sempre na primeira linha do arquivo fonte, da seguinte forma por exemplo:

  package cap1;

- É possível definir subpacotes separados por ".", por exemplo, para definir o subpacote $arranjo$ do pacote $cap3$ fazemos:

  package cap3.arranjo;

- A utilização de uma classe definida em outro pacote é realizada através da palavra chave **import**. O comando abaixo possibilita a utilização de todas as classes de um pacote:

Pacotes

- É possível utilizar determinada classe de um pacote sem importá-la, para isso basta prefixar o nome da classe com o nome do pacote durante a declaração de uma variável. Exemplo:

    cap3.arranjo.Lista lista;

- Para que uma classe possa ser importada em um pacote diferente do que ela foi definida é preciso declará-la como pública por meio do modificador **public**.

- Se o comando **package** não é colocado no código fonte, então Java adiciona as classes daquele código fonte no que é chamado de pacote *default*

- Quando o modificador de um campo ou método não é estabelecido, diz-se que o campo ou método possui visibilidade *default*, ou seja, qualquer objeto de uma classe do pacote pode acessar diretamente aquele campo (ou método).

Classes Internas

- Java permite realizar aninhamento de classes como abaixo:

```
package cap1;
public class Lista {
   // Código da classe Lista
   private class Celula {
      // Código da classe Celula
   }
}
```

- Classes internas são muito úteis para evitar conflitos de nomes.
- Os campos e métodos declarados na classe externa podem ser diretamente acessados dentro da classe interna, mesmo os declarados como **protected** ou **private**, mas o contrário não é verdadeiro.
- As classes externas só podem ser declaradas como públicas ou com visibilidade *default*.
- As classes internas podem também ser qualificadas com os modificadores **p**rivate, **p**rotected e **s**tatic e o efeito é mesmo obtido sobre qualquer atributo da classe externa

O Objeto *this*

- Toda instância de uma classe possui uma variável especial chamada **t**his, que contém uma referência para a própria instância.
- Em algumas situações resolve questões de ambigüidade.

```
package cap1;
public class Conta {
   private double saldo;
   public void alteraSaldo (double saldo) {
      this.saldo = saldo;
   }
}
```

- No exemplo acima, o parâmetro $saldo$ do método $alteraSaldo$ possui o mesmo nome do campo de instância $saldo$ da classe $Conta$.
- Para diferenciá-los é necessário qualificar o campo da instância com o objeto **t**his.

Exceções

- As exceções são erros ou anomalias que podem ocorrer durante a execução de um programa.
- Deve ser obrigatoriamente representada por um objeto de uma subclasse da classe ***Throwable***, que possui duas subclasses diretas: (i) ***Exception*** e (ii) ***Error***
- Uma abordagem simples para tratar uma exceção é exibir uma mensagem relatando o erro ocorrido e retornar para quem chamou ou finalizar o programa, como no exemplo abaixo:

```
int divisao (int a, int b) {
  try {
    if (b == 0) throw new Exception ("Divisao por zero");
    return (a/b);
  }
  catch (Exception objeto) {
    System.out.println ("Erro:" + objeto.getMessage());
    return (0);
  }
}
```

Exceções

- O comando **try** trata uma exceção que tenha sido disparada em seu interior por um comando **throw**
- O comando **throw** instancia o objeto que representa a exceção e o envia para ser capturado pelo trecho de código que vai tratar a exceção.
- O comando **catch** captura a exceção e fornece o tratamento adequado.
- Uma abordagem mais elaborada para tratar uma exceção é separar o local onde a exceção é tratada do local onde ela ocorreu.
- Importante pelo fato de que um trecho de código em um nível mais alto pode possuir mais informação para decidir como melhor tratar a exceção.

Exceções

- No exemplo abaixo a exceção não é tratada no local onde ela ocorreu, e esse fato é explicitamente indicado pelo comando **t**hrows

```
int divisao (int a, int b) throws {
  if (b == 0) throw new Exception ("Divisao por zero");
  return (a/b);
}
```

- Considerando que o método $divisao$ está inserido em uma classe chamada $Divisao$, o trecho de código abaixo ilustra como capturar o objeto exceção que pode ser criado no método:

```
Divisao d = new Divisao ();
try {
  d.divisao (3, 0);
}
catch(Exception objeto) {
  System.out.println("Erro:"+objeto.getMessage())
}
```

Saída de Dados

- Os tipos primitivos e objetos do tipo $String$ podem ser impressos com os comandos

  System.out.print (var);
  System.out.println (var);

- O método $print$ deixa o cursor na mesma linha e o método $println$ move o cursor para a próxima linha de saída.

Entrada de Dados

- Todo programa em Java que tenha leitura de dados tem de incluir o comando no início do programa

  import java.io.*;

- Método para ler do teclado uma cadeia de caracteres terminada com a tecla *Enter*:

```
public static String getString () throws
IOException {
   InputStreamReader inputString = new
InputStreamReader (System.in);
   BufferedReader buffer = new BufferedReader
(inputString);
   String s = buffer.readLine (); return s;
}
```

- Método para realizar a entrada de um caractere a partir do teclado:

```
public static char getChar () throws IOException {
  String s = getString ();
  return s.charAt (0);
}
```

- Se o que está sendo lido é de outro tipo, então é necessário realizar uma conversão

Diferenças entre Java e C++

- A maior diferença entre Java e C++ é a ausência de apontadores em Java(não utiliza apontadores explicitamente).
- Java trata tipos de dados primitivos, tais como **int**, **double** e **float**, de forma diferente do tramento dado a objetos.
- Em Java, uma referência pode ser vista como um apontador com a sintaxe de uma variável.
- A linguagem C++ tem variáveis referência, mas elas têm de ser especificadas de forma explícita com o símbolo &.
- Outra diferença significativa está relacionada com o operador de atribuição (=):
  - C++: após a execução de um comando com operador (=), passam a existir dois objetos com os mesmos dados estáticos.
  - Java: após a execução de um comando com operador (=), passam a existir duas variáveis que se referem ao mesmo objeto.

Diferenças entre Java e C++

- Em Java e em C++ os objetos são criados utilizando o operador **new**, entretanto, em Java o valor retornado é uma referência ao objeto criado, enquanto em C++ o valor retornado é um apontador para o objeto criado.
- A eliminação de apontadores em Java tem por objetivo tornar o *software* mais seguro, uma vez que não é possível manipular o endereço de $conta1$, evitando que alguém possa acidentalmente corromper o endereço.
- Em C++, a memória alocada pelo operador **new** tem de ser liberada pelo programador quando não é mais necessária, utilizando o operador **d**elete.
- Em Java, a liberação de memória é realizada pelo sistema de forma transparente para o programador (**coleta de lixo**, do inglês *garbage collection*).

Diferenças entre Java e C++

- Em Java, os objetos são passados para métodos como referências aos objetos criados, entretanto, os tipos primitivos de dados em Java são sempre passados por valor
- Em C++ uma passagem por referência deve ser especificada utilizando-se o &, caso contrário, temos uma passagem por valor.
- No caso de tipos primitivos de dados, tanto em Java quanto em C++ o operador de igualdade (==) diz se duas variáveis são iguais.
- No caso de objetos, em C++ o operador diz se dois objetos contêm o mesmo valor e em Java o operador de igualdade diz se duas referências são iguais, isto é, se apontam para o mesmo objeto.
- Em Java, para verificar se dois objetos diferentes contêm o mesmo valor é necessário utilizar o método $equals$ da classe $Object$ (O programador deve realizar a sobrescrita desse método para estabelecer a

Diferenças entre Java e C++

- Em C++ é possível redefinir operadores como $+$, $-$, $*$, $=$, de tal forma que eles se comportem de maneira diferente para os objetos de uma classe particular, mas em Java, não existe sobrecarga de operadores.
- Por questões de eficiência foram mantidos diversos tipos primitivos de dados, assim variáveis declaradas como um tipo primitivo em Java permitem acesso direto ao seu valor, exatamente como ocorre em C++.
- Em Java, o tipo **boolean** pode assumir os valores **false** ou **true** enquanto em C++ os valores inteiros 0 e 1
- O tipo **byte**não existe em C++.
- O tipo **char** em Java é sem sinal e usa dois bytes para acomodar a representação
- O tipo **Unicode** de caracteres acomoda caracteres internacionais de linguas tais como chinês e japonês.

Diferenças entre Java e C++

- O tipo **short** tem tratamento parecido em Java e C++.
- Em Java, o tipo **int** tem sempre 32 *bits*, enquanto em C++ de tamanho, dependendo de cada arquitetura do computador onde vai ser executado.
- Em Java, o tipo **float** usa o sufixo F (por exemplo, 2.357F) enquanto o tipo **d**ouble não necessita de sufixo.
- Em Java, o tipo **long** usa o sufixo L (por exemplo, 33L); quaisquer outros tipos inteiros não necessitam de sufixo.