Programação OO
Tour Rápido em Java

ANIMAlab

Introdução

Introdução

- Vamos desenvolver um **Sistema Fábrica de Veículos.**
  - Carros
  - Motos
  - Caminhões

Introdução

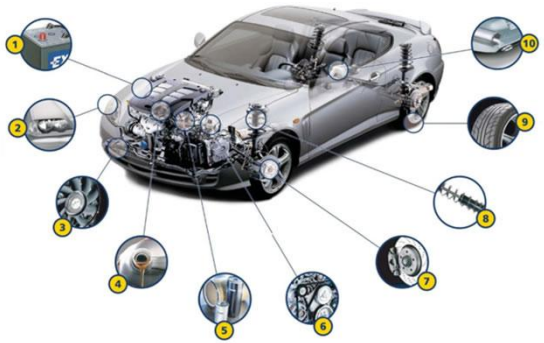
- Vamos desenvolver um **Sistema Fábrica de Veículos.**
  - Inicialmente, o sistema irá fabricar somente carros.

1. **Criar uma classe principal**
   a. Método principal (main)
2. **Criar a classe Carro**
3. **Criar objetos do tipo Carro**

Estudo de Caso:
Sistema Fábrica de Veículos

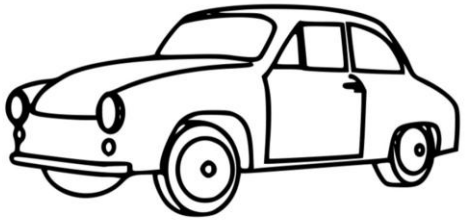
Estudo de Caso: Sistema Fábrica de Veículos

CLASSE CARRO



Para ter um objeto Carro no nosso sistema, **precisamos primeiro especificar** (definir) o que é um carro

OBJETO CARRO



1. **Propriedades**
2. **Comportamentos**

Estudo de Caso: Sistema Fábrica de Veículos

1. Criar uma a classe principal
2. **Criar objetos Carros**
    a. **Primeiro, precisamos criar a classe Carro**

Classes - Definição

Classes - Definição

Conceitos

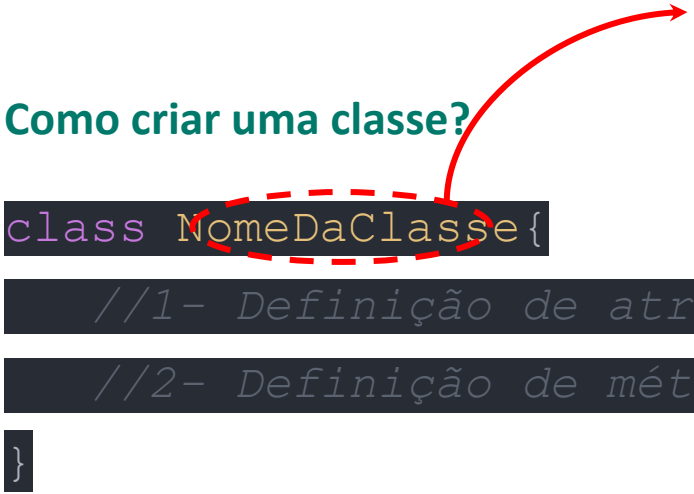
- Define um tipo (modelo) de objeto.
- Uma classe é constituída principalmente por:
  - **Propriedades → Atributos**
  - **Comportamentos → Métodos**

Classes - Definição

Regras da Nomenclatura da Classe

1. O nome da classe deve ser um substantivo.
2. O nome da classe deve iniciar com letra maiúscula.
3. Se o nome da classe for composto, então a letra inicial de cada nome deve ser maiúscula.
4. O nome do arquivo da classe deve ser o mesmo nome da classe.

Como criar uma classe?



```
class NomeDaClasse{
    //1- Definição de atributos
    //2- Definição de métodos
}
```

Classes - Atributos

Como criar uma classe?

```
class NomeDaClasse{
    //1- Definição de atributos
    //2- Definição de métodos
}
```

Atributos → Variáveis de Instância

- Também chamados de **Variáveis de Instância.**
- É definido na classe.
- Podem ser:
  - Tipos Primitivos.
  - Outros Objetos → objetos podem ser compostos por outros objetos

Classes - Atributos

Regras da Nomenclatura das Variáveis de Instância

Como criar uma classe?

```
class NomeDaClasse{
    //1- Definição de atributos
    //2- Definição de métodos
}
```

Atributos

- Sempre a primeira letra deve ser **minúscula**.
- Se o nome for composto, a primeira letra de das palavras **a partir da segunda devem ser maiúsculas**.

Classes - Atributos

Regras da Nomenclatura das Variáveis de Instância

Como criar uma classe?

```
class NomeDaClasse{
    //1- Definição de atributos
    //2- Definição de métodos
}
```

Atributos

String nome;

int idade;

long id;

double salario;

Classes - Métodos

Como criar uma classe?

```
class NomeDaClasse{
    //1- Definição de atributos
    //2- Definição de métodos
}
```

Métodos

- Indica uma ação sobre os **dados das variáveis de instância** ou d**ados recebidos pelos parâmetros**.

Classes - Métodos

Como criar uma classe?

```
class NomeDaClasse{
    //1- Definição de atributos
    //2- Definição de métodos
}
```

Métodos

Uma definição de método é composta de duas partes distintas:

- **Cabeçalho do método → Assinatura**

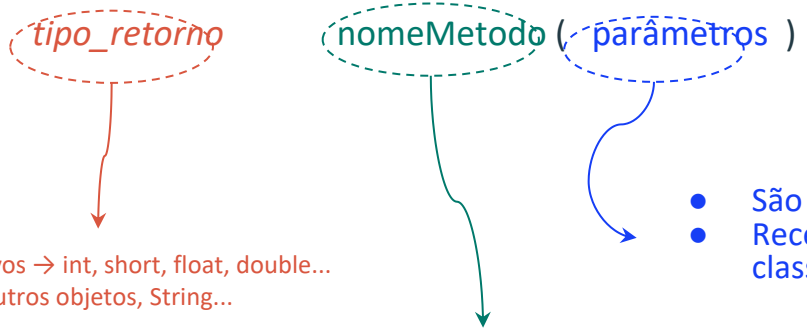
  tipo_retorno nomeMetodo(parâmetros)

- **Corpo de código** → implementação cercado por colchetes {...}.

Classes - Métodos

Métodos

- **Cabeçalho do método → Assinatura**



Tipos primitivos → int, short, float, double...
Objetos → Outros objetos, String...

- São variáveis.
- Recebem valores externos a classe.

- A 1ª letra é **minúscula**.
- Se o nome for composto, então a primeira letra das palavras posteriores à primeira devem ser **maiúsculas**

Estudo de Caso:
Sistema Fábrica de Veículos

Estudo de Caso: Sistema Fábrica de Veículos

1. Criar uma a classe principal
2. **Criar objetos Carros**
    a. **Primeiro, precisamos criar a classe Carro**

1º Passo

Precisamos definir quais os atributos e métodos importantes para definir um carro no sistema.

Estudo de Caso: Sistema Fábrica de Veículos

Atributos de um Carro

- Cor
- Nome
- Modelo
- Ano
- Número de Portas
- Motor
- Ligado ou desligado
- Em movimento ou parado

Comportamentos de um Carro

- Ligar ou desligar
- Mover
- Parar

Estudo de Caso: Sistema Fábrica de Veículos

1. Criar uma a classe principal
2. **Criar objetos Carros**
    a. **Primeiro, precisamos criar a classe Carro**

2º Passo

Tipo de dados de cada atributos.

Estudo de Caso: Sistema Fábrica de Veículos

Atributos de um Carro

- Cor → **cadeia de caracteres**
- Nome → **cadeia de caracteres**
- Modelo → **cadeia de caracteres**
- Ano → **valor numérico**
- Número de Portas → **valor numérico**
- Motor → **objeto**
- Ligado ou desligado → **valor lógico**
- Em movimento ou parado → **valor lógico**

Comportamentos de um Carro

- Ligar ou desligar
- Mover
- Parar

```java
public class Carro {
    // Atributos (variáveis de instância)
    String cor;
    String nome;
    String modelo;
    int ano;
    int numeroPortas;
    boolean ligado;
    boolean emMovimento;

    // Comportamentos (métodos)
    void ligar() {
        if (!ligado) {
            ligado = true;
            System.out.println(nome + " ligado.");
        }
    }

    void desligar() {
        if (emMovimento) {
            System.out.println("Pare antes de desligar!");
            return;
        }
        if (ligado) {
            ligado = false;
            System.out.println(nome + " desligado.");
        }
    }

    void mover() {
        if (!ligado) {
            System.out.println("Não posso mover: carro está
            desligado.");
            return;
        }
        emMovimento = true;
        System.out.println(nome + " em movimento.");
    }

    void parar() {
        if (emMovimento) {
            emMovimento = false;
            System.out.println(nome + " parado.");
        }
    }
}
```

Estudo de Caso: Sistema Fábrica de Veículos

Atributos de um Carro

- Motor → **objeto**
  - cilindradas → **valor ponto flutuante**
  - potência → **valor ponto flutuante**
  - cavalos → **valor numérico**
  - tipo de combustível → **cadeia de caracteres**

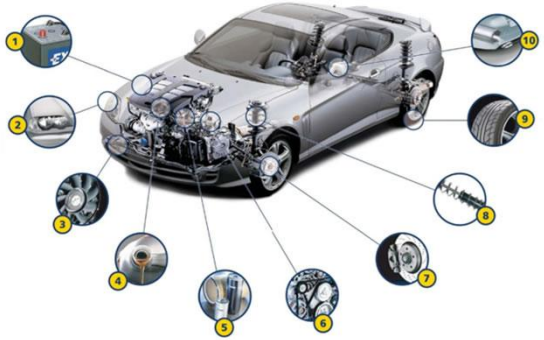
```
public class Motor {
    float cilindradas;
    float potencia;
    int cavalos;
    String combustivel;
}
```

Estudo de Caso: Sistema Fábrica de Veículos

1. Criar uma a classe principal
2. **Criar objetos Carros**
    a. Primeiro, precisamos criar a classe Carro.
    b. **Segundo, precisamos criar objetos carros.**

Estudo de Caso: Sistema Fábrica de Veículos

CLASSE CARRO



Agora podemos **criar vários carros** de acordo com o nosso modelo de carro definido na classe.

Instanciar Objetos

Instanciação de Objetos

Instanciação de Objetos

O ato de criar objetos é chamado de **instanciar objetos.**

Instanciação de Objetos

- A palavra-chave **new** instancia um novo objeto em memória a partir de uma classe**.**

  ```
  new NomeClasse();
  ```

- Para que esse novo objeto possa ser referido, ele precisa ser atribuído a uma variável apropriada (**do mesmo tipo**).

  ```
  NomeClasse nomeVariavel = new NomeClasse();
  ```

Acesso a Objetos e Envio de Mensagens

Acesso a Objetos e Envio de Mensagens

Uma vez que os **atributos e métodos de um objeto são considerados suas características**, eles são acessados diretamente por meio do operador de qualificação **“.”** com respeito a um objeto próprio.

Acesso a Objetos e Envio de Mensagens

```
Carro carro = new Carro();
carro.nome = "Uno";
carro.ano = 2015;
carro.numeroPortas = 4

carro.ligar();
carro.desligar();
```

```java
public class App {
    public static void main(String[] args) {
        // Instanciação de objetos (new)
        Carro uno = new Carro();
        uno.nome = "Uno";
        uno.cor = "Prata";
        uno.modelo = "Attractive";
        uno.ano = 2015;
        uno.numeroPortas = 4;

        Carro ka = new Carro();
        ka.nome = "Ka";
        ka.cor = "Vermelho";
        ka.modelo = "SE";
        ka.ano = 2018;
        ka.numeroPortas = 4;

        // Envio de mensagens (operador .)
        uno.ligar();
        uno.mover();
        uno.parar();
        uno.desligar();
        ka.ligar();
        ka.mover();
        ka.parar();
        ka.desligar();

        // Resumo simples
        System.out.printf("Resumo: %s (%s/%d) ligado? %b%n",
                uno.nome, uno.cor, uno.ano, uno.ligado);
        System.out.printf("Resumo: %s (%s/%d) ligado? %b%n",
                ka.nome, ka.cor, ka.ano, ka.ligado);
    }
}
```

Estudo de Caso:
Sistema Fábrica de Veículos

Estudo de Caso: Sistema Fábrica de Veículos

Vamos criar duas fábricas de carros por meio de criação de classes.

1. Fábrica de carros da Fiat.
    - Classe FabricaFiat
2. Fábrica de carros da Ford.
    - Classe FabricaFord

Estudo de Caso: Sistema Fábrica de Veículos

As duas fábricas terão um método com a seguinte assinatura:

```
public Carro fabricarCarro(String cor, int numeroPortas, float cilindradas, float tipo de combustível)
```

Ocultar Informações dos Objetos

Ocultar Informações dos Objetos

Embora o acesso direto aos atributos do objeto seja permitido, **não é ideal.**

- Qualquer mudança na representação do objeto se propaga para o código dependente, resultando em alto custo de manutenção de software.
- Uma prática comum de programação orientada a objetos é **ocultar informações**.

Ocultar Informações dos Objetos

Tornar as **representações (atributos e métodos)** de objetos inacessíveis aos clientes.

- As modificações nas representações de objetos não se propaguem excessivamente.
- Reduzindo o custo de manutenção de software.

Especificadores de Restrição

Especificadores de Restrição

Para limitar o acesso a representações de objeto, usa-se **especificadores de restrição.**

public → permite acessar diretamente elementos da classe **sem restrição.**

private → permite acessar diretamente elementos da classe **somente dentro da classe.**

Métodos de Acesso e Métodos Modificadores

Estudo de Caso: Sistema Fábrica de Veículos

Visto que **os especificadores de restrição** na definição de classe **ocultam** a representação interna (atributos e métodos), alguns componentes **não são mais acessíveis** para uso externo

Para disponibilizar os **atributos ocultos da classe**, são necessários **métodos Modificadores e de Acesso.**

Métodos de Acesso e Métodos Modificadores

Há dois tipos:

1. **Método get** → retorna uma variável de instância
2. **Método set** → atualiza uma variável de instância

Métodos de Acesso e Métodos Modificadores

Nomenclatura do método get

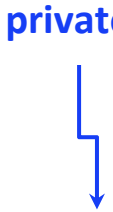
```
public tipoVariavelInstancia getNomeVariavel(){
        return this.variavelInstancia
}
```

Nomenclatura do método set

```
public void setNomeVariavel(tipoVariavelInstancia nomeParametro){
    this.variavelInstancia = nomeParametro;
}
```

Encapsulamento

Encapsulamento



A ação de **ocultar as variáveis de instância** de uma classe e **disponibilizar os métodos modificadores e de acesso** para cada uma delas é chamado de **Encapsulamento**.

Inicialização e Construtores

Inicialização e Construtores

Depois de instanciar um objeto, normalmente deseja **inicializá-lo.**

- Adicionar valores padrões nos atributos dos objetos.
- Chamar algum comportamento do próprio objeto.

Para inicializar um objeto, usa-se um método especial, chamado de **Método Construtor.**

Inicialização e Construtores

Há duas formas de nomenclatura.

1ª Forma - Método Construtor

```
public NomeClasse(){
    //Outras instruções
}
```

Construtor Padrão é aquele que não possui parâmetros!

2ª Forma - Método Construtor

```
public
NomeClasse(listaParâmetros){
    //Outras instruções
}
```

Inicialização e Construtores

OBSERVAÇÕES IMPORTANTES

1. Os construtores **não possuem retorno.**
2. Os construtores são chamados **automaticamente** logo após à instanciação.
3. Se o desenvolvedor não cria nenhum construtor dentro da classe, então o compilador irá criar automaticamente o **construtor padrão** da classe.

```java
public class Motor {
    private float cilindradas;
    private float potencia;
    private int cavalos;
    private String combustivel;

    public Motor() {}

    public Motor(float cilindradas, float potencia, int
    cavalos, String combustivel) {
        this.cilindradas = cilindradas;
        this.potencia = potencia;
        this.cavalos = cavalos;
        this.combustivel = combustivel;
    }

    public float getCilindradas() {
        return cilindradas;
    }
    public void setCilindradas(float cilindradas) {
        this.cilindradas = cilindradas;
    }
```

```java
    public float getPotencia() {
        return potencia;
    }
    public void setPotencia(float potencia) {
        this.potencia = potencia;
    }

    public int getCavalos() {
        return cavalos;
    }
    public void setCavalos(int cavalos) {
        this.cavalos = cavalos;
    }

    public String getCombustivel() {
        return combustivel;
    }
    public void setCombustivel(String combustivel) {
        this.combustivel = combustivel;
    }
}
```

```java
public class Carro {
    // Encapsulamento: tudo private + getters/setters
    private String cor;
    private String nome;
    private String modelo;
    private int ano;
    private int numeroPortas;
    private boolean ligado;
    private boolean emMovimento;
    private Motor motor; // composição: Motor é um objeto

    // Construtor padrão
    public Carro() {
        this.cor = "Branco";
        this.nome = "Genérico";
        this.modelo = "Base";
        this.ano = 2000;
        this.numeroPortas = 4;
        this.ligado = false;
        this.emMovimento = false;
    }

    // Construtor com parâmetros
    public Carro(String cor, String nome, String modelo,
    int ano, int numeroPortas, Motor motor) {
        this.cor = cor;
        this.nome = nome;
        this.modelo = modelo;
        this.ano = ano;
        this.numeroPortas = numeroPortas;
        this.motor = motor;
    }
```

```java
public void ligar() {
    if (!ligado) {
        ligado = true;
        System.out.println(nome + " ligado.");
    }
}

public void desligar() {
    if (emMovimento) {
        System.out.println("Pare antes de desligar!");
        return;
    }
    if (ligado) {
        ligado = false;
        System.out.println(nome + " desligado.");
    }
}

public void mover() {
    if (!ligado) {
        System.out.println("Não posso mover: carro está
        desligado.");
        return;
    }
    emMovimento = true;
    System.out.println(nome + " em movimento.");
}

public void parar() {
    if (emMovimento) {
        emMovimento = false;
        System.out.println(nome + " parado.");
    }
}
```

```java
    // Getters/Setters
    public String getCor() { return cor; }
    public void setCor(String cor) { this.cor = cor; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getModelo() { return modelo; }
    public void setModelo(String modelo) { this.modelo =
    modelo; }

    public int getAno() { return ano; }
    public void setAno(int ano) { this.ano = ano; }

    public int getNumeroPortas() { return numeroPortas; }
    public void setNumeroPortas(int numeroPortas) { this.
    numeroPortas = numeroPortas; }

    public boolean isLigado() { return ligado; }
    public boolean isEmMovimento() { return emMovimento; }

    public Motor getMotor() { return motor; }
    public void setMotor(Motor motor) { this.motor =
    motor; }
}
```

```java
public class FabricaFiat {
    public Carro fabricarCarro(String cor, int numeroPortas,
                               float cilindradas, float
                               potencia, String
                               combustivel) {
        Motor m = new Motor(cilindradas, potencia, (int)
        potencia, combustivel);
        Carro c = new Carro(cor, "Uno", "Way", 2016,
        numeroPortas, m);
        c.ligar();
        return c;
    }
}
```

```java
public class FabricaFord {
    public Carro fabricarCarro(String cor, int numeroPortas,
                               float cilindradas, float
                               potencia, String
                               combustivel) {
        Motor m = new Motor(cilindradas, potencia, (int)
        potencia, combustivel);
        Carro c = new Carro(cor, "Ka", "SE", 2018,
        numeroPortas, m);
        c.ligar();
        return c;
    }
}
```

```java
public class App {
    public static void main(String[] args) {
        FabricaFiat fiat = new FabricaFiat();
        FabricaFord ford = new FabricaFord();

        Carro uno = fiat.fabricarCarro("Prata", 4, 1.0f,
        75f, "Flex");
        Carro ka  = ford.fabricarCarro("Vermelho", 4, 1.5f,
        110f, "Gasolina");

        uno.mover(); uno.parar(); uno.desligar();
        ka.mover();  ka.parar();  ka.desligar();

        System.out.printf("Resumo Fiat: %s %s %d, %d
        portas, motor %.1fL (%s)%n",
                uno.getNome(), uno.getModelo(), uno.getAno
                (),
                uno.getNumeroPortas(), uno.getMotor().
                getCilindradas(), uno.getMotor().
                getCombustivel());

        System.out.printf("Resumo Ford: %s %s %d, %d
        portas, motor %.1fL (%s)%n",
                ka.getNome(), ka.getModelo(), ka.getAno(),
                ka.getNumeroPortas(), ka.getMotor().
                getCilindradas(), ka.getMotor().
                getCombustivel());
    }
}
```

Tipos de Dados de Variáveis

Tipos de Dados de Variáveis

Tipos Primitivos

- Valores simples
  - int
  - short
  - float
  - double
  - long
  - boolean
  - char

Tipos por Referência

- Armazena as localizações de objetos na memória do computador. Dizemos que uma variável de referência referencia um objeto no programa.
- Objetos que são referenciados podem conter muitas variáveis de instância.

Tipos de Dados de Variáveis

Tipos por Referência - NOTAS

- Objetos que são referenciados podem conter muitas variáveis de instância.
- Variáveis de instância de tipo por referência, se não forem inicializadas explicitamente, o são por **padrão para o valor null.**
- Para chamar métodos em um objeto, você precisa de uma referência a ele.

Blocos de Instruções

Blocos de Instruções

- Um blocos, indicado por {}, pode ocorrer **em qualquer local** onde uma declaração seja válida.
- Com exceção do bloco da classe, é considerada uma construção sequencial, pois o **grupo de instruções é tratado como uma única instrução ou unidade de processamento.**
- Um bloco pode ser usado onde **devem conter mais de uma instrução.**

Blocos de Instruções

Exemplos

- **Bloco da classe**
- **Bloco do método**
- **Bloco da condição if-else**

```
class App{
    public void metodoQualquer() {
     if(1 == 5){
            //Bloco da condição if-then
        }
        else{
            //Bloco da condição if-else
        }
    }
}
```

Variáveis de Instância

Variáveis de Instância

- São variáveis que são **definidas no bloco de instruções da classe**.

NOTAS

- Os valores das variáveis de instância **pertencem aos objetos**.
- As variáveis de instância de uma classe **somente podem ser usadas** quando se instancia objetos dessa classe.
- Objetos de um mesmo tipo possuem as mesmas variáveis de instância definidas na classe.
- O valor de uma variável de instância de um objeto é **independente dos outros objetos**.

Variáveis e Métodos Estáticos

Variáveis e Métodos Estáticos

- São variáveis ou métodos que são **definidos no bloco de instruções da classe com o qualificador static**.

Observações

- Variáveis estáticas também são chamadas de **variáveis de classe**.
- Os valores das variáveis estáticas pertencem à **classe em que foram definidas**.
- Objetos de um mesmo tipo possuem as mesmas variáveis estáticas definidas na classe.
- O valor de uma variável estática é **compartilhado entre todos os objetos de uma mesma classe**.

Variáveis e Métodos Estáticos

Observações

- Os métodos estáticos não exigem que tenha objeto (this) para serem invocados.
- Os métodos estáticos podem ser invocados diretamente pela classe.

Variáveis Locais

Variáveis Locais

- Dentro de um bloco de instruções, pode-se **definir variáveis.**
- Variáveis declaradas em qualquer bloco de instrução diferente do Bloco da classe são chamadas de **Variáveis Locais**.
- Essas variáveis só **podem ser utilizadas a partir da declaração e dentro do bloco que pertencem.**

Variáveis de Parâmetro

Variáveis de Parâmetro

- São variáveis definidas **na assinatura de um método.**

NOTAS

- Variáveis de Parâmetro **somente podem ser utilizadas no método em que foram definidas**.
- Os valores que são passados para as variáveis de parâmetro são chamados de **Argumentos.**

```java
public class Carro {
    // Instância (cada objeto tem a sua)
    private int kmRodados = 0;

    // Estática (pertence à classe; compartilhada)
    private static int totalProduzidos = 0;

    private String nome;

    // Construtor: incrementa contador estático
    public Carro(String nome) {
        this.nome = nome;        // parâmetro (escopo: corpo do método construtor)
        totalProduzidos++;       // acesso estático dentro da classe
    }

    public void rodar(int km) {  // 'km' é PARÂMETRO
        // Variável LOCAL: escopo é do ponto da declaração até o fim deste bloco
        int antes = kmRodados;
        if (km < 0) {
            System.out.println("Ignorado: km negativo.");
            return;
        }
        kmRodados += km;
        System.out.printf("%s rodou de %dkm para %dkm%n", nome, antes, kmRodados);
    }

    public String getNome() { return nome; }
    public int getKmRodados() { return kmRodados; }

    // Método estático (pode ser chamado pela classe, sem objeto)
    public static int getTotalProduzidos() { return totalProduzidos; }
}
```

Escopo das declarações

Escopo das declarações

O escopo de uma declaração é a parte do programa que pode referenciar a entidade declarada pelo seu nome.

Regras Básicas de Escopo

1. O escopo de uma **variável de parâmetro** é o corpo do método em que a declaração aparece.
2. O escopo de uma **variável local** é do ponto em que a declaração aparece até o final do bloco que foi declarada.

Escopo das declarações

Regras Básicas de Escopo

3. O escopo de uma **variável local** que aparece na seção de inicialização do cabeçalho de **laço for** é o corpo do **laço for** e as outras expressões no cabeçalho.
4. O escopo de um método ou atributo é o corpo da classe. Isso permite que os métodos de instância de uma classe usem os campos e outros métodos da classe.

```java
public class EscopoDemo {
    public static void mostrarBlocos() {
        // Bloco externo
        int x = 10; // variável LOCAL do bloco externo
        {
            // Bloco interno
            int y = 20; // só existe aqui dentro
            System.out.println("Dentro do bloco interno: x=" + x + ", y=" + y);
        }
        // System.out.println(y); // ERRO de escopo: y não existe mais aqui
        System.out.println("Fora do bloco interno: x=" + x);
    }

    public static void mostrarFor() {
        for (int i = 0; i < 3; i++) { // 'i' tem escopo do for
            System.out.println("i = " + i);
        }
        // System.out.println(i); // ERRO: 'i' não existe fora do for
    }
}
```

```java
public class App {
    public static void main(String[] args) {
        // Instância vs Estático
        Carro a = new Carro("Uno");
        Carro b = new Carro("Ka");

        a.rodar(10);  // parâmetro = 10; 'antes' é local dentro de rodar
        b.rodar(25);

        System.out.printf("%s total: %dkm%n", a.getNome(), a.getKmRodados());
        System.out.printf("%s total: %dkm%n", b.getNome(), b.getKmRodados());

        // Acesso estático: pela classe
        System.out.println("Total de carros produzidos:"+Carro.getTotalProduzidos());

        // Blocos e escopo (locais)
        EscopoDemo.mostrarBlocos();
        EscopoDemo.mostrarFor();
    }
}
```

Exercícios

Exercícios

1. Defina uma classe **Quadrado** com o comprimento do lado como variável de instância. Inclua um construtor adequado e métodos para ampliar uma instância e calcular sua área.
2. Usando a classe **Quadrado** do exercício 1, crie dez quadrados de tamanhos aleatórios e encontre a soma de suas áreas.
3. Adicione a funcionalidade para que um objeto **Quadrado** se desenhe usando caracteres ASCII (um quadrado de ‘*’, ou de ‘@’, ou de qualquer caracter)
4. Crie uma classe que utilize a classe **Quadrado** como atributo (conforme fizemos o Carro que possui como atributo o Motor)
5. Escreva um programa (Aplicativo) que use essas classes.