



Características das Linguagens de Programação I

Conteúdo: Métodos estáticos, relações e Interfaces

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Métodos estáticos

Atributos e métodos estáticos

- Os atributos e métodos estáticos **pertencem** à **classe** em si, e não a uma instância específica dessa classe.
- Eles podem ser acessados **sem** a **necessidade** de **criar** um **objeto** da classe.
- **Atributos estáticos** são compartilhados por todas as instâncias da classe.
 - Se uma instância modifica o valor do atributo estático, todas as outras instâncias vêem essa mudança
- **Métodos estáticos** podem ser chamados diretamente pela classe, sem precisar de uma instância da classe.
 - Eles não podem acessar diretamente atributos ou métodos não estáticos (de instância), pois pertencem à classe e não a uma instância específica.

Atributos estático: exemplo

- Um atributo estático é útil para **representar propriedades** que **são comuns a todas** as instâncias de uma classe, como uma contagem de quantas instâncias foram criadas.

```
public class Contador {  
    // Atributo estático  
    public static int totalObjetos = 0;  
  
    // Construtor  
    public Contador() {  
        totalObjetos++;  
    }  
    // mostrar a contagem de objetos  
    public static void mostrarContagem() {  
        System.out.println("Total de objetos criados: " + totalObjetos);  
    }  
}
```

Atributos estático: exemplo (cont.)

-

```
public static void main(String[] args) {  
    Contador obj1 = new Contador();  
    Contador obj2 = new Contador();  
    Contador obj3 = new Contador();  
  
    Contador.mostrarContagem();  
}
```

// Acessando o atributo
estático diretamente pela
classe

Método estático: exemplo

- Um método estático pode ser útil para utilitários e operações que não precisam de uma instância de classe.

```
public class MatematicaUtil {  
    // Método estático  
    public static int soma(int a, int b) {  
        return a + b;  
    }  
}
```

```
public static void main(String[] args) {  
    // Chamando o método estático diretamente pela classe  
    int resultado = MatematicaUtil.soma(5, 10);  
    System.out.println("Resultado da soma: " + resultado);  
}
```


Atributos e Métodos estáticos: exemplo

- Usando o Método e Atributo Estáticos

```
public class Banco {  
    // Atributo estático que mantém a taxa de juros padrão para todos os clientes  
    public static double taxaDeJuros = 5.0;  
  
    // Método estático que calcula o montante de juros sobre um valor  
    public static double calcularJuros(double valor, int anos) {  
        return valor * Math.pow((1 + taxaDeJuros / 100), anos);  
    }  
}
```

Atributos e Métodos estáticos: exemplo (cont.)

```
public static void main(String[] args) {  
    // Acessando o método estático diretamente pela classe  
    double montante = Banco.calcularJuros(1000, 2);  
    System.out.println("Montante com juros: " + montante);  
  
    // Alterando a taxa de juros  
    Banco.taxaDeJuros = 6.0;  
    double novoMontante = Banco.calcularJuros(1000, 2);  
    System.out.println("Montante com nova taxa de juros: " + novoMontante);  
}
```


Relacionamentos entre classes

Associação

- Associação é uma relação entre duas classes que permite que uma classe use os métodos e atributos de outra.
- A associação pode ser unidirecional (apenas uma classe conhece a outra) ou bidirecional (ambas as classes se conhecem).

Associação: Exemplo

```
public class Pessoa {  
    private String nome;  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return nome;  
    }  
}
```

```
public class Carro {  
    private Pessoa proprietario; // Associação com a  
                                // classe Pessoa  
  
    public Carro(Pessoa proprietario) {  
        this.proprietario = proprietario;  
    }  
    public void mostrarProprietario() {  
        System.out.println("Proprietário: " +  
                             proprietario.getNome());  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa("João");  
        Carro carro = new Carro(pessoa);  
        carro.mostrarProprietario();  
    }  
}
```

Composição

- Composição é um tipo mais forte de associação, onde uma classe contém uma instância de outra classe e a vida dessa instância é dependente da classe "pai".
- Se a classe que contém for destruída, as instâncias das classes que ela contém também serão destruídas.
- Em outras palavras, a composição implica uma relação "parte-todo" em que a parte não pode existir sem o todo.

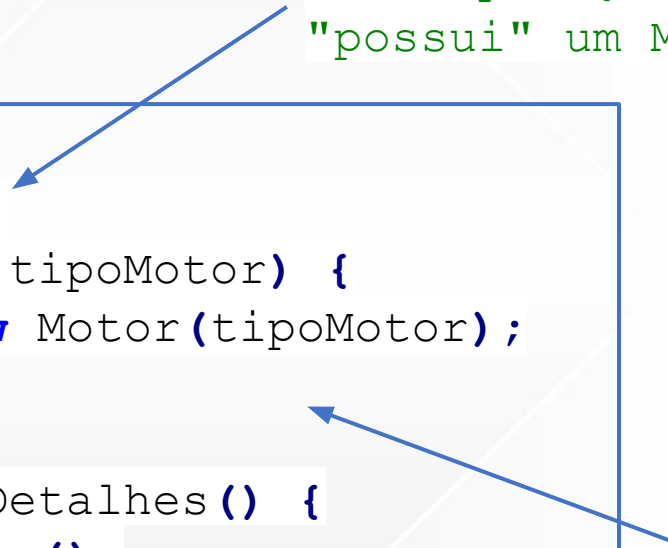
Composição: Exemplo

```
public class Motor {  
    private String tipo;  
  
    public Motor(String tipo) {  
        this.tipo = tipo;  
    }  
    public void mostrarTipo() {  
        System.out.println("Tipo do motor: " + tipo);  
    }  
}
```

Composição: Exemplo (cont.)

```
public class Carro {  
    private Motor motor;  
    public Carro(String tipoMotor) {  
        this.motor = new Motor(tipoMotor);  
    }  
  
    public void mostrarDetalhes() {  
        motor.mostrarTipo();  
    }  
}
```

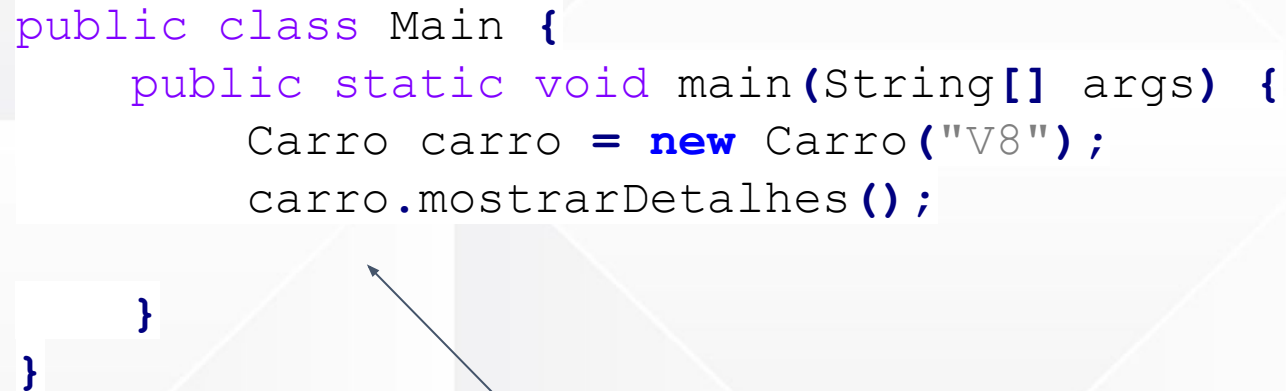
// Composição: Carro
"possui" um Motor



// O motor é criado
junto com o carro

Composição: Exemplo (cont.)

```
public class Main {  
    public static void main(String[] args) {  
        Carro carro = new Carro("V8");  
        carro.mostrarDetalhes();  
    }  
}
```



// Quando o objeto carro for destruído, o motor também será destruído

Interfaces

Interfaces

- Uma interface em Java nada mais é que um **conjunto de declarações de métodos** (nome, tipo de retorno, tipos dos argumentos) **desprovidos de implementação**.
- Cabe ao programador que deseja implementar a interface em questão, providenciar uma implementação destes métodos na classe que ele está desenvolvendo.

Interfaces: declaração

Uma Interface tem uma declaração parecida com uma classe, porém a palavra interface é usada no lugar da palavra class.

```
public interface Nome_da_Interface {  
  
    public tipo retorno método 1 (); // declaração do método 1  
    public tipo_retorno método_2 (); // declaração do método 2  
    ....  
    public tipo_retorno método_n (); // declaração do método n  
  
}
```

Interfaces: utilização

A palavra chave implements é utilizada para indicar que uma classe implementa uma determinada interface.

```
public class Nome Classe implements Nome_da_Interface {  
    //atributos da classe  
    public tipo retorno metodo 1( ) {  
        // implementação do método1  
        return ;  
    }  
    public tipo_retorno metodo_n ( ) {  
        // implementação do método_n  
        return ;  
    }  
}
```

Interfaces: Exemplo 1

Crie uma Interface chamado de **ObjetoGeometrico**, e terá a declaração dos métodos `calculaArea()` e `calculaPerimetro()`.

```
public interface ObjetoGeometrico {  
    public double calcularArea();  
    public double calcularPerimetro();  
}
```

Interfaces: Exemplo 1 (cont.)

Adicionaremos uma classe chamado de **Quadrado** que terá o atributo **lado**. Também adicionaremos os métodos getter e setter para esse atributo


```
public class Quadrado{  
    //atributos da classe  
    private double lado;  
  
    //métodos getter e setters  
    public double getLado(){  
        return this.lado;  
    }  
    public void setLado(double lado){  
        this.lado = lado;  
    }  
}
```

Interfaces: Exemplo 1 (cont.)

Fazemos que a classe **Quadrado** implemente a interface **ObjetoGeometrico**, o que acontece?.

```
public class Quadrado implements ObjetoGeometrico
{
    //atributos da classe
    private double lado;

    //métodos getter e setters
    public double getLado(){
        return this.lado;
    }
    public void setLado(double lado){
        this.lado = lado;
    }
}
```



Interfaces: Exemplo 1 (cont.)

Como a classe **Quadrado** **assina** um **contrato** com **ObjetoGeometrico**, devemos **obrigatoriamente** implementar seus métodos.

```
public class Quadrado implements ObjetoGeometrico {
    private double lado;

    //métodos getter e setters
    public double getLado () {
        return this.lado;
    }
    public void setLado (double lado) {
        this.lado = lado;
    }
    // outros métodos
    @Override
    public double calcularArea () {
        return this.lado*this.lado;
    }
    @Override
    public double calcularPerimetro (); {
        return 4*this.lado;
    }
}
```

Interfaces: Exemplo 1 (cont.)

- Finalmente criamos uma classe de teste

```
public class TesteInterface{  
    public static void main(String args[]){  
        Quadrado q1 = new Quadrado();  
        q1.setLado(10);  
        System.out.println("Area: "+q1.area() + " m2");  
        System.out.println("Perímetro: "+q1.perimetro() + " m");  
    }  
}
```

Interfaces: Exemplo 2

- Considerando o exemplo 1, adicione a classe Retangulo conforme mostrado:

```
public class Retangulo{  
    // Atributos  
    private double ladoMaior;  
    private double ladoMenor;  
  
    // Construtor  
    public Retangulo(double ladoMaior, double ladoMenor){  
        this.ladoMaior = ladoMaior;  
        this.ladoMenor = ladoMenor;  
    }  
    // métodos  
    public double getLadoMaior(){  
        return this.ladoMaior;  
    }  
    public double getLadoMenor(){  
        return this.ladoMenor;  
    }  
}
```

Interfaces: Exemplo 2 (cont.)

- Faça que a classe Retangulo implemente a interface ObjetoGeometrico

```
public class Retangulo implements ObjetoGeometrico {  
    // Atributos  
    private double ladoMaior;  
    private double ladoMenor;  
  
    // Construtor  
    public Retangulo(double ladoMaior, double  
ladoMenor){  
        this.ladoMaior = ladoMaior;  
        this.ladoMenor = ladoMenor;  
    }  
    // métodos  
    public double getLadoMaior(){  
        return this.ladoMaior;  
    }  
    public double getLadoMenor(){  
        return this.ladoMenor;  
    }  
}
```

```
    @Override  
    public double calcularArea(){  
        return this.ladoMaior*this.ladoMenor;  
    }  
    @Override  
    public double calcularPerimetro();{  
        return 2*this.ladoMaior + 2*this.ladoMenor;  
    }  
}
```

Modifique a classe Teste!

Interfaces: Exercício

- Implementar a classe Triangulo equilátero com os atributos base e altura e depois sobre escreva os métodos caculaArea() e calculaPerimetro().
- Faça uma aplicação de teste

Interfaces ou classes abstratas?

- O que foi feito com o uso de interface não poderia ser feito com uma classe abstrata?

```
public interface ObjetoGeometrico {  
  
    public double calcularArea();  
    public double calcularPerimetro();  
  
}
```

```
public abstract class ObjetoGeometrico {  
  
    public abstract double calcularArea();  
    public abstract double calcularPerimetro();  
  
}
```

Interfaces ou classes abstratas?

- Uma interface não é considerada uma Classe e sim uma **Entidade**;
- Não possui implementação, apenas **assinatura**, ou seja, apenas a definição dos seus métodos sem o corpo;
- Todos os métodos são abstratos;
- Seus **métodos** são implicitamente **Públicos e Abstratos**;
- Não há como fazer uma instância de uma Interface e nem como criar um Construtor;
- Funcionam como um tipo de "**contrato**", onde são especificados os atributos, métodos e funções que as classes que implementem essa interface são obrigadas a implementar;
- Permite implementar **Heranças Múltiplas** em Java

Interfaces: Exemplo 3

- Crie um novo projeto e adicione a classe TV.

```
public class TV {  
    private int tamanho;  
    private int canal;  
    private int volume;  
    private boolean ligada;  
  
    public TV(int tamanho)  
    {  
        this.tamanho = tamanho;  
        this.canal = 0;  
        this.volume = 0;  
        this.ligada = false;  
    }  
  
    // Outros construtores e métodos get e set...  
}
```

Interfaces: Exemplo 3 (cont.)

- Adicione a interface ControleRemoto

```
public interface ControleRemoto {  
    public void mudarCanal(int canal);  
    public void aumentarVolume (int taxa);  
    public void diminuirVolume (int taxa);  
    public void ligar();  
    public void desligar();  
}
```

Interfaces: Exemplo 3 (cont.)

- Adicione a classe ModeloTV001 derivada de **TV** e que implementa a interface **ControleRemoto**.

```
public class ModeloTV001 extends TV implements
ControleRemoto
{
    public final String MODELO = "TV001";

    public ModeloTV001(int tamanho) {
        super(tamanho);
    }
}
```

```
@Override
public void desligar() {
    super.setLigada(false);
}
@Override
public void ligar() {
    super.setLigada(true);
}
@Override
public void aumentarVolume(int taxa) {

}
@Override
public void diminuirVolume(int taxa) {

}
@Override
public void mudarCanal(int canal) {

}
}
```

Interfaces: Exemplo 3 (cont.)

- Adicione a classe ModeloX (uma TV mais moderna), filha da **TV** e implementa a interface **ControleRemoto**.

```
public class ModeloX extends TV implements ControleRemoto {  
    public final String MODELO = "TV-X";  
  
    public ModeloX(int tamanho) {  
        super(tamanho);  
    }  
}
```

```
@Override  
public void desligar () {  
    System.out.println("Obrigado por Utilizar a  
Televisão!");  
    super.setLigada (false);  
}  
@Override  
public void ligar () {  
    super.setLigada (true);  
}  
@Override  
public void aumentarVolume (int taxa) {  
  
}  
@Override  
public void diminuirVolume (int taxa) {  
  
}  
@Override  
public void mudarCanal (int canal) {  
  
}  
}
```

Interfaces: Exemplo 3 (cont.)

- Crie uma classe de Teste. Como pode ser visto, ambos possuem a mesma ação que é desligar, porém cada um executa de forma diferente.

```
public class ExemploInterface {  
    public static void main(String[] args) {  
  
        ModeloTV001 tv1 = new ModeloTV001 (21);  
        ModeloX tv2 = new ModeloX (42);  
        tv1.ligar();  
        tv2.ligar();  
        System.out.print("TV1 - modelo " + tv1.MODELO + " está ");  
        System.out.println(tv1.isLigada() ? "ligada" : "desligada");  
        System.out.print("TV2 - modelo " + tv2.MODELO + " está ");  
        System.out.println(tv2.isLigada() ? "ligada" : "desligada");  
  
        // ambas as TVs estão ligadas e vamos desligá-las  
        System.out.println("Desligando modelo " + tv1.MODELO); tv1.desligar();  
        System.out.println("Desligando modelo " + tv2.MODELO); tv2.desligar();  
    }  
}
```

Referências

BIBLIOGRAFIA BÁSICA:

- DEITEL, Harvery M.. Java : como programar. 10ª ed. São Paulo: Pearson - Prentice Hall, 2017.
- BORATTI, Isaías Camilo. Programação Orientada a Objetos em Java : Conceitos Fundamentais de Programação Orientada a Objetos. 1ª ed. Florianópolis: VisualBooks, 2007.
- SIERRA, Kathy; BATES, Bert. Use a Cabeça! Java. 2ª ed. Rio de Janeiro: Alta Books, 2007.

