

Linguagem de Programação Orientada a Objetos I

Herança

Prof. Tales Bitelo Viegas

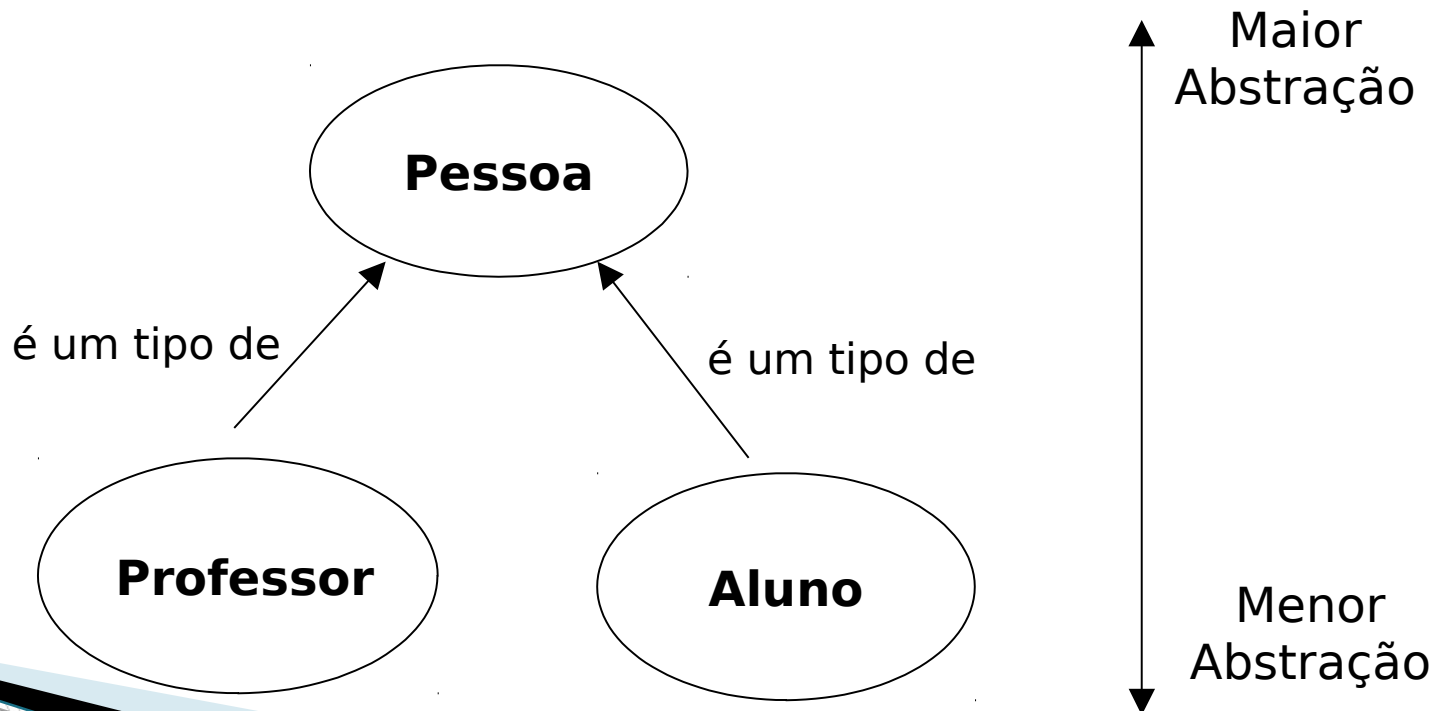
<https://fb.com/ProfessorTalesViegas>

The bottom of the slide features a decorative graphic consisting of several overlapping, wavy lines in shades of gray and light blue, creating a modern, abstract background element.

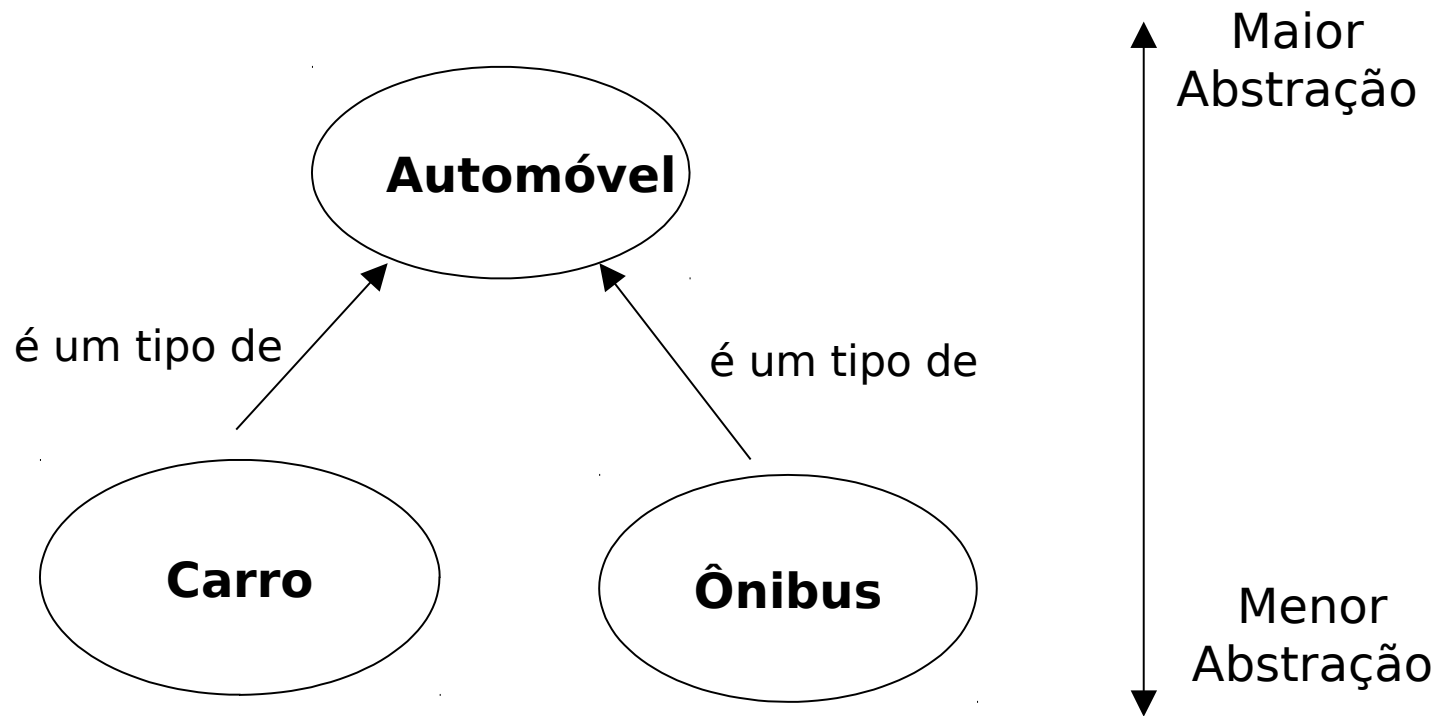
Herança

► Herança

- Diferenças ou variações de uma classe em particular podem ser organizadas de forma mais clara



Exemplo de Herança



Herança

► Herança

- Cada classe em um nível da hierarquia herda as características das classes dos níveis acima
- Este mecanismo facilita o compartilhamento de comportamento comum entre um conjunto semelhante de classes
- Um objeto da subclasse conterá todos os campos e métodos declarados na superclasse mais os campos e métodos declarados na própria subclasse

Herança

- ▶ Herança
 - simples:
 - ▢ objeto herda diretamente de uma classe apenas
 - múltipla:
 - ▢ objeto herda de duas ou mais superclasses
- ▶ JAVA não tem herança múltipla

Exemplo de Herança

```
public class Pessoa {  
    public String nome;  
    public char sexo;  
}
```

```
public class Professor extends Pessoa {  
    public String titulacao;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */  
}
```

```
public class Aluno extends Pessoa{  
    public int matricula;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */  
}
```

Exemplo de Herança

```
public class Automovel {  
    public String placa;  
    public String marca;  
    public String modelo;  
}
```

```
public class Carro extends Automovel {  
    public int numPortas;  
    /* Atributos Herdados  
    public String placa;  
    public String marca;  
    public String modelo; */  
}
```

```
public class Onibus extends Automovel {  
    public int numPassageiros;  
    public int numEmbratur;  
    /* Atributos Herdados  
    public String placa;  
    public String marca;  
    public String modelo; */  
}
```

Modificadores de Acesso

- ▶ Modificadores de Acesso
 - Mostram a visibilidade de um atributo ou método para outras classes
- ▶ Podem ser de três tipos:
 - **public**: qualquer classe ou método pode ter acesso
 - **private**: somente a própria classe pode ter acesso
 - **protected**: somente a própria classe pode ter acesso (e demais do próprio package)

Herança: o que herda ?

- ▶ Subclasse herda diretamente:
 - Atributos e métodos públicos e protegidos (protected)
- ▶ Subclasse NÃO herda diretamente:
 - Atributos e métodos privados (acessados apenas por métodos públicos da classe pai)
 - Construtores
 - Métodos de mesma assinatura (redefine)
 - Atributos de mesmo nome (esconde)

Exemplo de Herança

- ▶ Classe Mãe, Superclasse ou Base
 - Toda classe estende a Classe Object

```
public class Pessoa extends Object {  
    public String nome;  
    public char sexo;  
  
    public Pessoa(){}  
  
    public Pessoa(String nome, char sexo){  
        this.nome = nome;  
        this.sexo = sexo;  
    }  
    public void imprimir(){  
        System.out.println("Nome: " + this.nome);  
        System.out.println("Sexo: " + this.sexo);  
    }  
}
```

Exemplo de Herança

- ▶ Classe Filha, Subclasse ou Derivada

```
public class Aluno extends Pessoa {  
    private int matricula;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */
```

Novo
Atributo



```
    public Aluno(String nome, char sexo, int matricula){  
        this.nome = nome;  
        this.sexo = sexo;  
        this.matricula = matricula;  
    }  
}
```

Novo
Construtor



Redefinição de Métodos

► Redefinição de Métodos

- Novo método substituirá o método da classe base com a mesma assinatura
- Na classe Superclasse (Pessoa):

```
public void imprimir(){  
    System.out.println("Nome: " + nome);  
    System.out.println("Sexo: " + sexo);  
}
```

- Na classe Subclasse (Aluno):

```
public void imprimir(){  
    System.out.println("Nome: " + nome);  
    System.out.println("Sexo: " + sexo);  
    System.out.println("Matricula: " + matricula);  
}
```

Palavras-chave *this* e *super*

- ▶ Palavra-chave *this*
 - É uma referência a própria instância
 - Refere-se ao objeto

```
public class Pessoa {  
    protected String nome;  
    protected char sexo;  
  
    public Pessoa(String nome, char sexo){  
        this.nome = nome;  
        this.sexo = sexo;  
    }  
}
```

Palavras-chave *this* e *super*

- ▶ Palavra-chave *super*
 - Referência ao código da classe pai
 - Realiza acesso aos métodos ancestrais
 - Permitem aumentar a reutilização de código
 - Na classe filha (Aluno):

`@Override`

`public void imprimir(){`

`super.imprimir();`

`System.out.println("Matricula: " + matricula);`

`}`

Acesso ao método
da classe

Ancestral (Pessoa)

```
public void imprimir(){
    System.out.println("Nome: " + nome);
    System.out.println("Sexo: " + sexo);
}
```

Construtores e Herança


► Construtores

- Construtores NÃO são herdados
- Porém um construtor da superclasse pode ser chamado por um construtores da subclasse através da palavra *super*
- Construtores somente podem ser chamados dentro de construtores da subclasse
- Mesmo assim, se for declarado na primeira linha de código do construtor da subclasse
- Somente construtores da superclasse imediata podem ser chamados usando a palavra-chave *super*

Exemplo Construtor + *super*

```
public class Aluno extends Pessoa {  
    private int matricula;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */
```

```
    public Aluno(String nome, char sexo, int matricula){  
        super(nome, sexo);  
        this.matricula = matricula;  
    }  
}
```



chamada
construtor da
superclasse

Exemplo Construtor *+super* *+this*

```
public class Aluno extends Pessoa {  
    private int matricula;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */  
  
    public Aluno(String nome, char sexo, int matricula){  
        super(nome, sexo);  
        this.matricula = matricula;  
    }  
    public Aluno(){  
        this("", ' ', 0);  
    }  
}
```

← chamada do construtor declarado acima

Métodos + *super*

► Métodos

- Métodos da superclasse podem ser chamados pela palavra-chave *super* seguida de um ponto e do nome do método
- Somente métodos da superclasse imediata podem ser chamados usando a palavra-chave *super* (não existe *super.super*)
- Se um método de uma classe ancestral for herdado pela classe descendente, ele pode ser chamado diretamente pela palavra-chave *super*

Exemplo: métodos + *super*

```
public class Pessoa extends Object {  
    protected String nome;  
    protected char sexo;  
    public Pessoa(String n, char s){  
        nome = n;  
        sexo = s;  
    }  
    public void imprimir(){  
        System.out.println("Nome: " + nome);  
        System.out.println("Sexo: " + sexo);  
    }  
    public int calculaX(){  
        return 999;  
    }  
}
```

Exemplo: métodos + *super*

```
public class Aluno extends Pessoa {  
    private int matricula;  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */  
    ...  
    public void teste(){  
        int x = 0;  
        x = calculaX(); // chamou o método definido  
                        // na superclasse  
        x = super.calculaX(); // igual a linha acima  
    }  
}
```

Palavra reservada *final*

► Palavra reservada *final*

- Atributos definidos como final não podem ter seu valor alterado
- Métodos final não podem sofrer sobreposição
- Métodos final podem sofrer polimorfismo

```
public class Aluno extends Pessoa {  
    private int matricula;  
    private final int numMaximoDisciplinasSemestre = 5  
    /* Atributos herdados  
    public String nome;  
    public char sexo; */  
    ...  
    public final void teste(){  
        ...  
    }  
}
```