
Programação OO

6ª aula

Prof. Douglas Oliveira
douglas.oliveira@prof.infnet.edu.br



- ☐ Classificação ✓
- ☐ Abstração ✓
- ☐ Encapsulamento ✓
- ☐ Relacionamentos ✓
- ☐ Herança
- ☐ Polimorfismo

□ Note as duas classes a seguir:

```
public class Carro {  
    private String placa;  
    private String modelo;  
    private int anoFabricacao;  
    private int kilometragem;  
    private int qtdPassageiros;  
    private int capacidadeBagagem;  
  
    public float calculaConsumo()  
    {...}  
    public float calculaIPVA() {...}  
}
```

```
public class Caminhao {  
    private String placa;  
    private String modelo;  
    private int anoFabricacao;  
    private int kilometragem;  
    private int cargaMaxima;  
    private int numeroEixos;  
  
    public float calculaConsumo()  
    {...}  
    public float calculaIPVA() {...}
```

Definindo dessa forma, teríamos que representar os atributos e métodos comuns nas duas classes (duplicidade de código !)

□ Herança

Mecanismo pelo qual classes herdam atributos e métodos através de um relacionamento hierárquico.

É uma forma de dizer que uma classe "é um tipo de" outra classe.

□ Exemplos:

Um professor é um tipo de funcionário.

Um pentágono é um tipo de polígono.

Um carro é um tipo de veículo.

Um notebook é um tipo de computador.

Na herança, temos sempre uma classe definida de forma genérica que, posteriormente, é refinada em classes mais específicas.

□ Exemplos:

Um **professor** é um tipo de **funcionário**.

Um **pentágono** é um tipo de **polígono**.

Um **carro** é um tipo de **veículo**.

Um **notebook** é um tipo de **computador**.

Classes específicas
ou subclasses

Classes genéricas
ou superclasses

- Repare que podemos ter várias classes específicas para uma mesma classe genérica:

Um professor é um tipo de funcionário

Uma secretária é um tipo de funcionário

Um diretor é um tipo de funcionário

Um pentágono é um tipo de polígono

Um hexágono é um tipo de polígono

Um carro é um tipo de veículo

Um caminhão é um tipo de veículo

- Conforme dito anteriormente, a classe genérica compartilha atributos e métodos com as classes específicas.
- Ou seja, as classes específicas herdam atributos e métodos da classe genérica.
- Além disso, as classes específicas podem ter atributos e métodos somente seus.

□ No exemplo:

Um **carro** é um tipo de **veículo**

Um **caminhão** é um tipo de **veículo**

□ A classe **Veiculo** vai conter os atributos e métodos comuns a todos os veículos.

□ As classes **Carro** e **Caminhão** vão conter os atributos e métodos específicos de carros e caminhões, respectivamente.

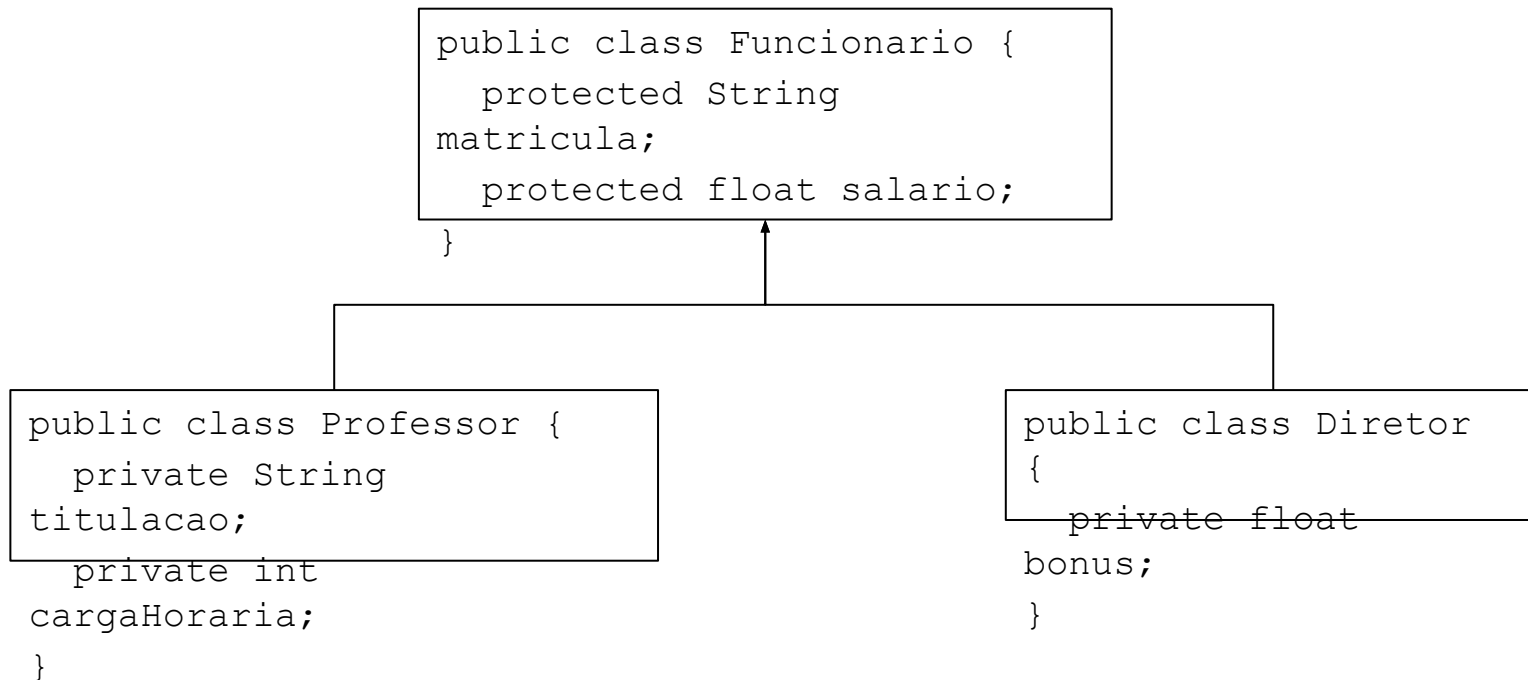
□ Redefinindo usando herança:

```
public class Veiculo {  
    protected String placa;  
    protected String modelo;  
    protected int anoFabricacao;  
    protected int kilometragem;  
    public float calculaConsumo()  
    {...}  
    public float calculaIPVA() {...}  
}
```

```
public class Carro {  
    private int qtdPassageiros;  
    private int  
    capacidadeBagagem;  
}
```

```
public class Caminhao {  
    private int  
    cargaMaxima;  
    private int  
    numeroEixos;
```

□ Outro exemplo:



□ Herança Simples x Herança Múltipla

- ✓ Se uma classe herda de apenas **uma superclasse**, temos uma **herança simples**.
- ✓ Se uma classe herda de **diversas superclasses**, temos uma **herança múltipla**.

Exemplo: **Carro anfíbio** é um tipo de **carro** e **barco**, ao mesmo tempo.

O carro anfíbio possui características de carro e de barco.

- ❑ A linguagem Java **não** possui herança múltipla, **somente** herança simples.
- ❑ Para definir a herança entre duas classes devemos usar a palavra reservada **extends** na definição da subclasse.

```
public class <nome da classe> extends <nome da superclasse>
```

- ❑ Em Java, a subclasse herda **todos** os atributos e métodos definidos como **public** ou **protected** na superclasse.
- ❑ Os atributos e métodos definidos como **private** **NÃO** são herdados pelas subclasses.

□ Exemplo:

```
public class Veiculo {  
    protected String placa;  
    protected String modelo;  
    protected int anoFabricacao;  
    protected int kilometragem;  
    public Veiculo(String placa, String modelo,  
                    int anoFabricacao, int kilometragem){  
        this.placa = placa;  
        this.modelo = modelo;  
        this.anoFabricacao = anoFabricacao;  
        this.kilometragem = kilometragem;  
    }  
}
```

Os atributos **protected**, assim, como os atributos **private**, não são visíveis para quem usa a classe.

Os atributos **protected** são visíveis apenas nas subclasses.

□ Exemplo:

```
public class Carro extends Veiculo {  
    private int qtdPassageiros;  
    private int capacidadeBagagem;  
}
```

Note que a classe **Carro**
possui os atributos:

- placa
- modelo
- anoFabricacao
- kilometragem
- qtdPassageiros
- capacidadeBagagem

□ Exemplo:

```
public class Carro extends Veiculo {  
    private int qtdPassageiros;  
    private int capacidadeBagagem;  
    public Carro(String placa, String modelo,  
                  int anoFabricacao, int kilometragem  
                  int qtdPassageiros, int capacidadeBagagem) {  
        super(placa, modelo, anoFabricacao, kilometragem);  
        this.qtdPassageiros = qtdPassageiros;  
        this.capacidadeBagagem = capacidadeBagagem;  
    }  
}
```

Em seguida, os atributos da classe **Carro** são inicializados.

Usamos a notação **super()** para executar o construtor da superclasse.

□ Exemplo:

```
public class Caminhao extends Veiculo {  
    private long cargaMaxima;  
    private int numeroEixos;  
  
}
```

A classe **Caminhao** possui os atributos:

- placa
- modelo
- anoFabricacao
- kilometragem
- cargaMaxima
- numeroEixos

□ Exemplo:

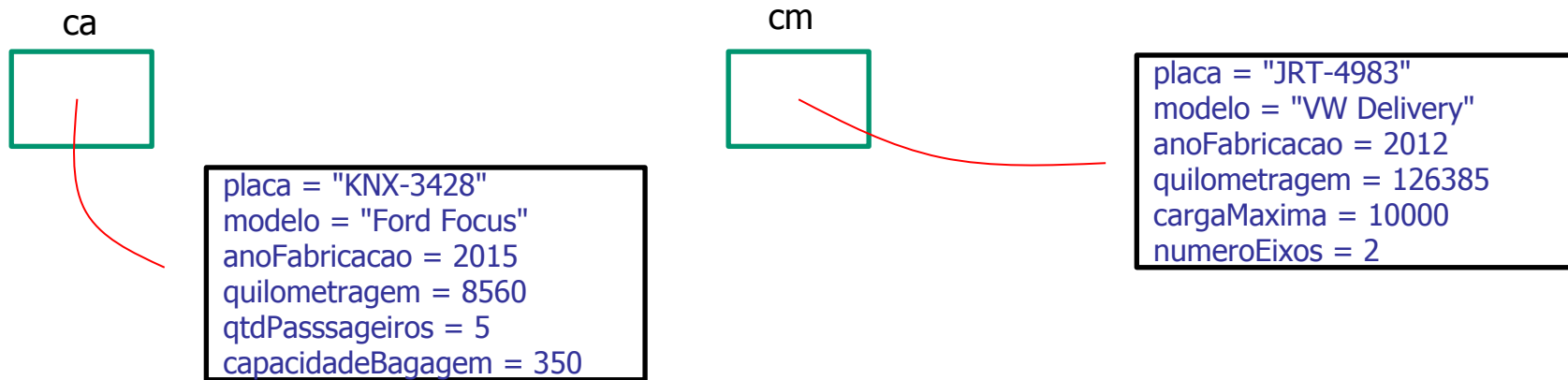
```
public class Caminhao extends Veiculo {  
    private long cargaMaxima;  
    private int numeroEixos;  
    public Caminhao(String placa, String modelo,  
                    int anoFabricacao, int kilometragem  
                    int cargaMaxima, int capacidadeBagagem) {  
        super(placa, modelo, anoFabricacao, kilometragem);  
        this.cargaMaxima = cargaMaxima;  
        this.numeroEixos = numeroEixos;  
    }  
}
```

Em seguida, os atributos da classe **Caminhao** são inicializados.

Usamos a notação **super()** para executar o construtor da superclasse.

□ Exemplo:

```
Carro ca = new Carro("KNX-3428", "Ford Focus",  
                    2015, 8560, 5, 350);  
Caminhao cm = new Caminhao("JRT-4983", "VW Delivery",  
                           2012, 126385, 10000, 2);
```



- Exercício 33: escreva uma hierarquia de classes e seus respectivos atributos e métodos para representar o seguinte cenário.
 1. Um loja vende 3 tipos de produto: livro, CD e software.
 2. Para todos os produtos existe código, descrição, preço e peso.
 3. Para o CD existe o nome da banda.
 4. Para o livro existe o nome do autor
 5. Para o software existe a categoria.
 6. Para entregar um produto, o cálculo do frete é feito multiplicando o peso do produto por R\$ 6,50.

- ❑ Classificação ✓
- ❑ Abstração ✓
- ❑ Encapsulamento ✓
- ❑ Relacionamentos ✓
- ❑ Herança ✓
- ❑ Polimorfismo

□ Polimorfismo

Um mesmo comportamento pode se apresentar de **forma diferente** em **classes diferentes**.

poli = vários

morfismo = forma

- ❑ O polimorfismo trabalha em conjunto com o conceito de herança.
- ❑ Quando definimos uma subclasse, ela herda os métodos da superclasse.
- ❑ Entretanto, a subclasse pode **alterar** um método herdado, ou seja, **redefinir** um método herdado.

- Assim, o polimorfismo é um mecanismo no qual um método da superclasse é **sobrescrito** na subclasse.
- Dessa forma podemos ter o mesmo método implementado de duas ou mais **formas** diferentes.
- O Java vai decidir qual método será chamado no momento da **execução do programa** (ligação tardia ou *late binding*).

Exemplo 1:

```
public class Animal {  
    public void emiteSom() {  
        System.out.println("???"); }  
}  
  
public class Gato extends Animal {  
  
}  
  
public class Cachorro extends Animal {  
    public void emiteSom() {  
        System.out.println("latido"); }  
}
```

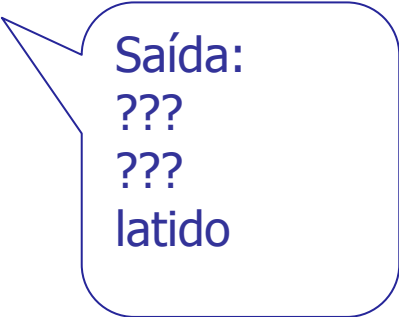
Método original

Método emiteSom() é
sobrescrito.

Note que a assinatura do
método continua a mesma !

❑ Exemplo 1:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal    a1 = new Animal();  
        Gato      a2 = new Gato();  
        Cachorro  a3 = new Cachorro();  
        a1.emiteSom();  
        a2.emiteSom();  
        a3.emiteSom();  
    }  
}
```

A blue speech bubble with a tail pointing towards the code. It contains the text "Saída:" followed by three lines of output: "???", "???", and "latido".

Saída:
???
???
latido

Exemplo 2:

```
public class Animal {  
    public void emiteSom() {  
        System.out.println("???");  
    }  
  
    public class Gato extends Animal {  
        public void emiteSom() {  
            System.out.println("miado");  
        }  
    }  
  
    public class Cachorro extends Animal {  
        public void emiteSom() {  
            System.out.println("latido");  
        }  
    }  
}
```


Método original

Métodos emiteSom() é
sobrescrito nas duas subclasses.

Note que a assinatura do
método continua a mesma !

□ Exemplo 2:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal    a1 = new Animal();  
        Gato       a2 = new Gato();  
        Cachorro  a3 = new Cachorro();  
        a1.emiteSom();  
        a2.emiteSom();  
        a3.emiteSom();  
    }  
}
```

A blue speech bubble pointing to the code. It contains the text "Saída:" followed by "???", "miado", and "latido" on separate lines.

Saída:
???
miado
latido

□ Exemplo 3:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal a1 = new Animal();  
        Animal a2 = new Gato();  
        Animal a3 = new Cachorro();  
        a1.emiteSom();  
        a2.emiteSom();  
        a3.emiteSom();  
    }  
}
```

Quando temos uma herança,
podemos adotar a seguinte regra:

Uma variável do tipo da
superclasse pode receber uma
instância da subclasse.

Exemplo 3:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal a1 = new Animal();  
        Animal a2 = new Gato();  
        Animal a3 = new Cachorro();  
        a1.emiteSom();  
        a2.emiteSom();  
        a3.emiteSom();  
    }  
}
```


Saída:
???
miado
latido

Apesar das variáveis **a2** e **a3** serem do tipo Animal, elas referenciam instâncias de Gato e Cachorro.

Assim, o Java decide qual método **emiteSom()** chamar baseado na instância que está sendo referenciada e não no tipo da variável.

□ Exemplo 4:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal a;  
        a = new Animal();  
        a.emiteSom();  
        a = new Gato();  
        a.emiteSom();  
        a = new Cachorro();  
        a.emiteSom();  
    }  
}
```



Saída:
???
miado
latido

□ Exemplo 5:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal[] v = new Animal[3];  
        v[0] = new Animal();  
        v[1] = new Gato();  
        v[2] = new Cachorro();  
        for (int i = 0; i < v.length; i++)  
            v[i].emiteSom();  
    }  
}
```

Note que o vetor é de **Animal**.
Entretanto, o vetor pode receber
instâncias das subclasses de
Animal (Gato e Cachorro)

□ Exemplo 5:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal[] v = new Animal[3];  
        v[0] = new Animal();  
        v[1] = new Gato();  
        v[2] = new Cachorro();  
        for (int i = 0; i < v.length; i++)  
            v[i].emiteSom();  
    }  
}
```

Note que **v[i]**, ora referencia primeiro uma instância de Animal, depois uma instância de Gato e, por último, uma instância de Cachorro.

□ Exemplo 5:

```
public class Teste {  
    public static void main(String[] args) {  
        Animal[] v = new Animal[3];  
        v[0] = new Animal();  
        v[1] = new Gato();  
        v[2] = new Cachorro();  
        for (int i = 0; i < v.length; i++)  
            v[i].emiteSom();  
    }  
}
```

Saída:
???
miado
latido

No momento da execução, o Java verifica o objeto referenciado por **v[i]** e decide qual método **emiteSom()** deve ser chamado.
Por isso, chamamos de **ligação tardia**

Sobrecarga (Overloading)

```
public class Teste {  
    public void imprime() {  
        System.out.println("*");  
    }  
  
    public void imprime(int n) {  
        for (int i = 1; i <= n;  
i++)  
            System.out.println("*");  
    }  
}
```

Sobrescrita (Overriding)

```
public class Animal {  
    public void emiteSom() {  
        System.out.println("???"); }  
}  
  
public class Gato extends Animal {  
    public void emiteSom() {  
        System.out.println("miado"); }  
}  
  
public class Cachorro extends Animal  
{  
    public void emiteSom() {  
        System.out.println("latido"); }  
}
```

Sobrecarga x Sobrescrita

- Existem diferenças fundamentais entre os conceitos OO de **sobrecarga** e **sobrescrita**:

Característica	Sobrecarga	Sobrescrita
Termo em inglês	Overload	Override
Local	Métodos da mesma classe	Métodos em classes diferentes em uma hierarquia de classes
Assinatura	Métodos com assinaturas diferentes	Métodos com a mesma assinatura
Chamada do método	Em tempo de compilação ou ligação precoce (<i>early binding</i>)	Em tempo de execução ou ligação tardia (<i>late binding</i>)

□ Exercício 34:

1. Todos os funcionários de uma empresa possuem nome e salário bruto.
2. O salário líquido é calculado como salário bruto – imposto de renda.
3. Existem dois tipos especiais de funcionários: estagiários e vendedores.
4. O IR é calculado sobre o salário bruto da seguinte forma:
 - a) Até 900,00 => desconto de ZERO
 - b) Até 1500,00 => desconto de 15%
 - c) Acima de 1500,00 => desconto de 20%
5. Todos os funcionários pagam IR, com exceção dos estagiários que não tem desconto do IR.
6. Além do salário bruto, os vendedores ganham um bônus, que é um percentual a ser acrescido sobre o salário bruto. Nesse caso, o IR é descontado sobre o total recebido pelo vendedor (salário bruto + bônus).

- Exercício 35: escreva uma hierarquia de classes com seus respectivos atributos e métodos para representar o seguinte cenário.
1. Um banco administra contas que possuem um número e um saldo.
 2. Existem 3 tipos de conta: conta comum, conta poupança e conta especial.
 3. É possível depositar e sacar dinheiro das contas. Também é possível consultar o saldo das contas.
 4. Para a conta comum e poupança não é possível sacar mais do que o saldo disponível.
 5. A conta especial possui um limite e é possível sacar além do saldo até o limite da conta.
 6. A conta poupança pode ter o saldo aumentado a partir de um percentual de rendimento.