Herança, reescrita e polimorfismo

Capítulo VI

Herança em Java

```
public class Carro {

    public static int totalCarros = 0;

    private String marca;
    private String cor;
    private int velMax;
    private int numMarchas;

    public Carro() {
        totalCarros++;
    }
```

Existem 4 tipos diferentes de Carros. Como implementar?

Herança em Java

A palavra reservada extends permite que todas os atributos e métodos da classe Carro sejam utilizados

```
public class Fusca extends Carro{

    private String estadoConservacao;
    private boolean radio;
```

```
public class Uno extends Carro{

    private boolean escada;
    private int numPortas;
```

```
public class Gallardo extends Carro{
    private boolean conversivel;
```

```
public class Camaro extends Carro {

    private boolean arCondicionado;
    private boolean tetoSolar;
```

Herança em Java

```java
public class Fusca extends Carro {

    private String estadoConservacao;
    private boolean radio;

    public Fusca() {
        marca = "Volkswagen";
    }
```

```java
public class Uno extends Carro{

    private boolean escada;
    private int numPortas;

    public Uno() {
        marca = "Fiat";
    }
```

```java
public class Camaro extends Carro {

    private boolean arCondicionado;
    private boolean tetoSolar;

    public Camaro() {
        marca = "Chevrolet";
    }
```

```java
public class Gallardo extends Carro{

    private boolean conversivel;

    public Gallardo() {
        marca = "Lamborghini";
    }
```

Se a marca pertence a classe mãe Carro, onde está o erro?

Modificadores de acesso em Java

```
marca has private access in Carro
---
(Alt-Enter mostra dicas)
```

Precisamos fornecer acesso para as classes filhas

Modificador	Classe	Pacote	Subclasses	"Mundo"
public	☑	☑	☑	☑
protected	☑	☑	☑	☒
"nenhum"	☑	☑	☒	☒
private	☑	☒	☒	☒

Modificadores de acesso em Java

```java
public class Carro {

    public static int totalCarros = 0;

    protected String marca;
    protected String cor;
    protected int velMax;
    protected int numMarchas;

    public Carro() {
        totalCarros++;
    }
```

```java
public class Camaro extends Carro {

    private boolean arCondicionado;
    private boolean tetoSolar;

    public Camaro() {
        marca = "Chevrolet";
    }
```

```java
public class Fusca extends Carro {

    private String estadoConservacao;
    private boolean radio;

    public Fusca() {
        marca = "Volkswagen";
    }
```

```java
public class Uno extends Carro{

    private boolean escada;
    private int numPortas;

    public Uno() {
        marca = "Fiat";
    }
```

```java
public class Gallardo extends Carro{

    private boolean conversivel;

    public Gallardo() {
        marca = "Lamborghini";
    }
```

Trabalhando com métodos em herança

```
public class Carro {

    public static int totalCarros = 0;

    protected String marca;
    protected String cor;
    protected int velMax;
    protected int numMarchas;
    protected float preco;
    protected int ano;

    public Carro() {
        totalCarros++;
    }

    public float calculaPreco() {
        preco = consultaTabelaFipe(ano);
        return preco;
    }
```

Mas cada tipo de Carro possui uma forma de calcular o seu preço, onde isso entraria?

Reescrita de métodos

```java
public class Gallardo extends Carro{

    private boolean conversivel;

    public Gallardo() {
        marca = "Lamborghini";
    }

    @Override
    public float calculaPreco() {
        float novoPreco = super.calculaPreco();
        if (conversivel) {
            novoPreco += 1500.0f;
        }
        return novoPreco;
    }
```

```java
public class Gallardo extends Carro{

    private boolean conversivel;

    public Gallardo() {
        marca = "Lamborghini";
    }

    @Override
    public float calculaPreco() {
        float novoPreco = super.calculaPreco();
        if (conversivel) {
            novoPreco += 1500.0f;
        }
```

method does not override or implement a method from a supertype
---
(Alt-Enter mostra dicas)

```java
    @Override
    public float calculaPreco(float precoAntigo){
        float novoPreco = super.calculaPreco();
        if (conversivel) {
            novoPreco += 1500.0f;
        }
        return novoPreco;
    }
```

Além de facilitar o entendimento do código, essa anotação indica ao compilador e pode nos ajudar a encontrar erros.

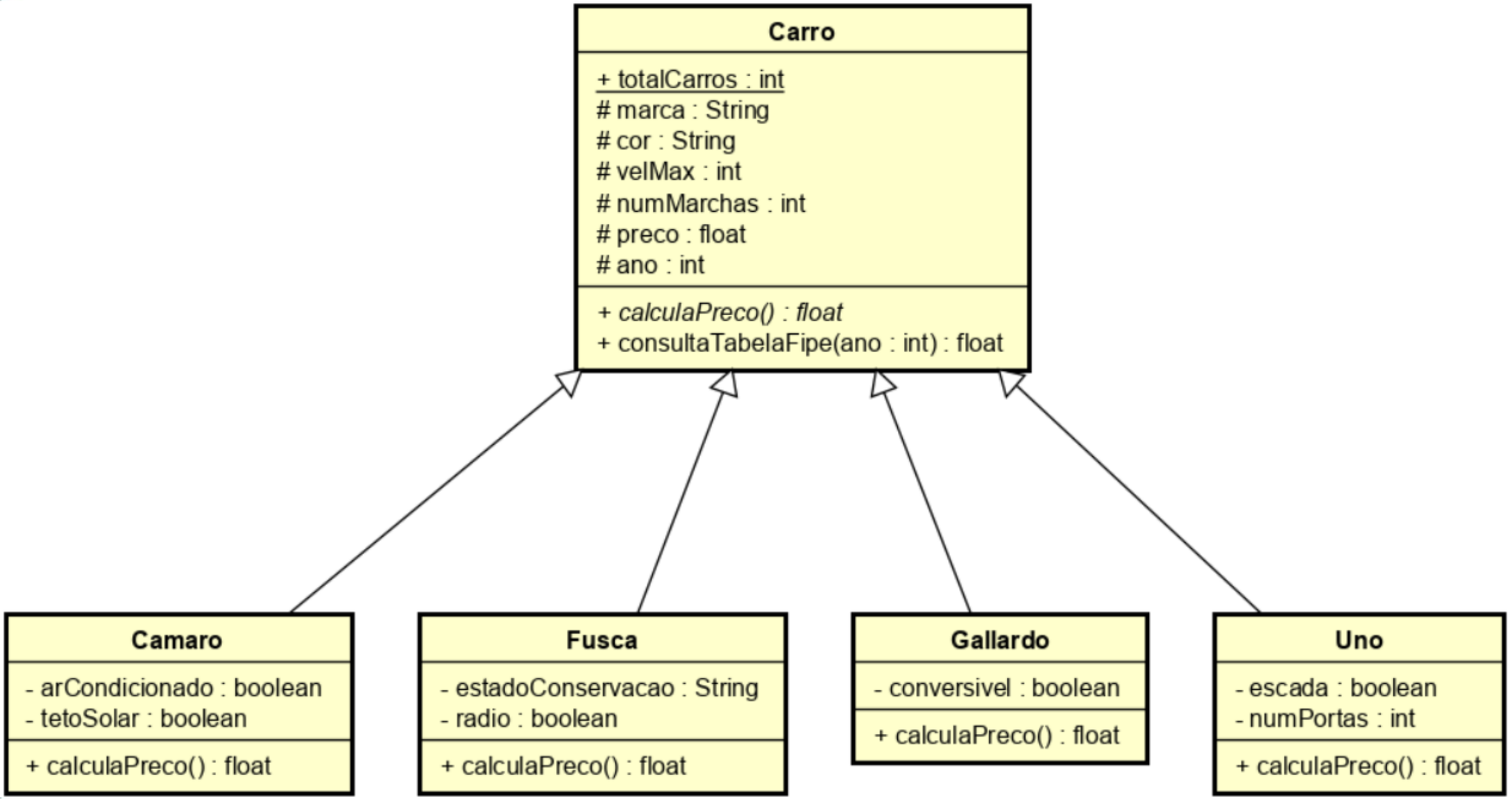
Polimorfismo para armazenar os veículos

```
public class Concessionaria {
    public Carro[] lerCarros() {

        Carro[] carros = new Carro[3];
        Gallardo gallardo = new Gallardo();
        Uno uno = new Uno();
        Camaro camaro = new Camaro();

        gallardo.setCor("Laranja");
        gallardo.setNumMarchas(8);
        gallardo.setVelMax(350);
        gallardo.setConversivel(true);

        uno.setCor("Vermelho");
        uno.setNumMarchas(6);
        uno.setNumPortas(4);
        uno.setVelMax(300);
        uno.setEscada(true);

        camaro.setCor("Amarelo");
        camaro.setNumMarchas(8);
        camaro.setTetoSolar(false);
        camaro.setVelMax(200);
        camaro.setArCondicionado(true);

        carros[0] = gallardo;
        carros[1] = uno;
        carros[2] = camaro;

        return carros;
    }
}
```

Polimorfismo para cálculo e validação de NFs

```
public class Main {

    public static void main(String[] args) {
        Carro[] car;
        Concessionaria lojaVeiculos = new Concessionaria();

        car = lojaVeiculos.lerCarros();

        for (Carro carro : car) {

            if (carro instanceof Camaro) {
                System.out.println("Temos um Camaro!");
            }
            if (carro instanceof Fusca) {
                System.out.println("Temos um Fusca!");
            }
            if (carro instanceof Gallardo) {
                System.out.println("Temos um Gallardo!");
            }
            if (carro instanceof Uno) {
                System.out.println("Temos um Uno!");
            }

        }

    }
}
```

O atributo **instanceof** significa "*é um*" ou "*é uma instância de*"


Carro
+ totalCarros : int
# marca : String
# cor : String
# velMax : int
# numMarchas : int
# preco : float
# ano : int
+ calculaPreco() : float
+ consultaTabelaFipe(ano : int) : float
Camaro
- arCondicionado : boolean
- tetoSolar : boolean
+ calculaPreco() : float
Fusca
- estadoConservacao : String
- radio : boolean
+ calculaPreco() : float
Gallardo
- conversivel : boolean
+ calculaPreco() : float
Uno
- escada : boolean
- numPortas : int
+ calculaPreco() : float

Classes final

```
public final class CartaoCredito {
    private int agencia;
    private int conta;
}
```

```
cannot inherit from final CartaoCredito
----
(Alt-Enter mostra dicas)
public class NovoCartaoCredito extends CartaoCredito {

}
```

Métodos final

```
public final float consultaTabelaFipe(int ano) {
    return Fipe.Verificacao(ano);
}
```

```
consultaTabelaFipe(int) in Gallardo cannot override consultaTabelaFipe(int) in Carro
 overridden method is final
----
(Alt-Enter mostra dicas)
    public float consultaTabelaFipe(int ano){
        return -1;
    }
```

Exercícios

1. Cria um sistema para controle de contas bancárias, sabendo que toda conta tem um número de identificação e uma agência. Existem vários tipos de conta, porém neste momento estamos interessados somente nos tipos abaixo:
   i. Pessoa física: possui informações do proprietário (nome e CPF) além do saldo
   ii. Pessoa jurídica: possui as informações da empresa (nome e CNPJ), além do saldo e limite de financiamento

2. Implemente um programa para o controle de inventário de equipamentos da sua empresa. Neste primeiro momento serão levantados e smartphones:
   i. Notebooks: Marca, modelo, matrícula do responsável e número de série do aparelho
   ii. Smartphone: Marca, modelo, IMEI, Centro de Custo e matrícula do responsável

3. Implemente o sistema de controle de acesso das informações de Recursos Humanos da sua empresa sabendo que existem, basicamente 3 tipos de funcionários:
   i. Comum: representa todos os funcionários e possui Nome, CPF, Matrícula e salário
   ii. Gestor: além de ser um funcionário comum ele possui uma gratificação anual. Pode possuir vários funcionários em seu time e com isso visualizar somente as informações deles
   iii. Recursos humanos: pode visualizar as informações de todos os funcionários da empresa

Obrigado!