iscte
INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

Módulo 1: Programação em Linguagem Java

Aula 7
Herança e Polimorfismo

UPskill

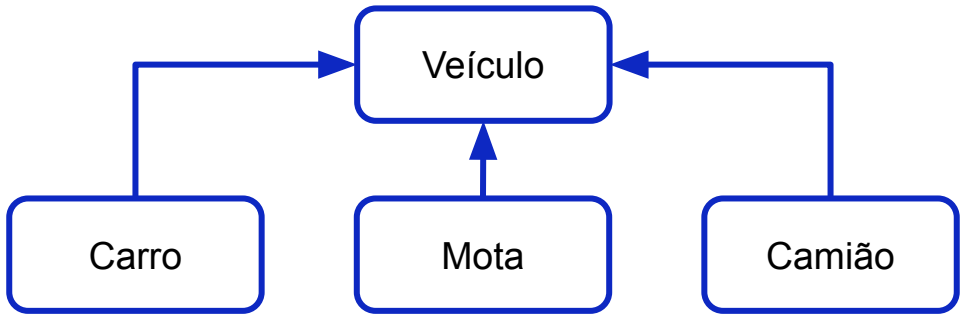
Herança

- Refere-se à capacidade no Java de uma classe herdar propriedades e funções de outra classe. Esta herança é concretizada quando uma classe **estende** outra.
    - Uma classe pode **estender** outra e herdar as suas propriedades.

- Quando uma classe herda de outra classe no Java, as duas ficam categorizadas da seguinte forma:
    - A classe que **estende** (herda da outra classe) é a ***subclasse***
    - A classe que **é estendida** (cujas propriedades são herdadas) é a ***superclasse***

Herança

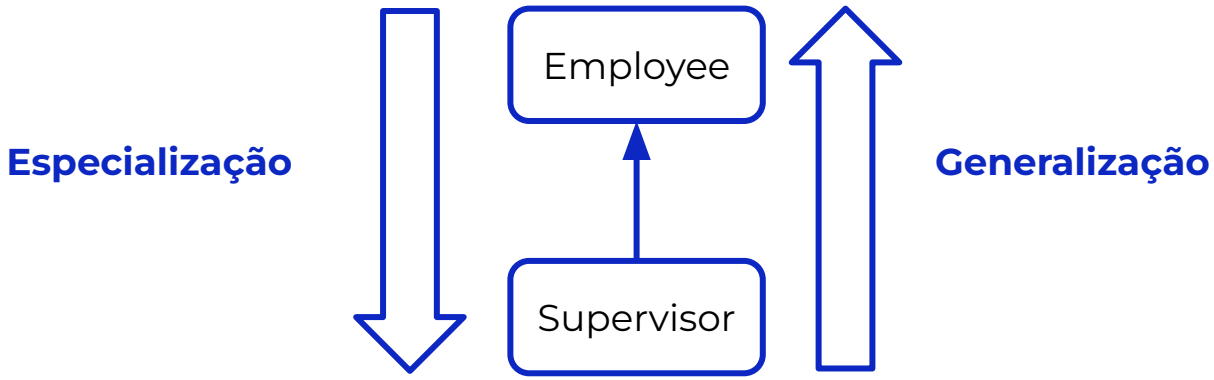
- A herança pode ser um método eficaz de partilhar código entre classes que têm características partilhadas, mas permitindo que algumas partes sejam diferentes.



Herança - Exemplo “Employee”

```
public class Employee {
   private String name;
   private String ssn;

   public Employee(String name, String ssn) {
       this.name = name;
       this.ssn = ssn;
   }

   public String getName() {
       return name;
   }

   public String getSsn() {
       return ssn;
   }

   @Override
   public String toString() {
       return "(" + getName() + ", " + getSsn() + ")";
   }

}
```

Relação de Generalização



Um Supervisor **é sempre** um Employee

Um Employee **pode ser** um Supervisor

Herança

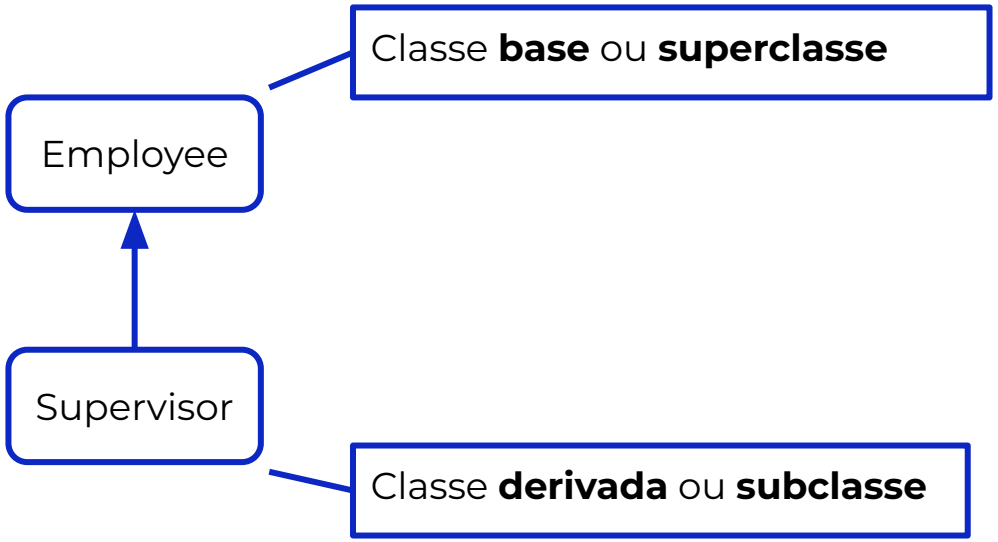
```
public class Supervisor extends Employee {
    private int level;

    public Supervisor(String name, String ssn, int level) {
        //...
    }

    public int getLevel() {
        return level;
    }

    @Override
    public String toString() {
        return "(" + getName() + ", " + getSsn() + ", " + getLevel() + ")";
    }

}
```

Um **Supervisor** é um **Employee**

Novo método especifico ao **Supervisor**

Sobreposição do método com a mesma assinatura da classe base

Relação



Herança

- Classe derivada **deriva** da classe base (subclasse deriva da superclasse)
- Métodos são herdados e mantêm categoria de acesso (visibilidade)
- Relação **“é um”** - Referências do tipo de classe base podem referir-se a objetos de classes derivadas

Exemplo:

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
Employee employee = new Supervisor("Felisberto", "987654321", 5);
```

Herança

- Classe derivada **tem todas as propriedades da base**

Exemplo:

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
Employee employee = new Supervisor("Felisberto", "987654321", 5);
String employee_ssn_id_1 = employee.getSsn();
String employee_ssn_id_2 = supervisor.getSsn();
```

Sobreposição

- Método de classe derivada pode **sobrepor-se** a método da classe base
- Sobreposição é uma forma de **especialização**
- Regras
  - Mesma assinatura e tipo de devolução compatível
  - Método na classe base não privado e não final
  - Método na classe derivada com acessibilidade igual ou superior
- **Definida pela keyword `@Override`**

Categorias de Acesso (Visibilidade)

- Atributos ou métodos podem ser:
  - **private** - acesso apenas por outros métodos da mesma **classe**
  - **package-private** - adicionalmente, acesso por métodos do mesmo **pacote**
  - **protected** - adicionalmente, acesso por métodos de **classes derivadas**
  - **public** - acesso universal

Interfaces de uma classe

- Dentro da própria classe temos acesso a:
  - Métodos e atributos da própria classe
  - Métodos e atributos não privados da super classe
- Nas classes do mesmo *package* temos acesso a:
  - Métodos e atributos não privados da classe ou das sua super classe
- Numa subclasse:
  - Métodos e atributos protegidos ou públicos da própria classe ou da sua superclasse

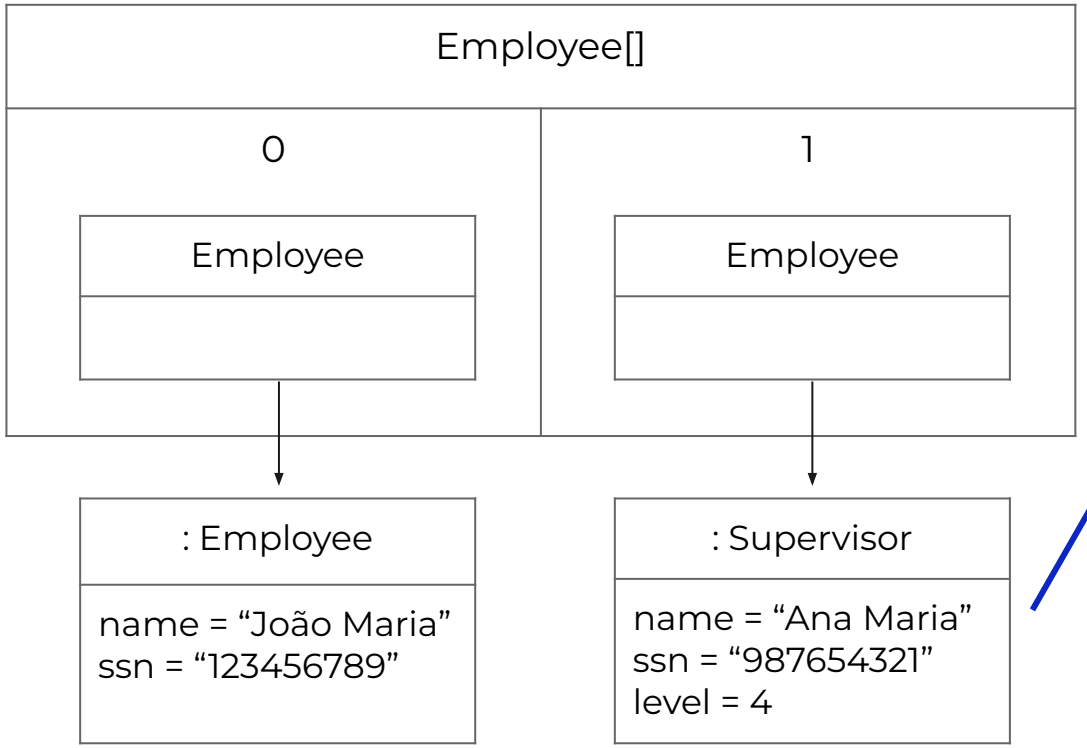
E o método **toString()?**

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
Employee employee = new Supervisor("Felisberto", "987654321", 5);
System.out.println(supervisor);
System.out.println(employee);
```

Qual é o método ***.toString()*** executado?

O resultado depende do tipo do objeto e não do tipo da referência!

E para guardar num **Vector?**



Possível porque a classe **Supervisor** deriva da classe **Employee.** Ou seja, é possível porque um Supervisor **é sempre também** um Employee

Polimorfismo

- Designa-se pela capacidade de um objeto tomar várias formas
  - A forma descrita pela classe **a que pertence**
  - As formas descritas pelas classes **acima** na hierarquia a que pertence
- Objeto pode ser referenciado por referências do tipo da classe a que pertence ou das classes acima na hierarquia (mais genéricas)

O que aparece na consola?

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
Employee anEmployee = new Supervisor("Felisberto", "987654321", 5);
Employee anotherEmployee = new Employee("Elvira", "111111111");
System.out.println(supervisor.toString());
System.out.println(anEmployee.toString());
System.out.println(anotherEmployee.toString());
```

O que aparece na consola?

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
Employee anEmployee = new Supervisor("Felisberto", "987654321", 5);
Employee anotherEmployee = new Employee("Elvira", "111111111");
System.out.println(supervisor.toString());
System.out.println(anEmployee.toString());
System.out.println(anotherEmployee.toString());
```

Resultado na consola:

```
(Guilhermina, 123456789, 3)
(Felisberto, 987654321, 5)
(Elvira, 111111111)
```

A classe “ **Object**”

```
public class Employee extends Object {
     private String name;
     private String ssn;

     public Employee(final String name, final String ssn) {
          ...
     }
```

Se uma classe **não derivar explicitamente de outra, derivará implicitamente da classe Object**, que está no topo da hierarquia de classes do Java.

Métodos *Final*

- Classe base pode proibir às classes derivadas a sobreposição de um seu método, que se dirá ser um método **final**
- Razão para um método ser final:
    - Programador que forneceu o método na classe base entendeu que classes derivadas não deveriam poder especializar o modo de funcionamento desse método

Exercício 1

Crie um programa que use um vetor de empregados para calcular o total dos salários a pagar de uma cadeia de lojas. O vetor de empregados deve conter empregados (sem especialização), gerentes de loja e diretores regionais. Para cada classe de empregados, o salário é calculado da seguinte maneira:

- **Empregados:** valor fixo de 800€;
- **Gerente de loja:** valor fixo igual ao dos empregados sem especialização, acrescido de um prémio de 200€ que é atribuído sempre que a loja cumpre os objectivos das vendas.
- **Diretor regional:** valor fixo igual ao dobro do dos empregados sem especialização, acrescido de um prémio que corresponde a 1% do lucro mensal nas lojas da região.

Nota: para simplificar, considere que ter cumprido ou não os objectivos de vendas é um atributo dos gerentes e que o lucro mensal da região é um atributo dos diretores.

Exercício 1 - **Resolução**

Empregado

```
public class Empregado {

    private String nome;
    private String ssn;
    private double salario = 800;

    public Empregado(String nome, String ssn){
        this.nome = nome;
        this.ssn = ssn;
    }

    public String getNome() {
        return nome;
    }

    public double getSalario() {
        return salario;
    }

    public String getSsn() {
        return ssn;
    }
}
```

GerenteDeLoja

```
public class GerenteDeLoja extends Empregado {

    private boolean objetivosCumpridos;
    private double premio = 200;

    public GerenteDeLoja(String nome, String ssn, boolean objetivosCumpridos){
        super(nome,ssn);
        this.objetivosCumpridos = objetivosCumpridos;
    }

    @Override
    public double getSalario() {
        if(objetivosCumpridos)
            return super.getSalario()+premio;
        return super.getSalario();
    }
}
```

Exercício 1 - **Resolução**

Diretor Regional

```
public class DiretorRegional extends Empregado {

    private double lucroMensal;

    public DiretorRegional(String nome, String ssn, double lucroMensal){
        super(nome,ssn);
        this.lucroMensal = lucroMensal;
    }

    @Override
    public double getSalario() {
        return super.getSalario()*2 + lucroMensal*0.01;
    }

}
```

Main

```
public class Main {

    public static void main(String[] args) {

        Empregado[] emp = new Empregado[5];
        emp[0] = new Empregado("Pedro","7456");
        emp[1] = new GerenteDeLoja("Maria","6732", false);
        emp[2] = new GerenteDeLoja("Joana","8342", true);
        emp[3] = new DiretorRegional("João","7654",100000);
        emp[4] = new DiretorRegional("Filipe","6721",120000);

        for(Empregado e : emp){
            System.out.println(e.getSalario());
        }

    }
}
```

Exercício 2

Crie uma classe chamada `Ingresso` que possui um valor em Euros e um método `imprimeValor()`.

- Crie uma classe `VIP`, que herda de Ingresso e possui um valor adicional. Crie um método que retorne o valor do ingresso VIP (com o adicional incluído).
- Crie uma classe `CamaroteInferior` (que possui a localização do ingresso e métodos para obter e imprimir esta localização) e uma classe `CamaroteSuperior`, que é mais cara (possui valor adicional). Esta última possui um método para retornar o valor do ingresso. Ambos os camarotes devem herdar as características dos Ingressos `VIP`.

Exercício 2 - **Resolução**

Ingresso

```
public class Ingresso {

    protected double preco;

    public Ingresso(double preco){
        this.preco = preco;
    }

    public void imprimeValor(){
        System.out.println(preco);
    }

}
```

IngressoVIP

```
public class IngressoVIP extends Ingresso {

    protected double valorAdicional;

    public IngressoVIP(double preco, double valorAdicional) {
        super(preco);
        this.valorAdicional = valorAdicional;
    }

    @Override
    public void imprimeValor() {
        System.out.println(preco+valorAdicional);
    }
}
```

Exercício 2 - **Resolução**

CamaroteInferior

```
public class CamaroteInferior extends IngressoIP {

    protected String localizacao;

    public CamaroteInferior(double preco, double
valorAdicional, String localizacao) {
        super(preco, valorAdicional);
        this.localizacao = localizacao;
    }

    public String getLocalizacao() {
        return localizacao;
    }

    public void imprimeLocalizacao(){
        System.out.println(localizacao);
    }
}
```

CamaroteSuperior

```
public class CamaroteSuperior extends CamaroteInferior {

    protected double valorAdicionalCamarote;

    public CamaroteSuperior(double preco, double valorAdicional, String
localizacao, double valorAdicionalCamarote) {
        super(preco, valorAdicional, localizacao);
        this.valorAdicionalCamarote = valorAdicionalCamarote;
    }

    @Override
    public void imprimeValor() {
        System.out.println(preco+valorAdicional+valorAdicionalCamarote);
    }
}
```

Exercício 3

Pretende-se escrever um programa para registar as estatísticas de jogadores de futebol:

- A classe (base) `Jogador` deve conter o nome e o número do jogador. Deve também ter um método para registar cada golo marcado e um inspetor para o número de golos marcados.
- A classe `GuardaRedes`, uma extensão de `Jogador`, deve permitir registar e consultar o número de golos sofridos.
- A classe `JogadorDeCampo`, uma extensão de `Jogador`, deve permitir registar e consultar o número de passes feitos e recebidos.

Teste criando um jogador de campo e um guarda-redes e use as funções próprias de cada classe para atribuir a cada jogador (desde que seja possível): 2 golos marcados, 3 golos sofridos, 4 passes feitos e 5 passes recebidos.
Sobreponha o método `toString()` em ambas as classes para melhor visualizar os resultados do teste, que devem ser impressos para o ecrã.

Exercício 3 - **Resolução**

Jogador

```
public class Jogador {

    protected String nome;
    protected String numero;
    protected int golosMarcados;

    public Jogador(String nome, String numero, int golosMarcados){
        this.nome = nome;
        this.numero = numero;
        this.golosMarcados = golosMarcados;
    }

    public int getGolosMarcados() {
        return golosMarcados;
    }

    public void marcaGolo(){
        golosMarcados++;
    }

    @Override
    public String toString() {
        return "Jogador{" +
                "nome='" + nome + '\'' +
                ", numero='" + numero + '\'' +
                ", golosMarcados=" + golosMarcados +
                '}';
    }

}
```

GuardaRedes

```
public class GuardaRedes extends Jogador {

    private int golosSofridos;


    public GuardaRedes(String nome, String numero, int golosMarcados, int golosSofridos) {
        super(nome, numero, golosMarcados);
        this.golosSofridos = golosSofridos;
    }

    public int getGolosSofridos() {
        return golosSofridos;
    }

    public void sofreGolo(){
        golosSofridos++;
    }

    @Override
    public String toString() {
        return "Jogador{" +
                "nome='" + nome + '\'' +
                ", numero='" + numero + '\'' +
                ", golosMarcados=" + golosMarcados +
                ", golosSofridos=" + golosSofridos +
                '}';
    }
}
```

Exercício 3 - **Resolução**

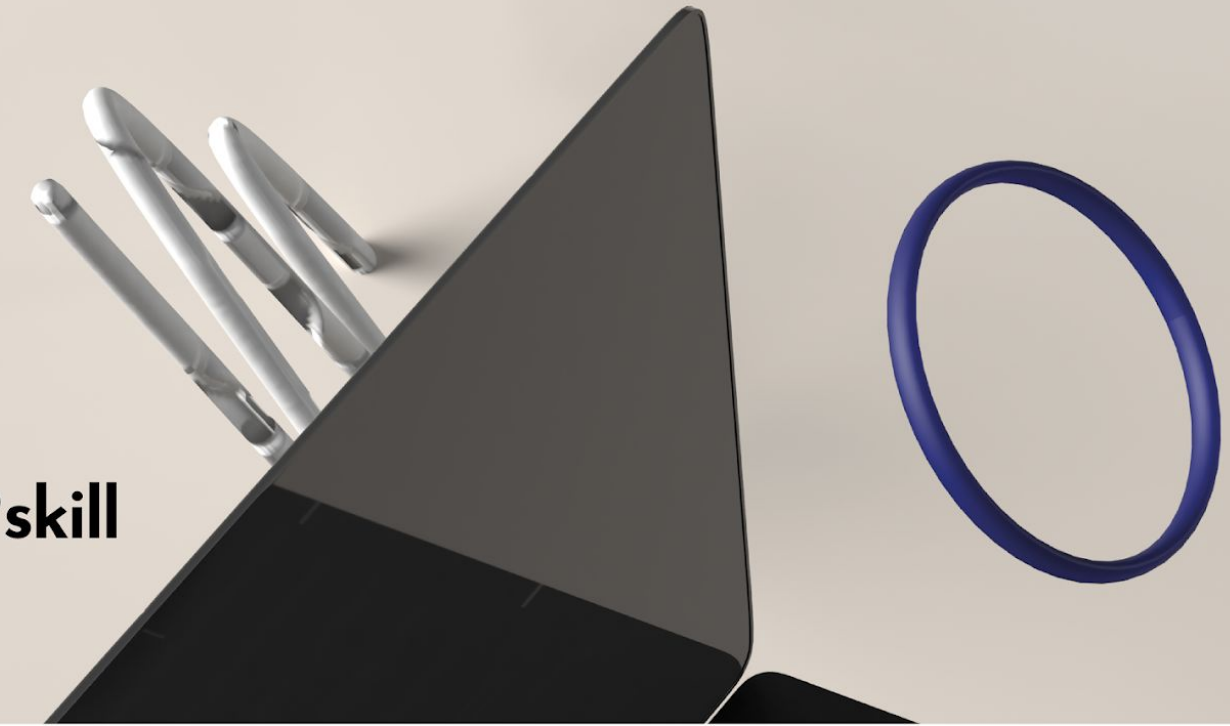
JogadorDeCampo

```
public class JogadorDeCampo extends Jogador {

    private int passesFeitos;
    private int passesRecebidos;


    public JogadorDeCampo(String nome, String numero, int golosMarcados, int
passesFeitos, int passesRecebidos) {
        super(nome, numero, golosMarcados);
        this.passesFeitos = passesFeitos;
        this.passesRecebidos = passesRecebidos;
    }

    @Override
    public String toString() {
        return "Jogador{" +
                "nome='" + nome + '\'' +
                ", numero='" + numero + '\'' +
                ", golosMarcados=" + golosMarcados +
                ", passesFeitos=" + passesFeitos +
                ", passesRecebidos=" + passesRecebidos +
                '}';
    }

}
```

Main

```
public class Main {

    public static void main(String[] args) {

        JogadorDeCampo j1 = new JogadorDeCampo("Pedro","2",0,4,5);
        GuardaRedes j2 = new GuardaRedes("Pedro","3",0,0);

        j1.marcaGolo();
        j1.marcaGolo();

        j2.sofreGolo();
        j2.sofreGolo();
        j2.sofreGolo();

        System.out.println(j1);
        System.out.println(j2);

    }

}
```

O futuro profissional começa aqui



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

UPskill