iscte
INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

Módulo 1: Programação em Linguagem Java

# Aula 8
# Classe Abstrata e Exercícios

UPskill

# O que é uma classe abstrata?

- As classes abstratas servem como “modelo” para outras classes que dela herdem, **não podendo ser instanciadas por si só**.

- Uma classe abstrata pode ter métodos abstratos e não abstratos. Os métodos abstratos são métodos **sem implementação**, que terão de ser implementados nas subclasses.

- Resumindo: classes abstratas não podem ser instanciadas, elas servem apenas para que outras classes a usem como modelo (a extendam e herdem os atributos/propriedades e métodos delas).

Classes abstratas **não podem ser instanciadas.**
Servem apenas para **que outras classes a usem como modelo** (a extendam e herdem os atributos / propriedades e métodos).

# O que é um método abstrato?

```
public abstract class ContaBancaria {

    ...

    public abstract double getSaldo();

}
```

**Apenas é definida a assinatura.** A implementação só é feita pelas subclasses

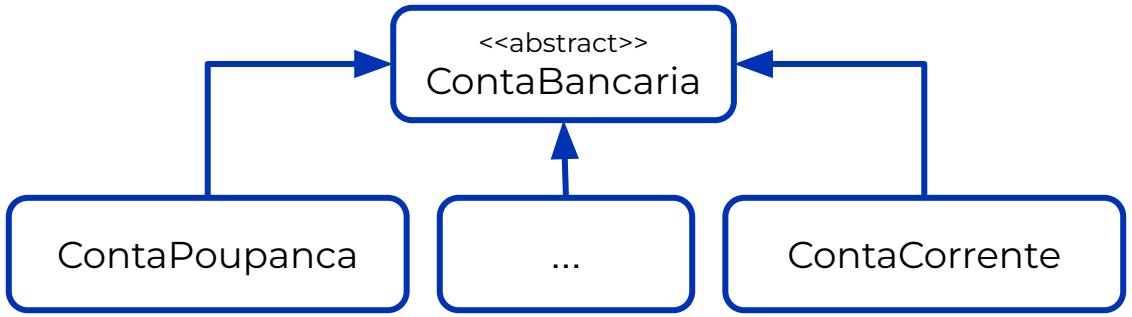
**Por exemplo, podemos usar uma classe abstrata para representar uma conta (*ContaBancaria*), com um método abstrato *getSaldo()* que deve ser implementado pelas suas subclasses.**

# Exemplo

```
public abstract class ContaBancaria {

   //...

   public abstract double getSaldo();
}
```

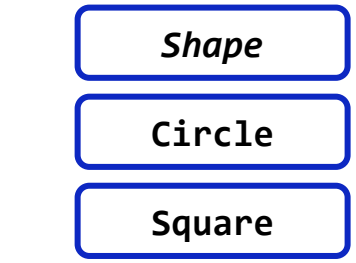
```
public class ContaPoupanca extends ContaBancaria {

   //...

   @Override
   public double getSaldo(){
       return saldo * juro;
   }
}
```

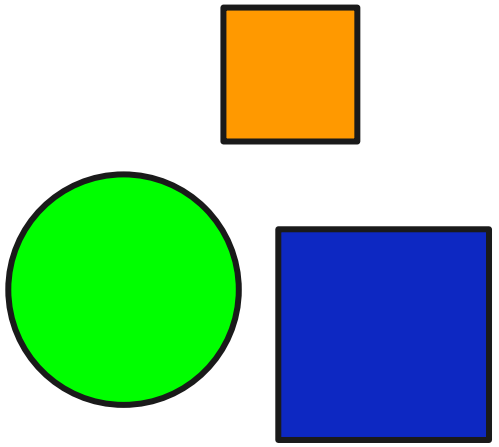
# Exemplo



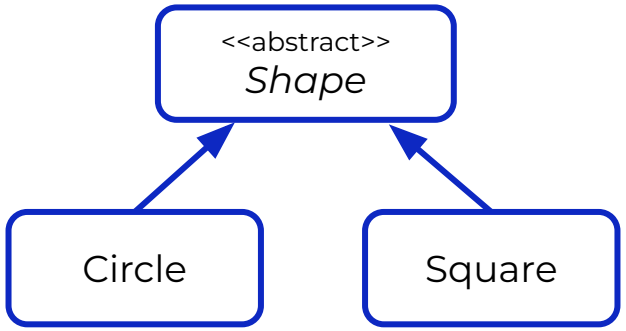
# Outro exemplo: **Conceito**

- Forma (Abstrata)
- Círculo
- Quadrado



# Outro exemplo: **Relações**




Um Círculo **é uma** Forma

Um Quadrado **é uma** Forma

# Outro exemplo: **Implementação Shape**

```
public abstract class Shape {
   private Position position;

   public Shape(Position position) {
       this.position = position;
   }

   public Position getPosition() {
       return position;
   }

   public abstract double getArea();
   public abstract double getPerimeter();

}
```

Operações abstractas, ou seja, operações **sem qualquer implementação** disponível até este nível da hierarquia.

# Outro exemplo: **Implementação Circle**

```
public class Circle extends Shape {
    private double radius;

    public Circle(Position position, double radius) {
        super(position);
        this.radius = radius;
    }

    public final double getRadius() {
        return radius;
    }

    //...

}
```

Um Circle **é uma** Shape e herda os atributos, propriedades e métodos da classe Shape.

É necessário apenas um atributo adicional, correspondente a uma das duas propriedades de um círculo (o raio), **já que a posição do centro é herdada da classe *Shape***.

# Implementação - Circle

```
    //...

    @Override
    public double getArea() {
        return Math.PI * getRadius() * getRadius();
    }

    @Override
    public double getPerimeter() {
        return 2.0 * Math.PI * getRadius();
    }
}
```

Qual é a área de um círculo?
Fácil: $\pi \times r^2$

Esta classe **fornece implementações**, para cada uma das operações abstratas (métodos) da classe *Shape*

# Atenção!

```
Shape a = new Shape();

Shape b = new Circle();

Circle c = new Circle();
```

**Não é possível instanciar uma classe abstrata!**

**Circle está a ser guardado com o tipo Shape -> polimorfismo.** Só é possível aceder aos métodos definidos na classe mãe. Vantagem: isto inclui os abstratos.

# Exercício 1

Crie uma hierarquia de classes para representar os diferentes tipos de funcionários de um escritório que tem os seguintes cargos: **gerente**, **assistente** e **vendedor**. Escreva uma classe base abstrata chamada Funcionário que declara um método abstrato:

```
double calculaSalario()
```

Esta classe também deve definir os seguintes atributos: nome (tipo String), código de funcionário (tipo String) e salario base (tipo double). Crie um construtor que recebe os valores a serem armazenados nos respectivos atributos. Esta classe abstrata deverá ser estendida pelas outras classes representativas dos tipos de funcionários. Em cada classe deve-se redefinir o método `calculaSalario()` de forma que o cálculo do salário seja feito da seguinte forma: o gerente recebe duas vezes o `salario_base`, o assistente recebe o `salário_base` e o vendedor recebe o `salario_base` mais uma comissão definida no momento de inicialização.

Teste a sua implementação criando um objeto de cada tipo e imprima para o ecrã os salários de cada um.

# Exercício 1 - **Resolução**

## Vendedor

```
class Vendedor extends Funcionario {
   private double comissao;

   public Vendedor(String nome, String codigo, double comissao) {
       super(nome, codigo);
       this.comissao = comissao;
   }

   @Override
   public double calculaSalario() {
       return getSalarioBase()+comissao;
   }
}
```

## Main

```
public class Main {

   public static void main(String[] args) {
       Assistente a = new Assistente("Andre", "a1");
       Gerente g = new Gerente("Maria", "g1");
       Vendedor v = new Vendedor("Luisa", "v1", 100);
       System.out.println(a.calculaSalario());
       System.out.println(g.calculaSalario());
       System.out.println(v.calculaSalario());

   }
}
```

# Exercício 2

Crie uma classe `Calculadora`. Esta classe deve ser abstrata e implementar as operações básicas (soma, subtração, divisão e multiplicação).

Utilizando o conceito de herança, crie uma classe chamada `CalculadoraCientífica` que implementa os seguintes cálculos: raiz quadrada e a potência.

Dica: utilize a classe Math do pacote java.lang.

# Exercício 2 - **Resolução**

## Calculadora

```
abstract class Calculadora {


   public Calculadora() {
   }


   public double soma(double a, double b){
       return  a+b;
   }
   public double sub(double a, double b){
       return  a-b;
   }
   public double div(double a, double b){
       return  a/b;
   }
   public double mul(double a, double b){
       return  a*b;
   }
}
```

## Calculadora Científica

```
class CalculadoraCientifica extends Calculadora {


   public CalculadoraCientifica() {
   }


   public double raizQuad(double a){
       return Math.sqrt(a);
   }


   public double potencia(double a, double exp){
       return Math.pow(a, exp);
   }
}
```

# Exercício 3

Escreva uma classe abstrata chamada `CartaoPresente`. Essa classe representa todos os tipos de cartões presente e conterá apenas um atributo: `destinatario` (tipo `String`).

Nessa classe, deverá também ser declarado o método *`public abstract void showMessage()`*.

Crie classes filhas da classe *`CartaoPresente`*: `DiaDosNamorados`, `Natal` e `Aniversário`. Cada uma dessas classes deve conter um método construtor que receba o nome do destinatário do cartão. Cada classe também deverá implementar o método *`showMessage()`*, mostrando uma mensagem ao utilizador com seu nome e que seja específica para a data de comemorativa do cartão.

Teste a sua implementação criando instâncias dos 3 tipos de cartões. Depois, exiba as mensagens desses cartões chamando o método *`showMessage()`*.

# Exercício 3 - **Resolução**

## Cartão Presente

```
public abstract class CartaoPresente {
	private String destinatario;

	public CartaoPresente(String destinatario){
	    this.destinatario = destinatario;
	}

	public abstract void showMessage();

	public String getDestinatario() {
	    return destinatario;
	}
}
```

## Dia Dos Namorados

```
public class DiaDosNamorados extends CartaoPresente {

   public DiaDosNamorados(String destinatario){

       super(destinatario);
   }

   public void showMessage(){

       System.out.println("Feliz Dia dos Namorados " +
getDestinatario());
   }
}
```

# Exercício 3 - **Resolução**

## Aniversário

```
public class Aniversario extends CartaoPresente {


   public Aniversario(String destinatario){
       super(destinatario);
   }


   public void showMessage(){
       System.out.println("Feliz Aniversario " +
getDestinatario());
   }
}
```

## Natal

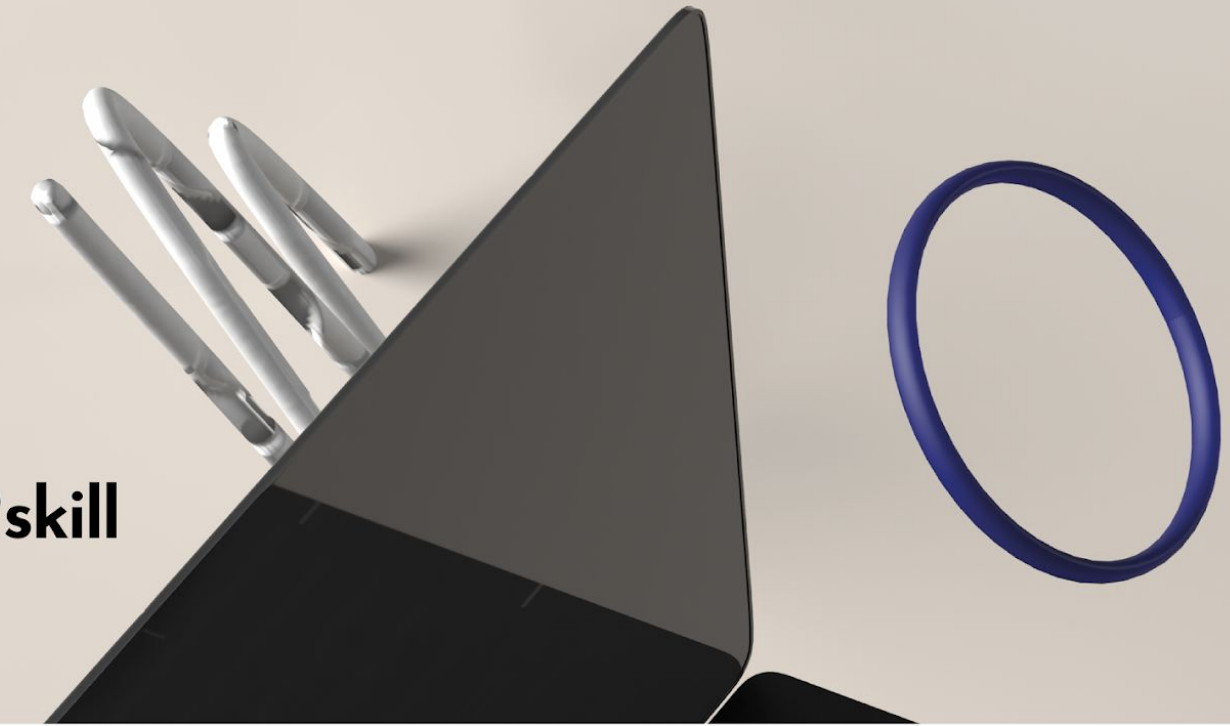
```
public class Natal extends CartaoPresente {


   public Natal(String destinatario){
       super(destinatario);
   }


   public void showMessage(){
       System.out.println("Feliz Natal " +
getDestinatario());
   }
}
```

# Exercício 3 - **Resolução**

## Main

```
public class Main {


   public static void main(String[] args) {


       CartaoPresente aniversario = new Aniversario("André");
       CartaoPresente diadosnamorados = new DiaDosNamorados("Rute");
       CartaoPresente natal = new Natal("Rui");
       Natal natal1 = new Natal("Maria");
       aniversario.showMessage();
       diadosnamorados.showMessage();
       natal.showMessage();
       natal1.showMessage();
   }
}
```

# O futuro profissional começa aqui



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

UPskill