# Programação I

## Classes Abstratas e Interfaces

Samuel da Silva Feitosa

Aula 10

# Classes Abstratas (1)

- No banco, todas as contas são de um tipo específico.
  - Por exemplo, conta poupança, conta corrente ou conta salário.
  - Essas contas poderiam ser modeladas utilizando o conceito de herança:

```java
public class Conta {
    // Atributos
    // Construtores
    // Métodos
}
```

```java
public class ContaCorrente extends Conta {
    // Atributos
    // Construtores
    // Métodos
}
```

```java
public class ContaPoupanca extends Conta {
    // Atributos
    // Construtores
    // Métodos
}
```

# Classes Abstratas (2)

- Para cada conta do domínio do banco devemos criar um objeto da classe correspondente ao tipo da conta.
  - Por exemplo, se existe uma conta poupança no domínio do banco devemos criar um objeto da classe ContaPoupanca.

```
ContaPoupanca cp = new ContaPoupanca();
```

- Faz sentido criar objetos da classe ContaPoupanca pois existem contas poupança no domínio do banco.
  - Dizemos que a classe ContaPoupanca é uma **classe concreta** pois criaremos objetos a partir dela.

# Classes Abstratas (3)

- Por outro lado, a classe Conta não define uma conta que de fato existe no domínio do banco.
  - Ela apenas serve como “base” para definir as contas concretos.
  - Não faz sentido criar um objeto da classe Conta pois estaríamos instanciado um objeto que não é suficiente para representar uma conta que pertença ao domínio do banco.
  - Mas, a princípio, não há nada proibindo a criação de objetos dessa classe.
  - Para adicionar essa restrição no sistema, devemos tornar a classe Conta abstrata.

# Classes Abstratas (4)

- Uma classe concreta pode ser diretamente utilizada para instanciar objetos.
- Por outro lado, uma **classe abstrata** não pode.
  - Para definir uma classe abstrata, basta adicionar o modificador **abstract**.

```
public abstract class Conta {
    // Atributos
    // Construtores
    // Métodos
}
```

  - Todo código que tenta criar um objeto de uma classe abstrata não compila.

# Métodos Abstratos (1)

- Suponha que o banco ofereça extrato detalhado das contas e para cada tipo de conta as informações e o formato desse extrato detalhado são diferentes.
  - Neste caso, parece não fazer sentido ter um método na classe Conta para gerar extratos detalhados pois ele seria reescrito nas classes específicas sem nem ser reaproveitado.
  - Poderíamos, simplesmente, não definir nenhum método para gerar extratos detalhados na classe Conta.
  - Porém, não haveria nenhuma garantia que as classes que derivam direta ou indiretamente da classe Conta implementem métodos para gerar extratos detalhados.

# Métodos Abstratos (2)

- Para garantir que toda classe concreta que deriva direta ou indiretamente da classe Conta tenha uma implementação de método para gerar extratos detalhados e além disso que uma mesma assinatura de método seja utilizada, devemos utilizar o conceito de **métodos abstratos**.
  - Na classe Conta, definimos um método abstrato para gerar extratos detalhados.
  - Um método abstrato não possui corpo (implementação).

```
public abstract class Conta {
    // Atributos
    // Construtores
    // Métodos

    public abstract String emiteExtratoDetalhado();
}
```

# Métodos Abstratos (3)

- As classes concretas que derivam direta ou indiretamente da classe Conta devem possuir uma implementação para o método emiteExtratoDetalhado().

- Se a classe concreta não possuir uma implementação deste método, ela não irá compilar.

```
public class ContaPoupanca extends Conta {
    // Atributos
    // Construtores
    // Métodos

    public String emiteExtratoDetalhado() {
        String extrato = "EXTRATO DETALHADO DE POUPANÇA";
        // Outras informações
        extrato += "Saldo: " + this.getSaldo();

        return extrato;
    }
}
```

# Interfaces (1)

- É o mecanismo pelo qual o programador estabelece um conjunto de operações sem se preocupar com a sua implementação.
  - Definição do modelo semântico para outras classes.
- Em sua forma mais simples pode conter:
  - Métodos sem implementação (assinaturas).
  - Constantes.
- Métodos em interfaces públicos e abstratos.
- Já as constantes são implicitamente públicas, estáticas e finais.

# Interfaces (2)

- Num sistema orientado a objetos, os objetos interagem entre si através de chamadas de métodos (troca de mensagens).
  - Podemos dizer que os objetos se “encaixam” através dos métodos públicos assim como um plugue se encaixa em uma tomada através dos pinos.
  - Para os objetos de uma aplicação “conversarem” entre si mais facilmente é importante padronizar o conjunto de métodos oferecidos por eles.
- Um padrão é definido através de especificações ou contratos.
  - Podemos criar um “contrato” para definir um determinado conjunto de métodos que deve ser implementado pelas classes que “assinarem” este contrato.
  - Em orientação a objetos, um contrato é chamado de **interface**.

# Interfaces (3)

- No sistema do banco, podemos definir uma interface (contrato) para padronizar as assinaturas dos métodos oferecidos pelos objetos que representam as contas do banco.

```
interface Conta {
    public void deposita(double valor);
    public boolean saca(double valor);
}
```

- Os métodos de uma interface não possuem corpo (implementação) pois serão implementados nas classes vinculadas a essa interface.
  - Todos os métodos de uma interface devem ser públicos e abstratos.
  - Os modificadores public e abstract são opcionais.

# Realização de Interfaces

- As classes que definem os diversos tipos de contas que existem no banco devem **implementar** (assinar) a interface Conta.

- As classes concretas que implementam uma interface são obrigadas a possuir uma implementação para cada método declarado na interface. Caso contrário, ocorrerá um erro de compilação.
- Útil para garantir a padronização e para forçar que classes implementem certos métodos.

```
public class ContaPoupanca implements Conta {
    // Atributos
    // Construtores
    // Métodos
    public void deposita(double valor) {
        // Implementação
    }
    public boolean saca(double valor) {
        // Implementação
        return true;
    }
}
```

# Interfaces - Novas Funcionalidades

- A partir da versão 8 do Java, novos conceitos podem ser utilizados com relação a interfaces.
- Métodos **default**
  - Implementação de métodos completos em interfaces (assinatura + corpo do método).
- Métodos estáticos
  - Funcionalidade similar à já existente em classes.
- Interfaces Funcionais
  - Uma interface que implementa apenas um método é considerada *Funcional*.
  - Podem ser representadas por *expressões lambda*.

# Considerações Finais

- Nesta aula trabalhamos com diversos novos conceitos de linguagens orientadas a objetos.
- Vimos o uso de herança de forma mais aprofundada.
- Trabalhamos com classes abstratas.
  - Vimos o conceito de classes concretas/abstratas.
- Iniciamos nossos estudos com Interfaces.
  - Definimos interfaces e vimos sua relação com as classes que as implementam.
  - Vimos as novas funcionalidades disponíveis a partir do Java versão 8.