

# POO - PROGRAMAÇÃO ORIENTADA A OBJETOS

## Especialização, Classes Abstratas e Polimorfismo

Rodrigo R Silva

Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense  
Campus Bagé

# Nesta Aula Veremos...

1 Introdução

2 Hierarquia de Classes

3 Polimorfismo

4 Classes Abstrata

# Introdução

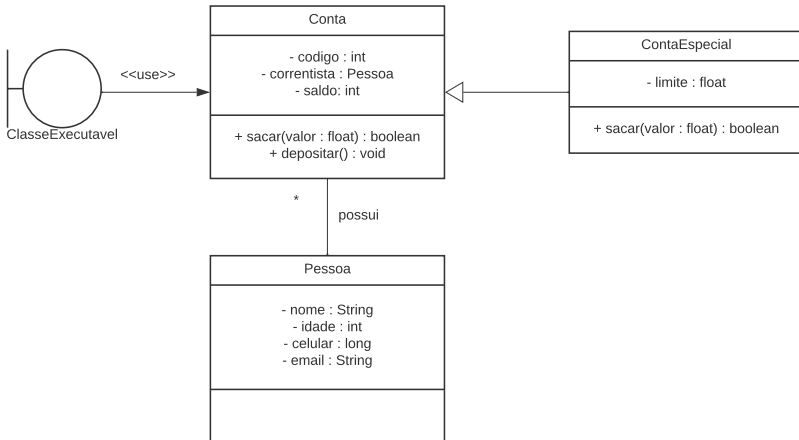
# Contextualizando...

- A aplicação bancária é constituída de contas que são movimentadas pelos clientes.
- Entretanto, podemos ter vários tipos e contas.
- Ex: conta corrente, conta poupança, conta especial, etc.
- Notem que todas são tipos de conta, ou seja, compartilham características em comum.
- Toda conta possui um número, correntista e saldo.

## Hierarquia de Classes

# A hierarquia de classes

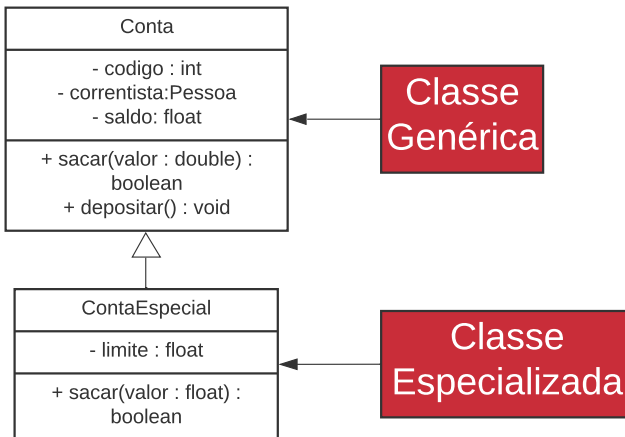
- Observem que uma classe pode ser descendente de outra.



# O conceito de especialização

- Especialização é um mecanismo da tecnologia de orientação a objetos que permite que uma classe estenda as características e funcionalidades de outra.
- Neste sentido, a **classe especializada** herda os atributos e comportamentos definidos na classe genérica.
- **ContaEspecial** herda os atributos e comportamentos definidos na classe **Conta**.

# Representação gráfica





# Aplicando especialização em Java

- Em Java, para criarmos uma classe especializada utilizamos a palavra reservada **extends**.
- Vejamos como criar a classe **ContaEspecial** a qual é uma especialização de **Conta**.

```
public class ContaEspecial extends Conta {  
  
}
```

# O construtor de uma especialização

- O construtor de classes especializadas possui uma característica que o distingue dos construtores de classes convencionais:
- além de inicializarem os atributos definidos na sua própria classe, também devem inicializar os atributos herdados da classe genérica;
- entretanto, a responsabilidade de inicialização dos atributos da classe genérica é do construtor definido na própria classe genérica.
- Ex: a classe **Conta** possui seus próprios construtores, assim como **ContaEspecial**.

# Os construtores

```
public class Conta {  
    private int codigo;  
    private Pessoa correntista;  
    private float saldo;
```

```
    public Conta() {  
    }  
}
```


```
    public Conta(int codigo, Pessoa correntista, float saldo) {  
        this.codigo = codigo;  
        this.correntista = correntista;  
        this.saldo = saldo;  
    }  
}
```

```
public class ContaEspecial extends Conta {
```

```
    private float limite;  
    public ContaEspecial() {  
    }  
}
```

```
    public ContaEspecial(int numero, Pessoa correntista, float saldo, float limite) {  
        this.limite = limite;  
    }  
}
```

O construtor de **Conta** inicializa somente os atributos definidos na própria classe **Conta**



O construtor de **ContaEspecial** inicializa somente os atributos definidos na própria classe **ContaEspecial**



## super()

- Super() é uma operação utilizada para referenciar um construtor definido na superclasse (generalização) de uma classe especializada.
- Com uso de super() um construtor pode repassar os demais atributos para o construtor da classe genérica da hierarquia de classes.
- O construtor da classe **ContaEspecial** pode usar super() para repassar os valores dos atributos herdados para o construtor da classe **Conta**.

# O uso do super()

```
public class Conta {  
    private int codigo;  
    private Pessoa correntista;  
    private float saldo;  
  
    public Conta() {  
    }  
  
    public Conta(int codigo, Pessoa correntista, float saldo) {  
        this.codigo = codigo;  
        this.correntista = correntista;  
        this.saldo = saldo;  
    }  
}  
  
public class ContaEspecial extends Conta {  
    private float limite;  
  
    public ContaEspecial() {  
        super();  
    }  
  
    public ContaEspecial(int numero, Pessoa correntista, float saldo, float limite) {  
        super(numero, correntista, saldo);  
        this.limite = limite;  
    }  
}
```

super() chama o construtor padrão(default) da classe **Conta**

## Polimorfismo

# Polimorfismo

- Notem que quando um método é herdado em uma classe especializada ele pode ter um comportamento distinto do método original definido na classe genérica.
- Este é o conceito de polimorfismo: “**modificação do comportamento do método herdado na classe especializada**”.
- Ex: método **sacar()** na classe **ContaEspecial**.
- Possui um comportamento diferente do original definido na classe Conta, visto que uma ContaEspecial pode possuir saldo negativo. De acordo com o limite de crédito da conta.

## Exemplo de polimorfismo

```
public boolean sacar(float valor) {  
    if(this.saldo - valor >= 0) {  
        this.saldo = this.saldo - valor;  
        return true;  
    }  
    return false;  
}
```

Método sacar original definido na classe **Conta**

```
@Override  
public boolean sacar(float valor) {  
    if(this.getSaldo() - valor >= this.limite) {  
        this.setSaldo(this.getSaldo() - valor);  
        return true;  
    }  
    return false;  
}
```

Método sacar redefinido na classe **ContaEspecial**

**OBS:** notem que a assinatura do método é igual em ambas as classes. Entretanto, o código (comportamento) é distinto.



## Classes Abstrata

## Retomando...

- Observamos que o modelo de classes de um sistema pode formar uma hierarquia.
- Geralmente, uma classe base da hierarquia serve para modelo de especializações.
- Notamos isso no modelo de classes do banco:
- Ex: conta corrente, conta poupança, conta especial, etc.
- Dessa forma, teremos variados tipos de contas na aplicação bancária.

# Definição de classe abstrata

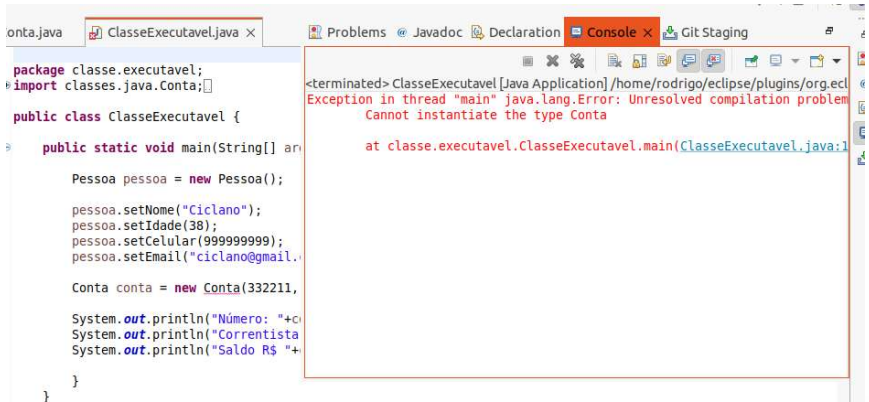
- Em Java, utilizamos a palavra reservada **abstract** para definição de uma classe abstrata.
- Quando uma classe é definida como abstrata, não é permitida a instanciação de objetos.
- Exemplo: tornaremos a classe Conta abstrata no modelo de classes da aplicação bancária.
- Isso quer dizer que não poderemos criar instâncias de conta diretamente.
- Somente poderemos criar instâncias das especializações de conta.

# Exemplo

- Vejamos como tornar a classe Conta abstrata.

```
package classes.java;  
  
public abstract class Conta {  
  
    private int codigo;  
    private Pessoa correntista;  
    private float saldo;  
  
    public Conta() {  
  
    }  
}
```

# Erro ao tentar instanciar



```
Conta.java  ClasseExecutavel.java x  Problems  @ Javadoc  Declaration  Console x  Git Staging

package classe.executavel;
import Classes.java.Conta;

public class ClasseExecutavel {

    public static void main(String[] args) {

        Pessoa pessoa = new Pessoa();

        pessoa.setNome("Ciclano");
        pessoa.setIdade(38);
        pessoa.setCelular(999999999);
        pessoa.setEmail("ciclano@gmail.com");

        Conta conta = new Conta(332211, "Ciclano", "Ciclano");

        System.out.println("Número: " + conta.getNumero());
        System.out.println("Correntista: " + conta.getCorrentista());
        System.out.println("Saldo R$ " + conta.getSaldo());

    }
}
```

<terminated> ClasseExecutavel [Java Application] /home/rodrigo/eclipse/plugins/org.eclipse.jdt.launcher/...  
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
Cannot instantiate the type Conta  
at classe.executavel.ClasseExecutavel.main(ClasseExecutavel.java:1)

# Os métodos de uma classe abstrata

- Classes abstratas podem conter dois tipos de métodos:
- **Métodos concretos:** aqueles que possuem assinatura e comportamento pré-definido.
- **Métodos abstratos:** possuem somente a assinatura.
- O comportamento dos métodos abstratos deve ser implementado pelas classes especializadas.

# Métodos abstratos

- Vejamos um exemplo de métodos abstratos.
- Tornaremos o método sacar abstrato.
- Dessa forma, o comportamento do método sacar deve ser definidos nas especializações de conta:
- São elas: conta corrente, conta poupança, etc.

```
public abstract boolean sacar(float valor);
```

# OBRIGADO!