- **UESC - Universidade Estadual de Santa Cruz**

Cap 3 – Herança e Polimorfismo

Parte 3 - Polimorfismo

Disciplina: Linguagem de Programação III
Professor: Otacílio José Pereira

Plano de Aula

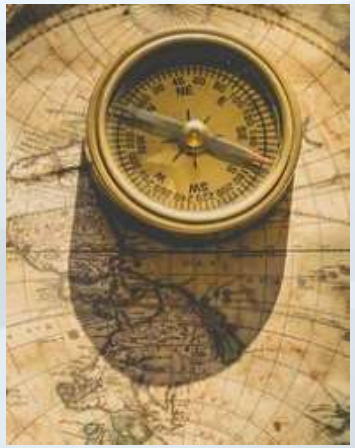
- ***Objetivos***
  - Compreender o polimorfismo com alguns exemplos
  - Reforçar alguns conceitos de OO com seus princípios
- ***Tópicos***
  - Contexto: Tópicos já explorados
  - Polimorfismo: Introdução
  - Exemplos e situações com polimorfismo
  - Tipos de polimorfismo
  - Polimorfismo de Sobrecarga
    - Detalhando características
    - Exemplos
  - Polimorfismo de Sobreposição
    - Detalhando características
    - Exemplos
  - Princípios da Orientação a Objetos

Contexto

- Onde estamos?
- Foco agora!
- Cenário de exemplo

Onde estamos?

- *Considerando nosso planejamento inicial*

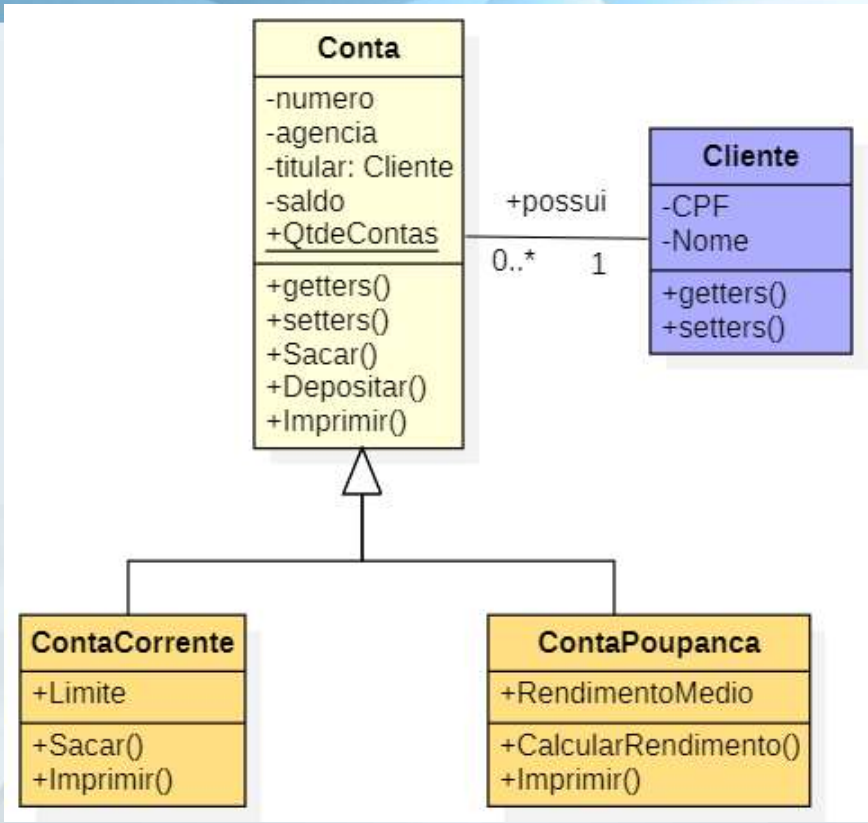


- Capítulo 1 – Introdução
- Capítulo 2 - Conceitos básicos de Orientação a Objetos
- Capítulo 3 – Herança
  - Parte 1 – Herança
  - **Parte 2 - Polimorfismo**

Foco

- Capítulo 4 – Classes abstratas e interfaces
  - Parte 1 – Classes Abstratas
  - Parte 2 - Interfaces
- Capítulo 5 – Generics, collections e outros tópicos
- Capítulo 6 – Desenvolvimento de um projeto em OO

Foco agora



- Compreender polimorfismo
  - Relembrar cenário de Banco
  - Conceito e tipos
  - Sobrecarga de métodos
  - Sobreposição

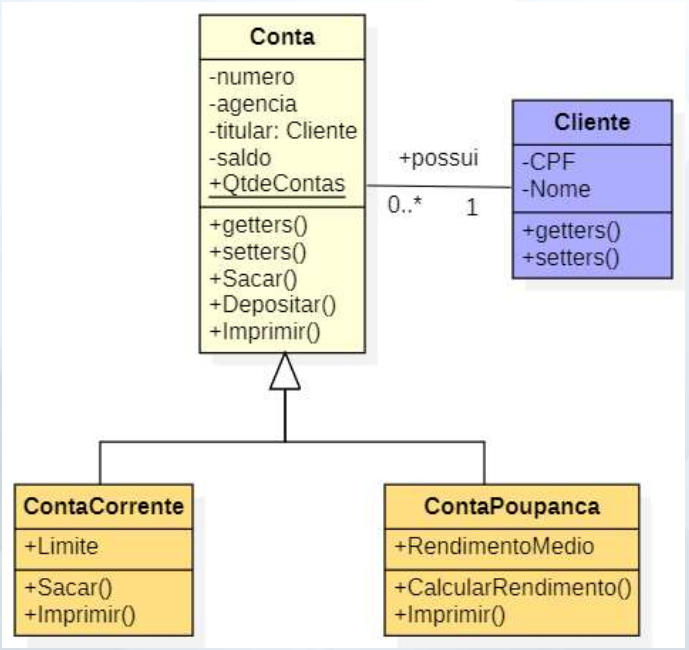
- Após primeiros exemplos com herança, o foco agora explorar outro princípio da OO que é o polimorfismo

Relembrando Cenário

- Modelo
- Codificação

Cenário do Banco

- Este cenário é bastante didático para compreender orientação a objetos
- Além disso, como é bastante popular, o aluno pode recorrer a outras fontes para esclarecer dúvidas ou aprofundar os assuntos
- Nas aulas usamos um cenário de Figuras Geométricas, para diversificar e ser mais espontâneo



Cenário do Banco

- Relembrando classes, atributos e métodos

```java
public class Conta {

    protected String  titular;
    protected int     agencia;
    protected int     numero;
    protected double  valor = 100.0;

    Conta()
    {
        System.out.println("Conta1");
        titular          = "Novo titular";
        agencia          = 1;
        numero           = 1;
        valor            = 200.0;
    }
```

```java
public void Sacar(Double pValor)
{
    if (pValor < this.valor)
        this.valor = this.valor - pValor;
}
```

```java
public class ContaCorrente extends Conta{

    private double  limite;

    ContaCorrente()
    {
        super();
        System.out.println(" ContaCorrente1 ");
        limite = 0;
    }
```

```java
public class ContaCorrente extends Conta{

    private double  limite;

    ContaCorrente()
    {
        super();
        System.out.println(" Cc
        limite = 300.0;
    }
```

```java
public class ContaPoupanca extends Conta{

    private double SaldoMinimo;

    ContaPoupanca()
    {
        this.SaldoMinimo = 50.0;
    }

    @Override
    public void Sacar(Double pValor)
    {
        if (pValor < ( this.valor - this.SaldoMinimo ))
            this.valor = this.valor - pValor;
    }
}
```

- ContaCorrente: possui limite
- ContaPoupança: deve prever um saldo mínimo para ser mantida

Cenário de Figuras Geométricas

- Relembrando os códigos
  - Figura2D, Retangulo e Círculo

```
public class Figura2D {

    protected double p1x, p1y;

    Figura2D()
    {
        this.p1x = 0.0;
        this.p1y = 0.0;
    }

    Figura2D(double pp1x, double pp1y)
    {
        this.p1x = pp1x;
        this.p1y = pp1y;
    }

    public double CalcularArea()
    {
        return 0;
    }
}
```

```
public class Retangulo extends Figura2D{

    private double p2x, p2y;

    Retangulo()
    {   super();
        this.p2x = 0.0;
        this.p2y = 0.0;     }

    Retangulo(double pp1x, double pp1y,
              double pp2x, double pp2y)
    {   super(pp1x, pp1y);

        this.p2x = pp2x;
        this.p2y = pp2y;}

    public double CalcularArea()
    {   double area;
        area = (this.p2x - this.p1x) *
               (this.p2y - this.p1y);

        return area;}
}
```

Polimorfismo

- Conceito
- Tipos
- Exemplo

Polimorfismo

- De maneira informal, no polimorfismo um determinado comportamento desejado (por exemplo via execução de um método), pode assumir diversas maneiras de ser executado a depender das circunstâncias em que é chamado

- Por exemplo, classes derivadas de uma mesma classe possuem métodos que se comportam de forma diferente
  - Classes ContaCorrente e ContaPoupança derivadas da superclasse Conta podem ter o método Sacar que para objetos de cada classe se comportam de forma diferente

Exemplo Conceitual

- Em muitos livros, existem os exemplos conceituais, que podem ilustrar o que é ter um mesmo comportamento desejado mas que se manifesta de forma diferente em classes mais específicas
  - Superclasse: Animal
    - Podem se Mover, Comer e se Reproduzir
  - Classes derivadas
    - Cachorro move-se, come e se reproduz de uma certa forma
    - Um passarinho já realiza estas operações de forma diferente
    - Uma cobra já move-se come e se reproduz conforme sua natureza
- Perceba, aqui é apenas um exemplo conceitual (sem código) para entender a ideia de polimorfismo, um mesmo comportamento de várias formas.

Tipos de Polimorfismo

- Polimorfismo é um conceito mais amplo e existem outros tipos
- O mais comum de vermos está associado ao polimorfismo de subclasses
- Vale a pena discutirmos os dois tipos de polimorfismo
  - Estático, relacionado com as assinaturas dos métodos
  - Dinâmica, relacionado com as subclasses e seus métodos especializados

Polimorfismo estático ou de Sobrecarga

- Assinatura de método
- Sobrecarga do método
- Discussão

Polimorfismo

- Observe o código a seguir referente as formas de Sacar em uma Conta

```
public void Sacar(Double pValor)
{
    if (pValor < this.valor)
        this.valor = this.valor - pValor;
}

public void Sacar(Double pValor, String pMoeda)
{
    if (pMoeda.equals("Dolar"))
        if ((pValor * 4.15) < this.valor)
            this.valor = this.valor - pValor;
    else
        if (pValor < this.valor)
            this.valor = this.valor - pValor;
}
```

- No primeiro caso o Sacar recebe apenas o parâmetro Valor
- No segundo caso há o parâmetro Valor e a Moeda

- Como estes métodos seriam chamados? Como seriam diferenciados?

Assinatura de método

- Assinatura de método
  - A assinatura de um método é a forma como se pode identificar unicamente um método em uma classe
  - Ele é composto pelo nome e pelo conjunto de parâmetos
  - Por isso que no código a seguir, consegue-se distinguir quais formas de sacar devem ser executadas

```
Conta c1 = new Conta();
...
c1.Sacar(200.0);
...
c1.Sacar(300.0, “Dolar”);
```

Polimorfismo dinâmico ou de Sobreposição

- Observe
- Codificação (Relembrando)

Polimorfismo de Sobreposição

- Neste caso o polimorfismo ocorre por meio das subclasses
- Observe este caso, ambas classes Círculo e Retângulo herdam de Figura2D
- Como figuras elas podem ter a área calculada

```
public class Circulo extends Figura2D{
    private double raio;

    Circulo()
    {   super();
        this.raio = 1.0;}

    Circulo(double pp1x, double pp1y, double praio)
    {   super(pp1x, pp1y);
        this.raio = praio;    }

    public double CalcularArea()
    {
        double area;
        area = Math.PI * Math.pow(this.raio, 2);
        return area;
    }
}
```

```
public class Retangulo extends Figura2D{

    private double p2x, p2y;

    Retangulo()
    {   super();
        this.p2x = 0.0;
        this.p2y = 0.0;    }

    Retangulo(double pp1x, double pp1y,
              double pp2x, double pp2y)
    {   super(pp1x, pp1y);

        this.p2x = pp2x;
        this.p2y = pp2y;}

    public double CalcularArea()
    {   double area;
        area = (this.p2x - this.p1x) *
               (this.p2y - this.p1y);

        return area;}
}
```

Exemplo típico

- Percebam que com a implementação anterior é possível implementar o código a seguir:

- Desenho é um vetor de que?
- Qual figura está na posição 0 e 1 do vetor?
- Ao percorrer o vetor como se sabe qual método para cálculo de área?

```
public class Principal {
    public static void main(String[] args)
    {
        Figura2D[] desenho = new Figura2D[3];
        double area;

        desenho[0] = new Retangulo();
        desenho[1] = new Circulo();
        desenho[2] = new Retangulo(2.0, 4.0, 5.0, 6.0);

        for (Figura2D f : desenho )
        {
            area = f.CalcularArea();
            System.out.println("Area : " + area);
        }
    }
}
```

Atenção

- Uso da anotação @Override
  - Vale lembrar que a anotação @Override permite que os métodos sobrecarregados nas subclasses sejam criticados se existem equivalentes na superclasse
  - Isso garante que todos os métodos em cada subclasse e na superclasse apresentem mesma definição
  - E assim erros são evitados, é possivel codificar o programa anterior sem inconsistências nas chamadas do método polimórfico

Princípios de Orientação a Objetos

- Relembrando foco
- Princípios
- Como codificar?
- Discussões

Relembrando Foco

- Passar por polimorfismo permite discutir os 4 princípios básicos da Orientação a Objetos
  - Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo

Abstração

- A abstração diz respeito ao que vai ser considerado do mundo real ao representar e codificar o objeto
- Alguns aspectos são importantes, a identidade do objeto, as suas propriedades e os coportamentos
- Em síntese, ao estabelecermos as classes, os atributos e métodos estamos encontrando uma abstração do que precisamos representar do mundo real
- Fazendo uma analogia com o mundo real, quando acionamentos um liquidificador sabemos que ele tem 3 velocidades mas não precisamos de detalhes de como o liquidificador funciona nem como as velocidades são calculadas. Basta abstrairmos e operarmos os botões.

Encapsulamento

- Como já vimos, o encapsulamento visa deixar visível apenas o que vai ser necessário para o uso de uma classe, o restante fica dentro da “caixa preta”, a nossa classe
- Defende-se que assim protege-se as classes de algum mau uso de um programador

- Novamente, em analogia com o mundo real, ao operarmos uma televisão e o liquidificador do caso anterior não futucamos dentro deles, isso evita algum dano e defeito por mau uso.

Herança

- O princípio da herança permite que classes (subclasses) reaproveitem o que codificado nas classes pai

- Este princípio tem forte relação com o reuso de código pois ao se herdar de uma classe não se precisa recodificar propriedades (atributos) e comportamentos (métodos) das classes base

Polimorfismo

- O polimorfismo consiste na alteração do funcionamento interno de um método herdado de um objeto pai.

- Por exemplo, imagine um objeto genérico “Eletrodoméstico”, ele possui o método ou comportamento de “Ligar()”
- Dois objetos derivados “Televisão” e “Geladeira” são tipos de eletrodomésticos (herança) mas cada um pode ter o seu funcionamento interno de “Ligar()”, isto é, os métodos são implementados para cada caso

Conclusões

- Retomando Plano de Aula
- Revisão
- Para saber mais

Conquistamos a aula!

- ***Objetivos***
  - Exercitar os conceitos iniciais de Orientação a Objetos

- ***Tópicos***
  - Visão geral da Introdução à Orientação a Objetos
  - Classes e objetos
  - Declaração, instanciação e inicialização
  - Atributos, Métodos e Estado
  - Encapsulamento
  - Modificadores de acesso
  - Métodos getters e setters
  - Construtores