Java
Fundamentos e
Orientação a
Objetos

Java – Aula 6

Java Fundamentos e Orientação a Objetos

Construtores, Herança e polimorfismo, Interfaces

Programação Java

Métodos construtores

São os responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida.

Diferente dos métodos, um construtor tem o mesmo nome da classe e não tem um retorno declarado.

Mas, se nunca escrevemos esse construtor, quem o fez?

Sempre que você não criar um construtor para suas classes, o compilador fará isso para você.

Programação Java

Construtor

É uma operação especial da classe, que não é executada em nenhum momento na instanciação do objeto.

Os mais comuns são:

Iniciar valores dos atributos

Permitir ou obrigar que o objeto receba dados no momento de sua instanciação

Programação Java

Exemplo:

```
public class Carro
{
        /* CONSTRUTOR DA CLASSE Carro */
        public Carro()
        {
                //Faça o que desejar na construção do objeto
        }
}
```

O construtor sempre tem a seguinte assinatura:
modificadores de acesso (**public** nesse caso) + nome da classe (**Carro** nesse caso) + parâmetros (nenhum definido neste caso).
O construtor pode ter níveis como: **public, private ou protected**.

Programação Java

Exemplo:

```
public class Carro
{
        /* CONSTRUTOR DA CLASSE Carro */
        public Carro()
        {
                //Faça o que desejar na construção do objeto
        }
}
public class Aplicacao
{
        public static void main(String[] args)
        {
                //Chamamos o construtor sem nenhum parâmetro
                Carro fiat = new Carro();
        }
}
```

Programação Java

Exemplo:

```
public class Carro
{
        private String cor;
        private double preco;
        private String modelo;

        /* CONSTRUTOR PADRÃO */
        public Carro(){
        }

        /* CONSTRUTOR COM 2 PARÂMETROS */
        public Carro(String modelo, double preco){
        //Se for escolhido o construtor sem a COR do veículo, definimos a cor padrão como sendo PRETA
                this.cor = “PRETA”;
                this.modelo = modelo;
                this.preco = preco;
        }

        /* CONSTRUTOR COM 3 PARÂMETROS */
        public Carro(String cor, String modelo, double preco){
                this.cor = cor;
                this.modelo = modelo;
                this.preco = preco;
        }
}
```

Programação Java

Exemplo:

```
public class Carro
{
        private String cor; private double preco; private String modelo;

        public Carro(String modelo, double preco){
                this.cor = “PRETA”;
                this.modelo = modelo;
                this.preco = preco;
        }

        public Carro(String cor, String modelo, double preco){
                this.cor = cor;
                this.modelo = modelo;
                this.preco = preco;
        }
}

public class Aplicacao {
        public static void main(String[] args) {
                Carro civicPreto = new Carro(“New Civic”, 40000);

                Carro golfAmarelo = new Carro(“Azul”, “Golf”, 38000);
        }
}
```

Programação Java

Construtor

É possível especificar mais de um construtor na mesma classe (sobrecarga).

```
public class Caixa {

	public double saldo = 0;

	public Caixa(double saldo) {
		this.saldo = saldo;
	}

	public Caixa() {

	}

	void sacar(double valor) {
		this.saldo = saldo - valor;
	}

	void depositar(double valor) {
		this.saldo = saldo + valor;
	}

	double exibirSaldo() {
		return this.saldo;
	}
}
```

Método construtor inicializando o atributo saldo, constrói um objeto com o valor saldo setado

Método construtor padrão, constrói um objeto vazio

Pilares do Modelo OO

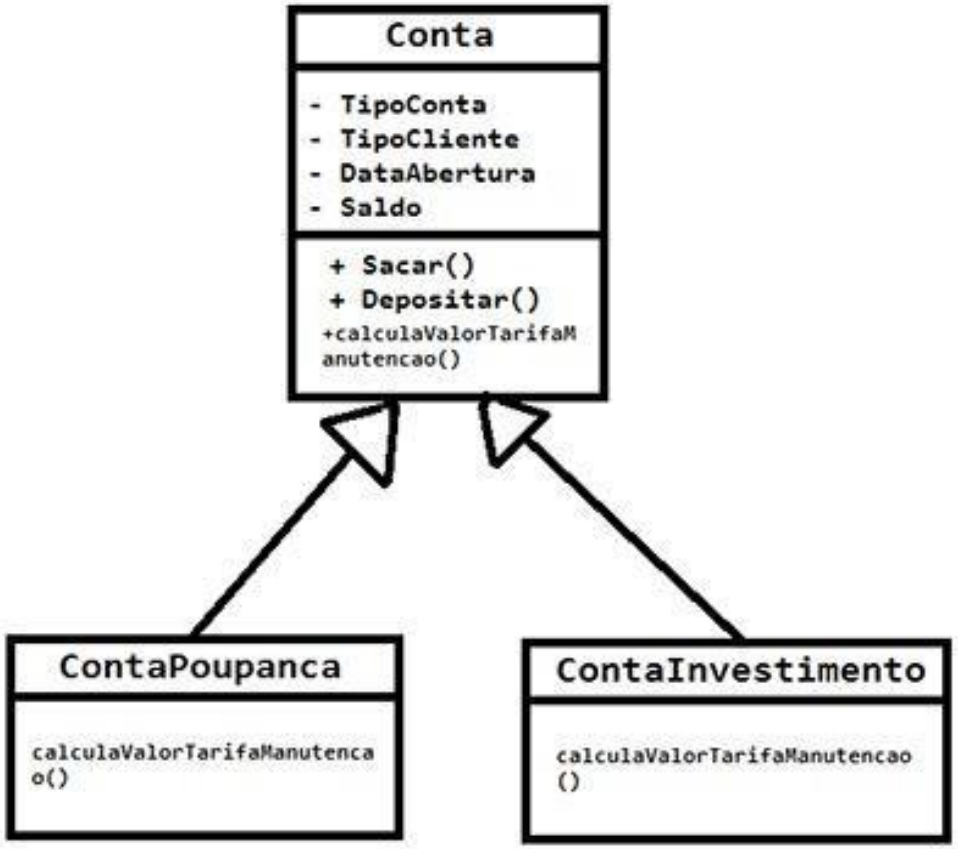
Herança

Trata-se de um mecanismo para derivar novas classes a partir da definição de classes existentes, que se dá por meio de um processo de refinamento. Uma classe derivada ou descendente herda os dados (atributos) e comportamento (métodos) da classe base ou ancestral ou ascendente.

A implementação da herança garante a reutilização de código que, além de economizar tempo e dinheiro, propicia mais segurança ao processo de desenvolvimento, posto que as funcionalidades da classe base já foram usadas e testadas.

Pilares do Modelo OO

Herança “Diagrama de Classe”



Pilares do Modelo OO

Polimorfismo

A palavra polimorfismo deriva do grego e significa “muitas formas”. A partir do momento em que uma classe herda atributos e métodos de uma (herança simples) ou mais (herança múltipla) classes base, ela tem o poder de alterar o comportamento de cada um desses procedimentos (métodos).

Isso amplia o poder do reaproveitamento de código promovido pela herança, permitindo que se aproveite alguns métodos e se altere outros. Um método com mesmo nome, em classes distintas, pode ter diferentes comportamentos.

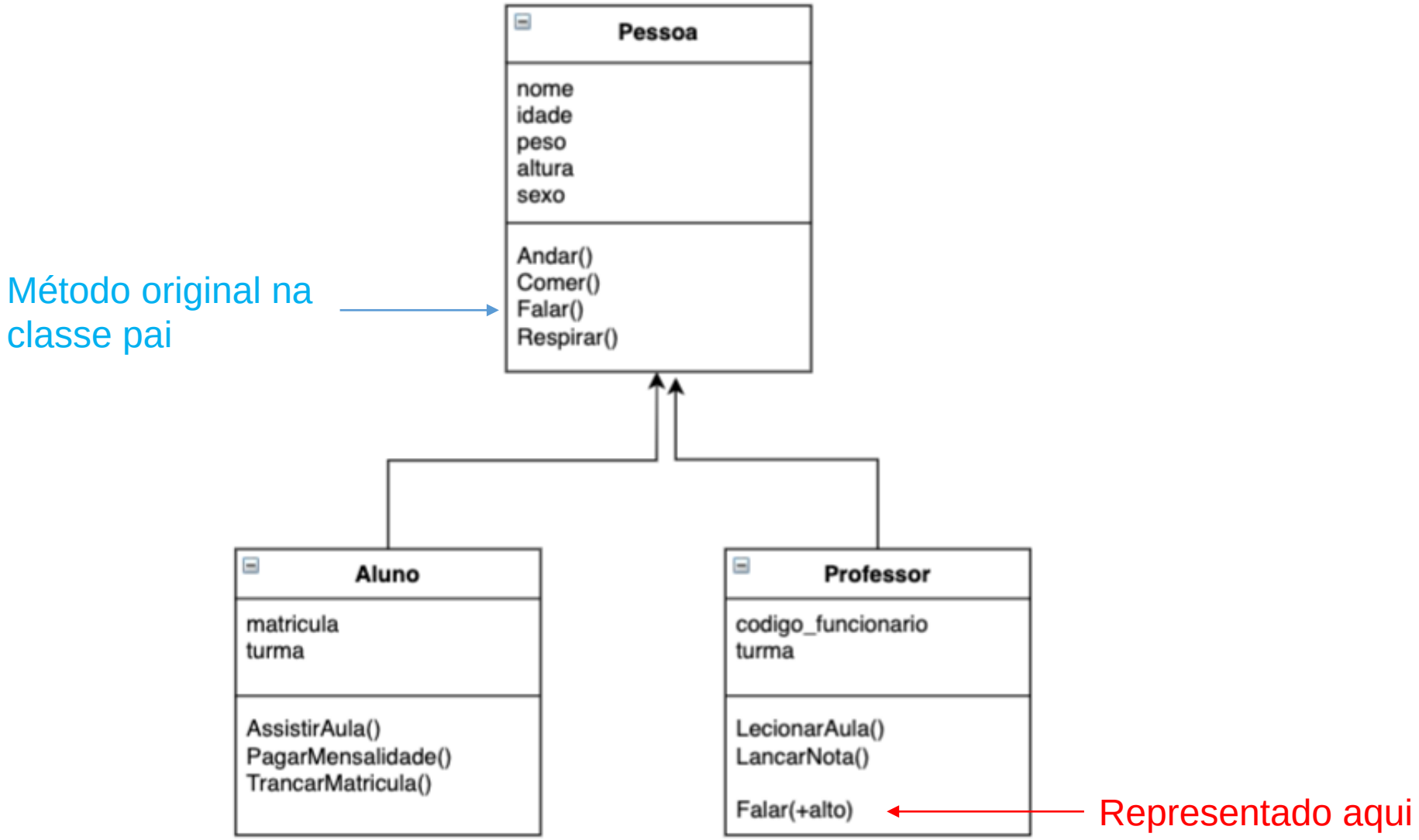
Pilares do Modelo OO

Polimorfismo “sobrescrita”

Por exemplo, uma ação normal de uma pessoa é falar, representado na classe Pessoa pelo método Falar(), mas um Professor em sala de aula tem que falar mais alto e claro para que os seus Alunos ouçam melhor. Então, o Professor não quer reaproveitar a forma como Falar () que acontece na classe pai.

Neste caso, descartamos todo o código existente e criamos "do zero" o método Falar(+ alto). Como estamos descartando tudo e escrevendo novamente, estamos fazendo uma **sobrescrita** do método Falar(). Isto pode ser interpretado no diagrama de classes da seguinte forma:

Pilares do Modelo OO



Pilares do Modelo OO

Polimorfismo “sobrecarga”

À **sobrecarga** de métodos acontece quando temos uma mesma função, que é executada de forma semelhante, com diferença apenas nos dados (parâmetros) que corrige.

Por exemplo, imagine que o Professor tenha que lançar como notas, e que podemos usar o método chamado **LancarNota ()**, porém, existem dois tipos de notas: as notas numéricas de 0 a 10 e o trabalho final que foi fornecido o conceito de A a F.

Pilares do Modelo OO

Polimorfismo “sobrecarga”

Desta forma, podemos ter dois **métodos LancarNota ()** com diferença em seu parâmetro de entrada:

- **LancarNota (numérico)**
- **LancarNota (letra)**

O primeiro método recebe como dado de entrada um **número** e executa uma operação qualquer para salvar a nota.

O segundo método recebe como dado de entrada uma **letra** e executa um procedimento diferente.

Mesmo sendo métodos diferentes e com tipo de dado de entrada diferente, eles possuem o mesmo nome.

Pilares do Modelo OO

Atividade Prática

Desenvolver um aplicação que contenha dois métodos **métodos LancarNota ()** com diferença em seu parâmetro de entrada:

- **LancarNota (numérico)**
- **LancarNota (letra)**

O primeiro método recebe como dado de entrada um **número** e executa uma operação qualquer para salvar a nota.

O segundo método recebe como dado de entrada uma **letra** e executa um procedimento diferente.

Mesmo sendo métodos diferentes e com tipo de dado de entrada diferente, eles devem possuir o mesmo nome.

Programação Java

Interface

É um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para **“obrigar”** a um determinado **grupo de classes** a ter **métodos ou propriedades em comum** para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente.

Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

Para acessar os métodos da interface, a interface deve ser "implementada" (meio como herdada) por outra classe com a **implements** palavra - chave (ao invés de extends).

Programação Java

Interface

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

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

Veja a seguir a classe ContaPoupanca implementando uma interface Conta:

```
class ContaPoupanca implements Conta {

      public void deposita(double valor) {

            this.saldo = this.saldo + valor

      }

      public void saca(double valor) {

            this.saldo = this.saldo - valor
      }
}
```