Programação Orientada a Objetos

ÍNDICE

Introdução à Programação Orientada a Objetos

O que é Programação Orientada a Objetos (POO)?

A Programação Orientada a Objetos (POO) é um paradigma de programação que organiza o código em torno de "objetos" — entidades que representam elementos do mundo real ou abstrações úteis. Cada objeto combina dados (chamados de atributos) e comportamentos (chamados de métodos). Este paradigma é amplamente utilizado por sua capacidade de organizar programas complexos de maneira clara e reutilizável.

Os quatro pilares da POO são:

- Abstração: Destacar os aspectos essenciais de um objeto, ignorando os detalhes irrelevantes.
- Encapsulamento: Proteger os dados internos de um objeto, permitindo o acesso apenas através de métodos controlados.
- Herança: Permitir que classes compartilhem atributos e comportamentos de outras classes.
- Polimorfismo: Capacitar objetos de diferentes classes a serem tratados como objetos de uma mesma classe-base.

Capítulo I: Classes e Objetos

Definição de Classe

Uma classe é um modelo ou molde para criar objetos. Ela define os atributos (dados) e métodos (funções) que os objetos criados a partir dela terão.

Exemplo em Java:

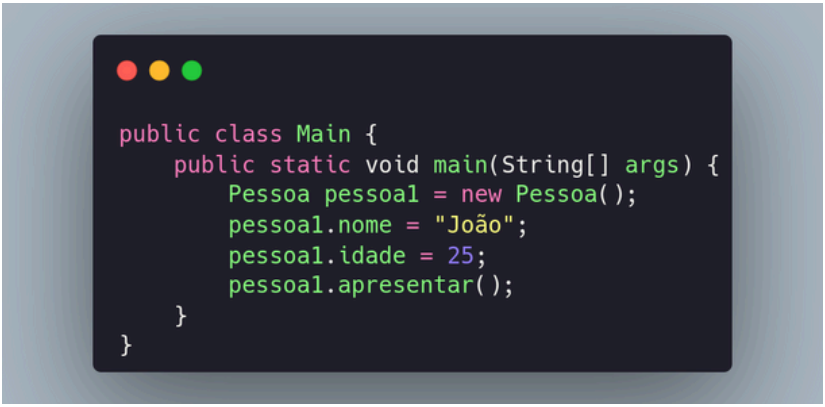
```java
public class Pessoa {
    String nome;
    int idade;

    void apresentar() {
        System.out.println("Olá, meu nome é " + nome + " e tenho " + idade + " anos.");
    }
}
  return go(f, seed, [])
}
```

Objetos

Um objeto é uma instância de uma classe, representando uma entidade específica.

Exemplo:

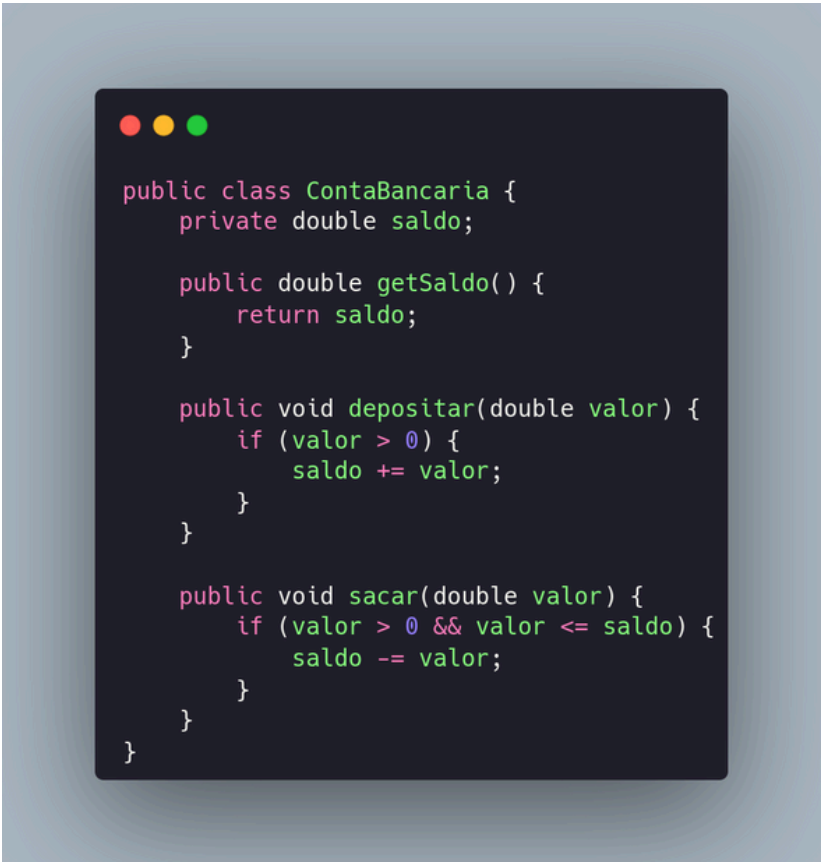


```java
public class Main {
    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa();
        pessoa1.nome = "João";
        pessoa1.idade = 25;
        pessoa1.apresentar();
    }
}
```

Capítulo 2: Encapsulamento

O encapsulamento é a prática de esconder os detalhes internos de uma classe, permitindo que sejam acessados e modificados apenas através de métodos específicos (getters e setters).

Exemplo:



```java
public class ContaBancaria {
    private double saldo;

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        }
    }

    public void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
        }
    }
}
```

Capítulo 3: Herança

A herança permite que uma classe (subclasse) reutilize atributos e métodos de outra classe (superclasse).

Exemplo:

```java
public class Animal {
    String nome;

    void comer() {
        System.out.println(nome + " está comendo.");
    }
}

public class Cachorro extends Animal {
    void latir() {
        System.out.println(nome + " está latindo.");
    }
}

public class Main {
    public static void main(String[] args) {
        Cachorro cachorro = new Cachorro();
        cachorro.nome = "Rex";
        cachorro.comer();
        cachorro.latir();
    }
}
```

Capítulo 4: Polimorfismo

O polimorfismo permite que o mesmo método tenha comportamentos diferentes dependendo do contexto. Isso pode ser feito através de:

1. **Sobrecarga de método:** Métodos com o mesmo nome, mas com assinaturas diferentes.
2. **Sobrescrita de método:** Subclasses redefinem métodos da superclasse.

Exemplo de sobrescrita:

```java
public class Animal {
    void fazerSom() {
        System.out.println("O animal faz um som.");
    }
}

public class Cachorro extends Animal {
    @Override
    void fazerSom() {
        System.out.println("O cachorro late.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal meuAnimal = new Cachorro();
        meuAnimal.fazerSom();
    }
}
```

Conclusão

A Programação Orientada a Objetos é um paradigma poderoso que facilita a organização e a manutenção de códigos complexos. Compreender conceitos como classes, objetos, encapsulamento, herança e polimorfismo é essencial para se tornar um programador mais eficiente e produtivo.