

Introdução à Programação Orientada a Objetos

Conceitos fundamentais com exemplos práticos em Java

1. O que é Programação Orientada a Objetos?

A Programação Orientada a Objetos (POO) é um paradigma de programação que organiza o código em torno de **objetos**, que representam entidades do mundo real ou conceitos abstratos do sistema. Em vez de estruturar o programa apenas como uma sequência de instruções que manipulam dados, a POO agrupa dados (atributos) e comportamentos (métodos) em uma mesma unidade, chamada **classe**.

Esse paradigma facilita a organização, a manutenção e a reutilização de código, principalmente em sistemas grandes e complexos. Linguagens como Java, C++, C#, Python e Kotlin adotam a POO como paradigma central ou como uma de suas principais abordagens. Neste texto, os exemplos práticos serão apresentados em **Java**, uma linguagem totalmente orientada a objetos e amplamente utilizada no mercado.

2. Os Quatro Pilares da POO

A Programação Orientada a Objetos se sustenta sobre quatro pilares fundamentais: **abstração**, **encapsulamento**, **herança** e **polimorfismo**. Nas próximas seções, cada um desses conceitos será explicado e ilustrado com exemplos de código em Java.

3. Classes e Objetos

Uma **classe** é um modelo (ou molde) que define os atributos e comportamentos que seus objetos terão. Um **objeto** é uma instância concreta dessa classe, criada em tempo de execução. No exemplo abaixo, a classe `Carro` define atributos como `marca` e `velocidade`, além de um método para `acelerar`.

```

public class Carro {
    private String marca;
    private int velocidade;

    public Carro(String marca) {
        this.marca = marca;
        this.velocidade = 0;
    }

    public void acelerar(int incremento) {
        this.velocidade += incremento;
        System.out.println(marca + " acelerando para " + velocidade + " km/h");
    }

    public static void main(String[] args) {
        Carro meuCarro = new Carro("Fusca");
        meuCarro.acelerar(20);
        meuCarro.acelerar(15);
    }
}

```

Saída do programa: Fusca acelerando para 20 km/h e em seguida Fusca acelerando para 35 km/h.

4. Encapsulamento

O encapsulamento consiste em restringir o acesso direto aos atributos de um objeto, protegendo seus dados internos. Em Java, isso é feito com os modificadores de acesso `private`, `protected` e `public`, combinados com métodos de acesso controlado (*getters* e *setters*).

```

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;
        } else {
            System.out.println("Saldo insuficiente.");
        }
    }
}

```

Note que o atributo `saldo` não pode ser alterado diretamente de fora da classe; toda modificação passa pelos métodos `depositar` e `sacar`, que aplicam regras de validação.

5. Herança

A herança permite que uma classe (subclasse) reaproveite atributos e métodos de outra classe (superclasse), promovendo reutilização de código. Em Java, isso é feito com a palavra-chave

extends.

```
public class Animal {
    protected String nome;

    public Animal(String nome) {
        this.nome = nome;
    }

    public void emitirSom() {
        System.out.println(nome + " faz um som.");
    }
}

public class Cachorro extends Animal {
    public Cachorro(String nome) {
        super(nome);
    }

    @Override
    public void emitirSom() {
        System.out.println(nome + " late: Au au!");
    }
}
```

A classe Cachorro herda o atributo nome e reaproveita o construtor da superclasse por meio de super (nome), além de sobrescrever o método emitirSom.

6. Polimorfismo

Polimorfismo significa "muitas formas": objetos de classes diferentes podem ser tratados de maneira uniforme por meio de uma referência comum (geralmente a superclasse ou uma interface), cada um respondendo de forma específica ao mesmo método.

```
public class Gato extends Animal {
    public Gato(String nome) {
        super(nome);
    }

    @Override
    public void emitirSom() {
        System.out.println(nome + " mia: Miau!");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal[] animais = {
            new Cachorro("Rex"),
            new Gato("Mimi")
        };

        for (Animal animal : animais) {
            animal.emitirSom();
        }
    }
}
```

Mesmo percorrendo um array do tipo Animal, cada objeto executa a versão de emitirSom definida em sua própria classe — esse comportamento é conhecido como polimorfismo em tempo de execução.

7. Conclusão

A Programação Orientada a Objetos oferece uma forma organizada e reutilizável de estruturar software, aproximando o código da forma como pensamos sobre entidades do mundo real. Os quatro pilares — abstração, encapsulamento, herança e polimorfismo — trabalham em conjunto para tornar os sistemas mais flexíveis, seguros e fáceis de manter. Dominar esses conceitos em Java é um passo essencial para qualquer desenvolvedor que deseja construir aplicações robustas e escaláveis.