A **Programação Orientada a Objetos (POO)** em Java é um paradigma de programação baseado no conceito de **objetos**, que são instâncias de **classes**. Java é uma linguagem totalmente orientada a objetos, o que significa que tudo em Java gira em torno de **classes** e **objetos**.

**Principais Conceitos da POO em Java**

**1. Classes e Objetos**

* **Classe**: é um molde (ou blueprint) que define atributos e comportamentos (métodos) de um objeto.
* **Objeto**: é uma instância de uma classe, ou seja, um elemento criado a partir dessa estrutura.

**Exemplo de classe e objeto:**

class Carro {

String marca;

int ano;

void buzinar() {

System.out.println("Biiip! Biiip!");

}

}

public class Main {

public static void main(String[] args) {

Carro meuCarro = new Carro(); // Criando um objeto

meuCarro.marca = "Toyota";

meuCarro.ano = 2023;

meuCarro.buzinar(); // Chamando o método

}

}

**2. Encapsulamento**

* Protege os dados internos de um objeto, permitindo o acesso apenas por meio de métodos.
* Usa **modificadores de acesso** como private, public, e protected.

**Exemplo de encapsulamento usando getters e setters:**

class Pessoa {

private String nome;

// Getter

public String getNome() {

return nome;

}

// Setter

public void setNome(String novoNome) {

nome = novoNome;

}

}

public class Main {

public static void main(String[] args) {

Pessoa pessoa = new Pessoa();

pessoa.setNome("João");

System.out.println(pessoa.getNome()); // João

}

}

**3. Herança**

* Permite que uma classe herde atributos e métodos de outra classe, promovendo **reutilização de código**.
* Usa a palavra-chave extends.

**Exemplo de herança:**

class Animal {

void fazerSom() {

System.out.println("O animal faz um som");

}

}

class Cachorro extends Animal {

void latir() {

System.out.println("Au Au!");

}

}

public class Main {

public static void main(String[] args) {

Cachorro meuCachorro = new Cachorro();

meuCachorro.fazerSom(); // Herda de Animal

meuCachorro.latir();

}

}

**4. Polimorfismo**

* Permite que um método tenha **diferentes comportamentos** dependendo da classe que o implementa.
* Pode ser feito por **sobrescrita** (override) ou **sobrecarga** (overload).

**Exemplo de sobrescrita de método (@Override):**

class Animal {

void fazerSom() {

System.out.println("Som genérico de animal");

}

}

class Gato extends Animal {

@Override

void fazerSom() {

System.out.println("Miau!");

}

}

public class Main {

public static void main(String[] args) {

Animal meuAnimal = new Gato();

meuAnimal.fazerSom(); // Miau!

}

}

**5. Abstração**

* Esconde os detalhes da implementação e expõe apenas as funcionalidades essenciais.
* Feita por meio de **classes abstratas** (abstract) ou **interfaces**.

**Exemplo de classe abstrata:**

abstract class Forma {

abstract void desenhar();

}

class Circulo extends Forma {

void desenhar() {

System.out.println("Desenhando um círculo");

}

}

public class Main {

public static void main(String[] args) {

Forma minhaForma = new Circulo();

minhaForma.desenhar();

}

}

**Resumo**

* **Classe e Objeto**: Uma classe define um modelo e um objeto é uma instância dessa classe.
* **Encapsulamento**: Protege os atributos de um objeto, permitindo o acesso controlado por métodos.
* **Herança**: Uma classe pode herdar características de outra classe.
* **Polimorfismo**: Métodos podem ter diferentes comportamentos dependendo do contexto.
* **Abstração**: Permite criar estruturas mais flexíveis escondendo detalhes da implementação.

Esses conceitos são essenciais para desenvolver programas modulares, reutilizáveis e fáceis de manter em Java! 🚀