POO em Java Sem Sofrência

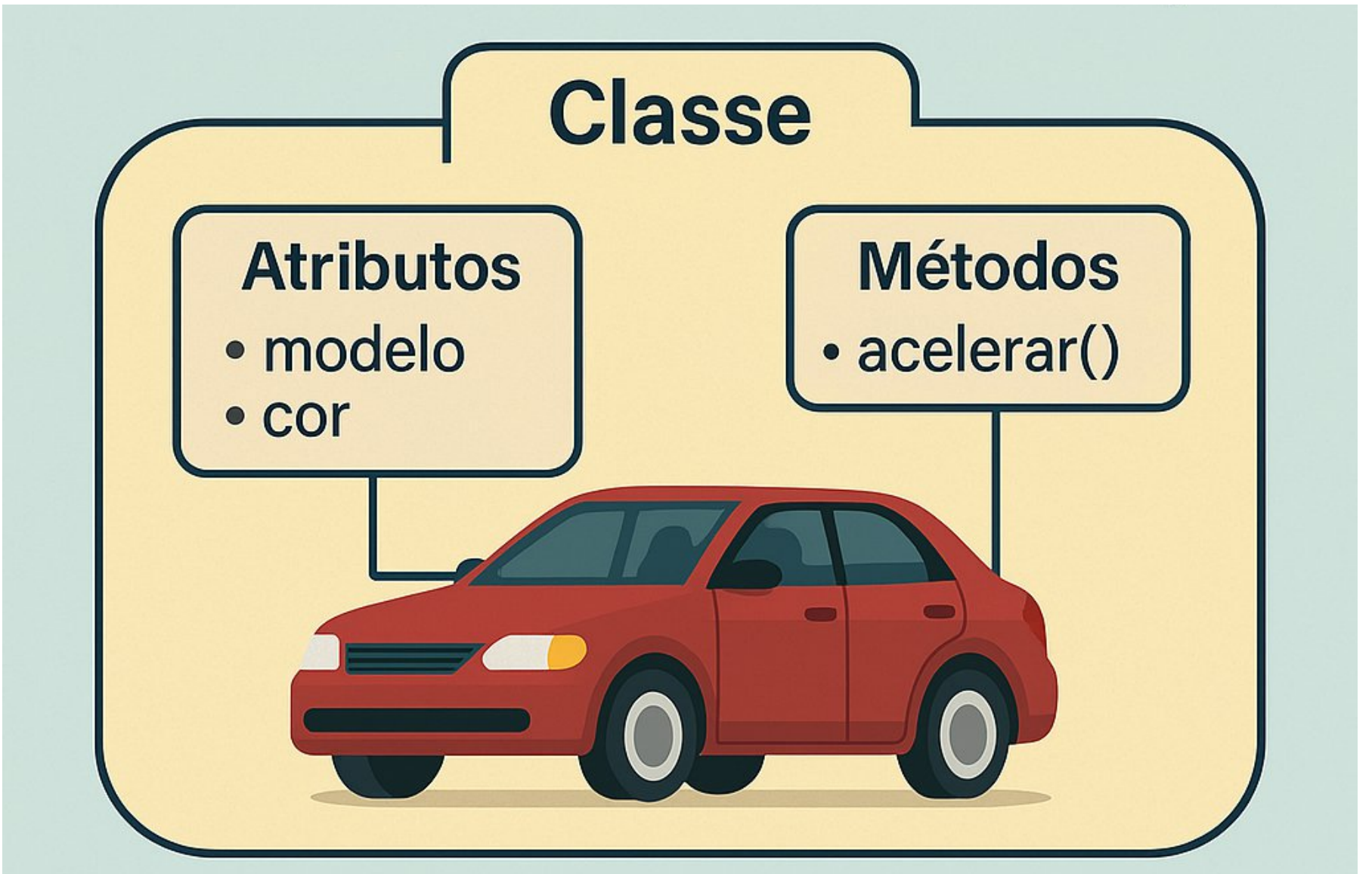
Entenda de Verdade, Codifique com Liberdade!

01

O QUE É POO?

Entendendo o que é POO de verdade

POO é um jeito de programar pensando nos elementos do mundo real. Os quatro pilares da POO são: Abstração, Encapsulamento, Herança e Polimorfismo. Em Java, tudo gira em torno de classes e objetos.
Ao invés de pensar só em funções soltas, a gente organiza tudo em **objetos** — que têm **atributos** (características) e **métodos** (ações).
Imagine um carro. Ele tem um modelo, uma cor e pode acelerar. Isso, em POO, seria uma classe com atributos e métodos.



Imagine um carro. Ele tem um modelo, uma cor e pode acelerar. Isso, em POO, seria uma classe com **atributos** e métodos

02

CLASSES E OBJETOS

Classes e Objetos: A Base de Tudo

Uma classe é um modelo que define atributos e métodos.
Um objeto é uma instância dessa classe.

Exemplo: Criando uma Classe Carro

```
public class Carro {
    // Atributos (características)
    String marca;
    String modelo;
    int ano;

    // Método (comportamento)
    public void acelerar() {
        System.out.println("O carro está acelerando!");
    }
}
```

Criando um Objeto Carro:

```
public class Main {
    public static void main(String[] args) {
        Carro meuCarro = new Carro(); // Instanciando um objeto
        meuCarro.marca = "Toyota";
        meuCarro.modelo = "Corolla";
        meuCarro.ano = 2022;

        meuCarro.acelerar(); // Saída: "O carro está acelerando!"
    }
}
```

03

ENCAPSULAMENTO

Encapsulamento: Protegendo os Dados

Encapsular significa esconder os detalhes internos de uma classe e controlar o acesso aos atributos usando getters e setters.

Exemplo: Classe ContaBancaria com Encapsulamento

```
public class ContaBancaria {
    private double saldo; // Atributo privado

    // Getter (obter valor)
    public double getSaldo() {
        return saldo;
    }

    // Setter (definir valor com validação)
    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        }
    }
}
```

Usando a Classe ContaBancaria

```
public class Main {
    public static void main(String[] args) {
        ContaBancaria conta = new ContaBancaria();
        conta.depositar(1000);
        System.out.println("Saldo: " + conta.getSaldo()); // Saída: Saldo: 1000.0
    }
}
```

04

HERANÇA

Herança: Reutilizando Código

Herança permite que uma classe (subclasse) herde atributos e métodos de outra (superclasse).

Exemplo: Veiculo (Superclasse) e Moto (Subclasse)

```
public class Veiculo {
    String marca;

    public void ligar() {
        System.out.println("Veículo ligado!");
    }
}

public class Moto extends Veiculo { // Herança
    int cilindradas;
}
```

Usando Herança

```
public class Main {
    public static void main(String[] args) {
        Moto minhaMoto = new Moto();
        minhaMoto.marca = "Honda";
        minhaMoto.cilindradas = 300;

        minhaMoto.ligar(); // Saída: "Veículo ligado!"
    }
}
```

05

POLIMORFISMO

Polimorfismo: Um Método, Múltiplas Formas

Polimorfismo permite que um mesmo método se comporte de maneira diferente em classes distintas.

Exemplo: Sobrescrita de Método (Animal e Cachorro)

```
public class Animal {
    public void emitirSom() {
        System.out.println("Som genérico");
    }
}

public class Cachorro extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Au au!"); // Sobrescrevendo o método
    }
}
```

Testando o Polimorfismo

```
public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Animal dog = new Cachorro();

        animal.emitirSom(); // Saída: "Som genérico"
        dog.emitirSom();   // Saída: "Au au!"
(polimorfismo)
    }
}
```

06

CONCLUSÃO

Conclusão: Por Que Usar POO em Java?

- Organização: Código mais limpo e modular.
- Reutilização: Herança e composição evitam repetição.
- Manutenção: Facilita atualizações e correções.

Agora você já tem uma base sólida para começar a aplicar POO em seus projetos Java!

07

EXERCÍCIOS

10 Exercícios Práticos de POO em Java

1. Criando uma Classe Pessoa:

Crie uma classe Pessoa com os atributos:

- nome (String)
- idade (int)
- cpf (String)

Adicione um método apresentar() que exibe:

"Olá, meu nome é [nome], tenho [idade] anos e meu CPF é [cpf]."

2. Encapsulamento em ContaCorrente:

Crie uma classe ContaCorrente com:

- Atributo privado saldo (double)
- Métodos públicos:
  - depositar(double valor)
  - sacar(double valor) (não permitir saque se saldo insuficiente)
  - getSaldo()

10 Exercícios Práticos de POO em Java

3. Herança: Animal e Gato:

Crie uma classe Animal com:

- Atributo nome
- Método emitirSom() (exibe "Som genérico")

Em seguida, crie uma classe Gato que herda de Animal e sobrescreve emitirSom() para exibir "Miau!".

4. Polimorfismo: Forma e Subclasses:

Crie uma classe abstrata Forma com um método abstrato calcularArea().
Depois, crie duas subclasses:

- Quadrado (com atributo lado)
- Circulo (com atributo raio)

Implemente calcularArea() em cada uma.

10 Exercícios Práticos de POO em Java

5. Classe Produto com Construtor:

Crie uma classe Produto com:

- Atributos: nome, preco, quantidadeEmEstoque
- Um construtor que recebe esses valores
- Método calcularValorTotalEmEstoque() (retorna preco * quantidadeEmEstoque)

6. Interface Autenticavel:

Crie uma interface Autenticavel com o método autenticar(String senha).

- Implemente essa interface em uma classe Usuario que verifica se a senha é "1234".

7. Composição: Computador e Memoria:

Crie uma classe Memoria com:

- capacidadeEmGB (int)
- tipo (String, ex: "DDR4")

Depois, crie uma classe Computador que contém uma Memoria como atributo.

10 Exercícios Práticos de POO em Java

8. Sobrecarga de Métodos em Calculadora:

Crie uma classe Calculadora com métodos sobrecarregados:

- somar(int a, int b)
- somar(double a, double b)
- somar(String a, String b) (concatena as strings)

9. Classe Estática Contador:

Crie uma classe Contador com:

- Um atributo estático total (int)
- Métodos estáticos incrementar() e getTotal()

10. Tratamento de Exceções em Divisao:

Crie uma classe Divisao com um método dividir(int a, int b) que:

- Retorna a / b
- Lança ArithmeticException se b == 0

08

EXERCÍCIOS RESOLVIDOS

10 Respostas Resolvidas

1. Classe Pessoa:

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

    public void apresentar() {
        System.out.printf(
        "Olá, meu nome é %s, tenho %d anos e meu CPF é %s.%n",
        nome, idade, cpf
        );
    }
}
```

2. Encapsulamento em ContaCorrente:

```
public class ContaCorrente {
    private double saldo;

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

    public void sacar(double valor) {
        if (valor <= saldo) {
            saldo -= valor;
        } else {
            System.out.println("Saldo insuficiente!");
        }
    }

    public double getSaldo() {
        return saldo;
    }
}
```

10 Respostas Resolvidas

3. Herança: Animal e Gato:

```
public class Animal {
    String nome;

    public void emitirSom() {
        System.out.println("Som genérico");
    }
}
public class Gato extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Miau!");
    }
}
```

4. Polimorfismo: Forma e Subclasses:

```
public abstract class Forma {
    public abstract double calcularArea();
}

public class Quadrado extends Forma {
    double lado;

    @Override
    public double calcularArea() {
        return lado * lado;
    }
}

public class Circulo extends Forma {
    double raio;

    @Override
    public double calcularArea() {
        return Math.PI * raio * raio;
    }
}
```

10 Respostas Resolvidas

5. Classe Produto com Construtor:

```
public class Produto {
    String nome;
    double preco;
    int quantidadeEmEstoque;

    public Produto(String nome, double preco, int quantidadeEmEstoque) {
        this.nome = nome;
        this.preco = preco;
        this.quantidadeEmEstoque = quantidadeEmEstoque;
    }

    public double calcularValorTotalEmEstoque() {
        return preco * quantidadeEmEstoque;
    }
}
```

6. Interface Autenticavel:

```
public interface Autenticavel {
    boolean autenticar(String senha);
}

public class Usuario implements Autenticavel {
    @Override
    public boolean autenticar(String senha) {
        return senha.equals("1234");
    }
}
```

10 Respostas Resolvidas

7. Composição: Computador e Memoria:

```
public class Memoria {
    int capacidadeEmGB;
    String tipo;
}

public class Computador {
    Memoria memoria; // Composição
    String modelo;

    public Computador() {
        memoria = new Memoria();
    }
}
```

8. Sobrecarga de Métodos em Calculadora:

```
public class Calculadora {
    public int somar(int a, int b) {
        return a + b;
    }

    public double somar(double a, double b) {
        return a + b;
    }

    public String somar(String a, String b) {
        return a + b;
    }
}
```

10 Respostas Resolvidas

9. Classe Estática Contador:

```
public class Contador {
    private static int total = 0;

    public static void incrementar() {
        total++;
    }

    public static int getTotal() {
        return total;
    }
}
```

10. Tratamento de Exceções em Divisao:

```
public class Divisao {
    public static int dividir(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Divisão por zero!");
        }
        return a / b;
    }
}
```

OBRIGADO POR LER ATÉ AQUI

Esse Ebook foi gerado por IA, e diagramado por humano.
O passo a passo se encontra no meu Github.

Esse conteúdo foi gerado com fins didáticos de construção, não foi realizado uma validação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.

https://github.com/GNunnes