Os Piratas da Codificação

Rumo ao Grande JavaPoo

RUAN RODRIGUES

Programação Orientada a Objetos com Java: Herança e Polimorfismo
sum dolor sit amet, consectetuer

Introdução à POO

- **A Programação Orientada a Objetos (POO)** é um paradigma de programação que utiliza "objetos" – instâncias de classes – para organizar e estruturar o código. Dois conceitos fundamentais da POO são a Herança e o Polimorfismo. Neste eBook, vamos explorar esses conceitos com exemplos práticos em Java.

01

HERANÇA: REUTILIZAÇÃO E EXTENSÃO DE CÓDIGOTOREDS

- **A herança permite que uma classe (subclasse) herde características (atributos e métodos) de outra classe (superclasse). Isso facilita a reutilização de código e a criação de hierarquias.**

HERANÇA: REUTILIZAÇÃO E EXTENSÃO DE CÓDIGO

- A herança permite que uma classe (subclasse) herde características (atributos e métodos) de outra classe (superclasse). Isso facilita a reutilização de código e a criação de hierarquias.

- Exemplo Real: Sistema de Veículos
- Imagine que estamos desenvolvendo um sistema para gerenciar diferentes tipos de veículos.

JvaPoo -Ruan Rodrigues

```java
// Classe base Veiculo
class Veiculo {
    String marca;
    String modelo;

    void ligar() {
        System.out.println("O veiculo está ligado.");
    }
}

// Subclasse Carro que herda de Veiculo
class Carro extends Veiculo {
    int numeroDePortas;

    void abrirPortaMalas() {
        System.out.println("Porta-malas aberto.");
    }
}

// Subclasse Moto que herda de Veiculo
class Moto extends Veiculo {
    boolean temCarenagem;

    void empinar() {
        System.out.println("A moto está empinando!");
    }
}
```

02

POLIMORFISMO: FLEXIBILIDADE NO USO DE OBJETOSTILIZAÇÃO E EXTENSÃO DE CÓDIGOTOREDS

- **A herança permite que uma classe (subclasse) herde características (atributos e métodos) de outra classe (superclasse). Isso facilita a reutilização de código e a criação de hierarquias.**

POLIMORFISMO: FLEXIBILIDADE NO USO DE OBJETOS

- O polimorfismo permite que um objeto de uma subclasse seja tratado como um objeto de sua superclasse. Isso traz flexibilidade ao código, permitindo a troca de objetos de forma mais simples e dinâmica.
- Exemplo Real: Gerenciamento de Veículos
- Continuando com nosso sistema de veículos, vamos ver como o polimorfismo pode ser aplicado.

JvaPoo -Ruan Rodrigues

```java
class Main {
    public static void main(String[] args) {
        Veiculo meuCarro = new Carro();
        Veiculo minhaMoto = new Moto();

        // Polimorfismo em ação
        meuCarro.ligar();
        minhaMoto.ligar();

        // Usando um array de Veiculos
        Veiculo[] veiculos = {meuCarro, minhaMoto};
        for (Veiculo veiculo : veiculos) {
            veiculo.ligar();
        }
    }
}
```

03

Encapsulamento: Protegendo e Organizando o Código

- **O encapsulamento é um dos princípios fundamentais da Programação Orientada a Objetos (POO). Ele ajuda a proteger os dados da classe e a controlar como esses dados são acessados e modificados.**

ENCAPSULAMENTO: PROTEGENDO E ORGANIZANDO O CÓDIGO

- O encapsulamento é um dos princípios fundamentais da Programação Orientada a Objetos (POO). Ele ajuda a proteger os dados da classe e a controlar como esses dados são acessados e modificados.

- Exemplo Real: Sistema de Conta Bancária
- Imagine que estamos desenvolvendo um sistema de contas bancárias. Vamos usar o encapsulamento para proteger os dados da conta.

JvaPoo -Ruan Rodrigues

```
class ContaBancaria {
    private String titular;
    private double saldo;

    public ContaBancaria(String titular, double saldoInicial) {
        this.titular = titular;
        this.saldo = saldoInicial;
    }

    public String getTitular() {
        return titular;
    }

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
            System.out.println("Depósito de R$" + valor + "
realizado com sucesso.");
        } else {
            System.out.println("Valor de depósito inválido.");
        }
    }

    public void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
            System.out.println("Saque de R$" + valor + " realizado
com sucesso.");
        } else {
            System.out.println("Saldo insuficiente ou valor de saque
inválido.");
        }
    }
}

class SistemaBancario {
    public static void main(String[] args) {
        ContaBancaria conta = new ContaBancaria("João Silva",
1000.0);
        conta.depositar(500.0);
        conta.sacar(200.0);
        System.out.println("Saldo atual: R$" + conta.getSaldo());
    }
}
```

04

Abstração: Focando no Essencial

- **A abstração é outro princípio fundamental da POO. Ela permite que se concentre nos aspectos essenciais de um objeto, ignorando detalhes mais complexos ou irrelevantes.**

ABSTRAÇÃO: FOCANDO NO ESSENCIAL

- A abstração é outro princípio fundamental da POO. Ela permite que se concentre nos aspectos essenciais de um objeto, ignorando detalhes mais complexos ou irrelevantes.

- Exemplo Real: Sistema de Pedidos

- Vamos criar um sistema simples de gerenciamento de pedidos, utilizando abstração para focar nos aspectos essenciais dos pedidos.

JvaPoo -Ruan Rodrigues

```
abstract class Pedido {
    protected int numeroPedido;
    protected double valorTotal;

    public Pedido(int numeroPedido, double valorTotal) {
        this.numeroPedido = numeroPedido;
        this.valorTotal = valorTotal;
    }

    public abstract void processarPedido();
}

// Subclasse PedidoOnline
class PedidoOnline extends Pedido {
    private String enderecoEntrega;

    public PedidoOnline(int numeroPedido, double valorTotal, String
enderecoEntrega) {
        super(numeroPedido, valorTotal);
        this.enderecoEntrega = enderecoEntrega;
    }

    @Override
    public void processarPedido() {
        System.out.println("Processando pedido online número " +
numeroPedido);
        System.out.println("Endereço de entrega: " +
enderecoEntrega);
        System.out.println("Valor total: R$" + valorTotal);
    }
}

// Subclasse PedidoPresencial
class PedidoPresencial extends Pedido {
    private String localRetirada;

    public PedidoPresencial(int numeroPedido, double valorTotal,
String localRetirada) {
        super(numeroPedido, valorTotal);
        this.localRetirada = localRetirada;
    }

    @Override
    public void processarPedido() {
        System.out.println("Processando pedido presencial número " +
numeroPedido);
        System.out.println("Local de retirada: " + localRetirada);
        System.out.println("Valor total: R$" + valorTotal);
    }
}

class SistemaPedidos {
    public static void main(String[] args) {
        Pedido pedido1 = new PedidoOnline(123, 250.0, "Rua A, 123");

        Pedido pedido2 = new PedidoPresencial(456, 150.0, "Loja B");


        pedido1.processarPedido();
        pedido2.processarPedido();
    }
}
```

Conclusões

RUAN ODRIGUES

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/Ruandev2/prompts-recipe-to-create-a-ebook