Universidade Federal da Paraíba
Centro de Ciências Aplicadas à Educação
Programação Orientada a Objetos
Professor: Matheus Barbosa

Atividade: Cirurgia de Código — Operando o SOLID

Objetivo:

Analisar exemplos de código com violações de princípios SOLID, identificar o princípio quebrado e propor uma solução refatorada.

Regras:

1. Você receberá problemas de código, cada um com uma violação de um princípio SOLID.
2. Para cada problema, identifique qual princípio foi quebrado, explique o problema e proponha uma refatoração.

Resumo dos princípios SOLID:

S — Single Responsibility: Uma classe deve ter apenas um motivo para mudar.
O — Open/Closed: Aberto para extensão, fechado para modificação.
L — Liskov Substitution: Subtipos devem poder substituir seus tipos-base sem quebrar o comportamento.
I — Interface Segregation: Interfaces específicas, não forçar métodos desnecessários.
D — Dependency Inversion: Depender de abstrações, não concreções.

Problema 1

```
public class RelatorioService {
  public void gerarRelatorio() {
  // código para gerar relatório
  }
  public void enviarPorEmail(String destinatario) {
  // código para enviar e-mail
  }
  public void salvarNoBanco() {
  // código para salvar no banco
  }
}
```

Problema 2

```
public class Usuario {
    public void salvarNoBanco() {
        // código de persistência
    }
    public void autenticar(String usuario, String senha) {
        // lógica de autenticação
    }
    public void enviarEmailBoasVindas() {
        // lógica de envio de e-mail
    }
}
```

Problema 3

```
public class GerenciadorDeProjeto {
    public void criarProjeto() { /* ... */ }
    public void calcularCusto() { /* ... */ }
    public void gerarRelatorioFinanceiro() { /* ... */ }
}
```

Problema 4

```
public class CalculadoraFrete {
  public double calcular(String tipoEntrega) {
    if (tipoEntrega.equals("CORREIOS")) {
      return 10.0;
    } else if (tipoEntrega.equals("TRANSPORTADORA")) {
      return 20.0;
    } else if (tipoEntrega.equals("DRONE")) {
      return 50.0;
    }
    return 0.0;
  }
}
```

Problema 5

```
public class CalculadoraPreco {
    public double calcularPreco(String categoria) {
        if (categoria.equals("LIVRO")) {
            return 10.0;
        } else if (categoria.equals("EBOOK")) {
            return 5.0;
        } else if (categoria.equals("DVD")) {
            return 20.0;
        }
        return 0.0;
    }
}
```

Problema 6

```
public class RelatorioGenerator {
    public void gerar(String tipo) {
        if (tipo.equals("PDF")) {
            // gera PDF
        } else if (tipo.equals("CSV")) {
            // gera CSV
        } else if (tipo.equals("XLS")) {
            // gera XLS
        }
    }
}
```

Problema 7

```java
public class Retangulo {
  protected double largura;
  protected double altura;
  public void setLargura(double largura) { this.largura = largura; }
  public void setAltura(double altura) { this.altura = altura; }
  public double getArea() { return largura * altura; }
}
public class Quadrado extends Retangulo {
  @Override
  public void setLargura(double largura) {
    this.largura = largura;
    this.altura = largura;
  }
  @Override
  public void setAltura(double altura) {
    this.altura = altura;
    this.largura = altura;
  }
}
```

Problema 8

```java
public class Motor {
    public void ligar() { /* ... */ }
}

public class MotorEletrico extends Motor {
    @Override
    public void ligar() {
        throw new UnsupportedOperationException("Motor elétrico precisa de bateria carregada!");
    }
}
```

Problema 9

```java
public class Conta {
    public void sacar(double valor) { /* ... */ }
}

public class ContaPoupanca extends Conta {
    @Override
    public void sacar(double valor) {
        if (valor > 1000) {
            throw new RuntimeException("Limite de saque excedido!");
        }
        super.sacar(valor);
    }
}
```

Problema 10

```java
public interface IImpressora {
  void imprimirDocumento(String doc);
  void enviarFax(String numero);
}
public class ImpressoraDigital implements IImpressora {
  public void imprimirDocumento(String doc) {
  // imprime normalmente
  }
  public void enviarFax(String numero) {
  // não suporta fax, método vazio ou erro
  }
}
```

Problema 11

```
public interface Veiculo {
    void dirigir();
    void voar();
    void navegar();
}

public class Carro implements Veiculo {
    public void dirigir() { /* ok */ }
    public void voar() { /* não aplicável */ }
    public void navegar() { /* não aplicável */ }
}
```

Problema 12

```
public interface DispositivoMultifuncional {
    void imprimir();
    void escanear();
    void enviarFax();
}

public class ScannerSimples implements DispositivoMultifuncional {
    public void imprimir() { /* não aplicável */ }
    public void escanear() { /* ok */ }
    public void enviarFax() { /* não aplicável */ }
}
```

Problema 13

```
public class PedidoService {
  public void processarPedido() {
    EmailSMTP email = new EmailSMTP();
    email.enviar("cliente@exemplo.com", "Pedido recebido!");
  }
}
public class EmailSMTP {
  public void enviar(String destino, String mensagem) {
  // código para envio SMTP
  }
}
```

Problema 14

```
public class RelatorioService {
    private PDFGenerator generator = new PDFGenerator();

    public void gerarRelatorio() {
        generator.gerar();
    }
}
```

Problema 15

```
public class PagamentoService {
    public void processar() {
        BoletoProcessor processor = new BoletoProcessor();
        processor.pagar();
    }
}
```

Problema 16

```
public class LojaOnline {
  public void processarCompra(String pagamento) {
    if (pagamento.equals("PIX")) {
        // processa PIX
    } else if (pagamento.equals("Cartao")) {
        // processa Cartão
    }
    enviarEmailConfirmacao();
    gerarNotaFiscal();
  }
  private void enviarEmailConfirmacao() {
    // código SMTP fixo
  }
  private void gerarNotaFiscal() {
     // gera e salva nota fiscal
  }
}
```