****

Práticas com SOLID:

Boas Práticas de Programação.

Nome: Julio Eduardo de Carvalho Almeron

Leandro Mesquita

## 1. Princípio da Responsabilidade Única(SRP):

Processador de Encomendas

Problema:

* A classe ProcessadorEncomendas.java era responsável por processar encomendas e também executar a lógica principal do programa, ou seja violando o SRP.

Correção:

* Foram criadas classes separadas para realizar cada um dos processos do Processador de Encomendas

Motivo:

* Cada classe deve ter uma única responsabilidade. A separação da lógica principalmente que o torna o torna o código mais modular e fácil de manter na programação.

## 2. Princípios Aberto/Fechado (OCP):

Sistema de Pagamento

Problema:

* A classe SistemaPagamento.java utilizava uma série de instruções de if-else para determinar o método de pagamento(cartão, PIX, boleto). Isso tornava o código difícil de estender, uma vez que para cada novo tipo de pagamento adicionado, a classe do sistema de pagamento deveria ser alterada..

Correção:

* Foi introduzida a interface RealizarPagamento.java, que foi implementada por classes específicas para cada método de pagamento da conta:
* PagamentoCartao.java;
* PagamentoPix.java;
* PagamentoBoleto.java.
* A classe SistemaPagamentos.java foi refatorada para aceitar objetos do tipo RealizarPagamento.java, eliminando as necessidade dos if - else.

Motivo:

* O OCP garante que a classe seja aberta para extensão sem precisar modificar o código existente, reduzindo o risco de introduzir erros e diminuindo a necessidade de manutenção em outras partes do código.

## 3. Princípio da Substituição de Liskov(LSP):

Conta Bancária

Problema:

* A classe ContaPoupanca.java estendia a classe ContaBancaria.java, mas implementou o método o método sacar de forma inadequada, lançando uma exceção(UnsupportedOperationException) para indicar que os saques não eram permitidos diretamente. Isso violava o LSP, pois a classe derivada (ContaPoupanca.java) não podia ser usada como substituta da classe base (ContaBancaria.java) sem alterar o comportamento esperado da classe.

Correção:

* A classe ContaPoupanca.java foi refatorada para não implementar o método sacar, já que ele não é aplicável a contas da poupança.
* Foi criada a classe ContaCorrente.java, que implementa o método para sacar de forma adequada.
* A classe ContaBancaria.java foi ajustada para conter apenas métodos genéricos aplicáveis a todos os tipos de contas, como depositar e getSaldo.

Motivo:

* O LSP garante que classes derivadas possam ser usadas como substitutas de suas classes base sem alterar o comportamento esperado do programa. Com a refatoração, cada classe agora reflete corretamente suas responsabilidades, respeitando o comportamento esperado do código.

## 4. Princípio da Segregação de Interfaces (ISP):

Veículos

Problema:

* A interface da classe Veiculo.java forçava classes como Carro e Avião a implantação dos métodos que não eram importantes para elas funcionarem, como voar e navegar. Isso violava o ISP, pois as classes eram obrigadas a se depender de métodos que eles não utilizavam em funcionamento.

Correção:

* A interface da classe Veiculo.java foi dividida em interfaces menores e mais específicas:
* Veiculo.java (com o método de direção terrestre).
* VeiculoQueVoa.java (com o método de voar).
* VeiculoQueNavega.java (com o método de navegação aquática).

Motivo:

* O ISP garante que as classes não sejam forçadas a se depender dos métodos que não utilizamos na subclasse dos veículos. Com a divisão das interfaces, de cada classe da implementação apenas os métodos necessários que tornam o código mais modular e fácil de se manter.

Concluímos após essas alterações que os códigos ficaram mais limpos, manuteníveis e menos propensos a erros por parte dos desenvolvedores, mostrando a importância das boas práticas por meio dos princípios SOLID.