Boas práticas

Princípios SOLID - Luis Afonso Mineo

S - **Single-responsibility Principle**

Para resolver o problema de responsabilidade única foi criado uma classe *Encomenda*, a qual define o modelo de uma encomenda, e posteriormente é passada como um parâmetro para os cálculos.
A segunda mudança foi separar o *Exportador de Arquivos* em uma classe própria, chamada *ExportadorDeArquivos*. Essa classe será instanciada em outra classe.
A terceira mudança foi ajustar a classe *ProcessadorEncomendas*, onde a mesma será responsável apenas por processar uma nova encomenda. Para isso, seu método *processar*, recebe um objeto de *Encomenda*, faz os cálculos necessários, e ao final instancia *ExportadorDeArquivos* e chama o método de *salvarEmArquivo*.

O - Open-closed principle

A solução implementada foi criar uma nova interface, a qual possui a assinatura do método comum a todos os tipos de pagamentos.
Depois, os tipos de pagamentos foram separados cada um em sua própria classe. Estas classes, por sua vez, implementam a interface *IRealizarPagamento*, assim também implementando o método da interface. Esse método é sobrescrito, alterando o tipo de pagamento descrito.
Por último, a classe *SistemaPagamento*, foi remodelada, onde agora possui um método que irá receber um valor, e um objeto de *IRealizarPagamento*, assim com essa única classe podemos passar um objeto de qualquer um dos tipos de pagamentos, pois eles implementam a interface em qustão.

L - Liskov Substitution Principle

Para evitar o problema evidenciado pelo princípio de substituição de Liskov, foi criada uma nova interface *IConta*, a qual possui todas as assinaturas de métodos necessários à uma conta bancária.
Depois, ambes as classes de conta implementaram a interface e sobrescreveram os métodos. A principal diferença agora, é no método *sacar*, agora, o mesmo, retorna um *boolean*, assim, a conta corrente que permite sacar, realiza a lógica de saque e retorna *true*, enquanto a conta poupançao, que não o permite, apenas retorna *false*, e com isso podemos lidar com a situação proposta.

I - Interface Segregation Principle

Neste exemplo simplesmente separamos a interface *Veiculo*, em outras 3 interfaces, as quais descrevem de forma mais precisa o tipo de associação a ser feita com os métodos presentes na interface *veiculo*. As novas interfaces são *VeiculoAereo*, *VeiculoNautico* e *VeiculoRodoviario*, assim cada uma das interfaces recebeu o método apropriado e poderão ser implementadas por classes apropriadas. No caso do exemplo, a classe *Carro* se encaixa em *VeiculoRodoviario*, assim, a classe implementa a interface, e agora possui apenas o método *dirigir*.