Roteiro SOLID 02

Os roteiros a serem desenvolvidos visam trazer a percepção da evolução no processo de desenvolvimento de software. Por isso, daremos sequência ao projeto iniciado no Roteiro 01 criando vários pacotes, onde cada pacote representa a evolução da implementação deste projeto.

Neste caso iremos explorar um dos princípios do SOLID :

O - Open/Closed Principle (Princípio do Aberto/Fechado)

As entidades de software (classes, módulos, etc.) devem estar abertas para extensão, mas fechadas para modificação.

O princípio tem como objetivo trabalhar diretamente com a manutenção das classes e tem como premissa a seguinte afirmativa:

“Você deve ser capaz de estender um comportamento de uma classe sem a necessidade de modificá-lo.”

Ou seja, as entidades de software como classes, módulos, funções, etc., devem estar abertas para extensão, porém fechadas para modificação.

Em outras palavras significa que esta classe pode ter seu comportamento alterado com facilidade quando necessário, sem a alteração do seu código fonte. Essa extensão pode ser feita através de herança, interface e composição.

Início do projeto – Pacote : roteiro2.parte1

1 – Dê sequência ao mesmo projeto no NetBeans chamado **SOLIDroteiros**

2 – Dentro do projeto criar um pacote chamado **roteiro2.parte1**

Cenário :

Iremos montar a simulação de um sistema simples para gerenciamento de pedidos em uma loja de roupas. O gerente da loja deseja que seja possível a aplicação de descontos no valor do pedido, mas à princípio não tem regras muito claras de quando os vendedores podem aplicar os descontos. Os motivos para conceder ou não os descontos são variados, então cada caso é avaliado de forma isolada. A única regra neste sentido é que o desconto não pode ultrapassar 30%.

3 – Diante deste cenário a primeira classe a ser criada é **PedidoService** conforme o código abaixo.

Esta classe à princípio é responsável por:

- Processamento de pedidos
- Cálculo de preços (integração com as regras de negócio)
    - Para efeitos didáticos deixamos itens de compra com valores fixos no código
        - Camiseta = R$ 50,00
        - Calça = R$ 100,00
        - Jaqueta = R$ 200,00
- Aplicação de Descontos (Também com regra de negócio)
    - Os descontos não podem ultrapassar 30%

```
package roteiro2.parte1;

import java.util.List;

public class PedidoService {
   private List<String> itens;
   private double total;

   public PedidoService(List<String> itens) {
      this.itens = itens;
      this.total = calcularTotal();
   }

   public double aplicarDesconto(double desconto) {
      if (desconto > 0 && desconto <= 0.3) {
         return total - (total * desconto);
      }
      return total;
   }
   private double calcularTotal() {
      double total = 0;
      for (String item : itens) {
         if (item.equals("Camiseta")) total += 50.0;
         else if (item.equals("Calça")) total += 100.0;
         else if (item.equals("Jaqueta")) total += 200.0;
      }
      return total;
   }

   public void processarPedido(double desconto) {
      double valorFinal = aplicarDesconto(desconto);
```

5 - Neste mesmo pacote crie a classe a seguir para que possamos testar a criação de um pedido

```
package roteiro2.parte1;

import java.util.Arrays;

public class TestePedido {

    public static void main(String[] args) {
        PedidoService pedido = new PedidoService(Arrays.asList("Camiseta", "Calça",
"Jaqueta"));
        pedido.processarPedido(0.1); // Desconto de 10%
        pedido.processarPedido(0.0); // Sem desconto
        pedido.processarPedido(0.4); // Desconto de 40%, mas não aplicado
    }
```

Análise Crítica

Podemos observar que nesta modelagem a classe **PedidoService viola o princípio de Responsabilidade Única** visto no roteiro anterior, mas daremos ênfase aqui a **violação do princípio do Aberto/Fechado.** De a cordo com o cenário descrito já percebemos que a aplicação dos descontos é uma questão crítica, já que não possui regras claras e certamente pode trazer problemas futuros.

- Os vendedores podem aplicar descontos livremente. Mesmo existindo no código um limite de controle para os 30% de desconto, essa interação com o usuário não parece muito adequada.
- Não existe controle ou padronização para a aplicação dos descontos. Isso pode trazer alguma dificuldade para evolução do projeto quando for necessário a inclusão de novos controles neste sentido. (**Difícil extensibilidade**)

Pacote : roteiro2.parte2

1 – No mesmo projeto crie o pacote roteiro2.parte2

2 – Copie todas as classes criadas na parte1 para o novo pacote

3 – Para melhorar o controle e permitir a extensibilidade nesta modelagem, devemos criar a interface **RegraDesconto**, que representará as regras de desconto da loja.

```
package roteiro2.parte2;

public interface RegraDesconto {
   double calcular(double total);
}
```

4 – Seguindo na melhoria da modelagem crie a classe **DescontoLivre** que implementa a interface **RegraDesconto**. Observe que o cálculo do desconto que originalmente estava na classe PedidoService foi trazido para esta nova classe.

```
package roteiro2.parte2;

public class DescontoLivre implements RegraDesconto {
   private double desconto;

   public DescontoLivre(double desconto) {
      this.desconto = desconto;
   }

   @Override
   public double calcular(double total) {
      if (desconto > 0 && desconto <= 0.3) { // Máx. 30%
         return total - (total * desconto);
      }
      return total;
   }
}
```

5 – Precisamos agora fazer os ajustes necessários refatorando a classe **PedidoService**

```
package roteiro2.parte1;

import java.util.List;

public class PedidoService {
    private List<String> itens;
    private double total;
    private RegraDesconto regraDesconto;

    public PedidoService(List<String> itens, RegraDesconto regraDesconto) {
        this.itens = itens;
        this.regraDesconto = regraDesconto;
        this.total = calcularTotal();
    }

    public double aplicarDesconto(double desconto) {
        if (desconto > 0 && desconto <= 0.3) {
            return total - (total * desconto);
        }
        return total;
    }

    private double calcularTotal() {
        double total = 0;
        for (String item : itens) {
            if (item.equals("Camiseta")) total += 50.0;
            else if (item.equals("Calça")) total += 100.0;
            else if (item.equals("Jaqueta")) total += 200.0;
        }
        return total;
    }

    public void processarPedido(double desconto) {
        double valorFinal = regraDesconto.calcular(total);
        System.out.println("Pedido processado. Valor final com desconto: R$ " +
```

6 – Relembrando o roteiro 1, qual foi a técnica aplicada nesta refatoração utilizada no construtor da classe **PedidoService** ?

7 - Agora tente fazer os ajustes necessários na classe **TestePedido**

Análise Crítica

Faça a sua análise crítica desta etapa da refatoração. O Princípio Aberto/Fechado já está contemplado nesta etapa ?

Mudança no cenário :

Conforme previsto no cenário original, o gerente da loja percebeu a necessidade de um maior controle na liberação dos descontos. Pretende categorizar os descontos da seguinte forma :

- **Desconto Padrão – 10% de desconto**
- **Desconto VIP – 20% de desconto**

Pacote : roteiro2.parte3

1 – No mesmo projeto crie o pacote roteiro2.parte3

2 – Copie as classes seguintes classes da parte2 para o novo pacote : **PedidoService**, **RegraDesconto**, **TestePedido**.

3 – Vamos agora tentar fazer os ajustes solicitados nesta mudança de cenário. Para isso, devemos criar 2 novas classes que implementam a interface **RegraDesconto**. São elas : **DescontoPadrão**, **DescontoVIP**.

```
package roteiro2.parte3;

public class DescontoPadrao implements RegraDesconto {

    @Override
    public double calcular(double total) {
        return total * 0.9; // 10% de desconto
    }
}
```

```
package roteiro2.parte3;

public class DescontoVIP implements RegraDesconto {

    @Override
    public double calcular(double total) {
        return total * 0.8; // 20% de desconto
    }
}
```

4 – Agora vamos fazer os ajustes necessários na classe **TestePedido**

```
package roteiro2.parte3;

import java.util.Arrays;

public class TestePedido {
   public static void main(String[] args) {

       // Teste com desconto padrão
       RegraDesconto descontoPadrao = new DescontoPadrao();
       PedidoService pedidoPadrao = new PedidoService(Arrays.asList("Camiseta", "Jaqueta"),
descontoPadrao);
       pedidoPadrao.processarPedido();

       // Teste com desconto VIP
       RegraDesconto descontoVIP = new DescontoVIP();
       PedidoService pedidoVIP = new PedidoService(Arrays.asList("Camiseta", "Calça", "Jaqueta"),
descontoVIP);
```

Nova mudança no cenário :

O gerente da loja percebeu uma certa resistência por parte dos vendedores, pois antes tinham muita liberdade na negociação com os clientes, e agora têm apenas 2 opções de negociação (Desconto Padrão ou Desconto VIP). O gerente acredita que loja está passando por um momento de transição, e precisa promover um treinamento dos vendedores. Enquanto isso, ele deseja que o sistema volte a ter uma certa liberdade, mas também não quer perder o trabalho implementado com a categorização dos descontos.

5 – Diante desta nova mudança de cenário, que tipo de ajuste você acha necessário na modelagem atual ?