**Roteiro SOLID 05**

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

Neste caso iremos explorar um dos princípios do SOLID :

**D - Dependency Inversion Principle (Princípio da Inversão de Dependências)**

“Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações”.

“Abstrações não devem depender de detalhes. Detalhes devem depender de abstrações”

O princípio da inversão de dependência ajuda a reduzir o acoplamento no código, facilitando a manutenção e extensibilidade. Isso é feito ao garantir que as dependências sejam gerenciadas por meio de **interfaces** ou **classes abstratas** em vez de classes concretas.

**Pacote : roteiro5.parte1**

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

2 – Dentro do projeto criar um pacote chamado **roteiro5.parte1**

**Cenário :**

**Vamos modelar um sistema de notificação para uma plataforma digital. A plataforma deve enviar notificações aos usuários por diferentes canais :**

* **E-mail**
* **SMS**
* **Push Notification (notificação no aplicativo)**

3 – Vamos iniciar criando a classe **NotificacaoService**, conforme o código abaixo.

package roteiro5.parte1;

public class NotificacaoService {

public void enviarNotificacaoEmail(String mensagem) {

System.out.println("Enviando e-mail: " + mensagem);

}

public void enviarNotificacaoSMS(String mensagem) {

System.out.println("Enviando SMS: " + mensagem);

}

public void enviarNotificacaoPush(String mensagem) {

System.out.println("Enviando Push Notification: " + mensagem);

}

}

4 – Agora crie a classe **TesteNotificacao**, conforme o código abaixo para que façamos os testes.

package roteiro5.parte1;

public class TesteNotificacao {

public static void main(String[] args) {

NotificacaoService notificacao = new NotificacaoService();

notificacao.enviarNotificacaoEmail("Bem-vindo ao sistema!");

notificacao.enviarNotificacaoSMS("Seu código de verificação é 1234.");

notificacao.enviarNotificacaoPush("Você tem uma nova mensagem.");

}

}

**Análise Crítica**

A classe **NotificacaoService** **viola o princípio da Inversão de Dependencia.,** pois depende diretamente das implementações concretas**.**

Se quisermos adicionar ou remover um tipo de notificação, precisaremos modificar o código existente, aumentando os riscos de bugs e dificultando a manutenção.

Consegue identificar se esta modelagem viola algum outro princípio do SOLID ?

**Pacote : roteiro5.parte2**

1 – No mesmo projeto crie o pacote roteiro5.parte2

2 – Na ideia de tentar seguir o princípio da Inversão de Dependências. Devemos criar interface Notificacao, conforme o código abaixo.

package roteiro5.parte2;

public interface Notificacao {

void enviar(String mensagem);

}

3 – Agora crie as classes correspondentes a cada tipo de notificação. São elas : **NotificacaoEmail**, **NotificacaoSMS**, **NotificacaoPush**.

Cada uma das classes criadas deve implementar a interface Notificacao. Veja abaixo a classeNotificacaoEmail e faça o mesmo nas outras duas classes.

package roteiro5.parte2;

public class NotificacaoEmail implements Notificacao {

@Override

public void enviar(String mensagem) {

System.out.println("Enviando e-mail: " + mensagem);

}

}

4 – Refatore o código da classe **NotificacaoService**, aplicando a injeção de dependência

package roteiro5.parte2;

public class NotificacaoService {

private Notificacao notificacao;

public NotificacaoService(Notificacao notificacao) {

this.notificacao = notificacao;

}

public void enviarNotificacao(String mensagem) {

this.notificacao.enviar(mensagem);

}

}

4 – Agora faça os testes na classe **TesteNotificacao**, conforme o código abaixo.

package roteiro5.parte2;

public class TesteNotificacao {

public static void main(String[] args) {

Notificacao email = new NotificacaoEmail();

Notificacao sms = new NotificacaoSMS();

Notificacao push = new NotificacaoPush();

NotificacaoService service1 = new NotificacaoService(email);

NotificacaoService service2 = new NotificacaoService(sms);

NotificacaoService service3 = new NotificacaoService(push);

service1.enviarNotificacao("Bem-vindo ao sistema!");

service2.enviarNotificacao("Seu código de verificação é 1234.");

service3.enviarNotificacao("Você tem uma nova mensagem.");

}

}

**Pacote : roteiro5.parte3**

1 – No mesmo projeto crie o pacote roteiro5.parte3

2 –A modelagem da parte2 pareceu interessante, será que poderíamos reduzir o número de classes ? A proposta aqui é eliminar a classe **NotificacaoService**.

Ou Seja, copie as classes da parte2 para o novo pacote, **exceto** a classe **NotificacaoService**.

3 – Agora faça os testes na classe **TesteNotificacao**, conforme o código abaixo.

package roteiro5.parte3;

public class TesteNotificacao {

public static void main(String[] args) {

Notificacao email = new NotificacaoEmail();

Notificacao sms = new NotificacaoSMS();

Notificacao push = new NotificacaoPush();

email.enviar("Bem-vindo ao sistema!");

sms.enviar("Seu código de verificação é 1234.");

push.enviar("Você tem uma nova mensagem.");

}

}

**Análise Crítica**

Faça a sua análise comparando as propostas da parte2 e parte3

Onde está sendo aplicado o Princípio da Inversão de dependência ?

Temos a violação de outros princípios do SOLID ?

**Mudança no cenário**

**Precisamos evoluir o projeto para que as notificações possam ser enviadas também via WhatsApp**

**Pacote : roteiro5.parte4**

1 – No mesmo projeto crie o pacote roteiro5.parte3

2 –Faça as implementações necessárias para atender ao novo cenário

**Análise Crítica**

Qual abordagem resolveu seguir ? Parte2 ou Parte3 ? Por quê ?