ATIVIDIDADES SOBRE INJEÇÃO DE DEPENDÊNCIA

Exercício 1: "Sistema de Gerenciamento de Pedidos de Restaurantes"

Objetivo: Implementar um sistema de gerenciamento de pedidos de um restaurante utilizando os princípios de orientação a objetos e injeção de dependência. O sistema deve ser modular, permitindo a substituição fácil de diferentes implementações, como tipos de pagamento e sistemas de notificação.

Requisitos:

1. Classe Consumidora: `RestaurantOrderSystem`

- Esta é a classe principal onde o sistema é executado. Ela deve gerenciar a criação de pedidos e o fluxo completo de processamento, desde a criação do pedido até a notificação do cliente.

2. Interface de Dependência: `PaymentProcessor`

- Define métodos para processar pagamentos, como `processPayment(double amount)`.

- Várias implementações devem ser criadas para diferentes métodos de pagamento, como `CreditCardProcessor`, `PayPalProcessor`, e `BitcoinProcessor`.

3. Interface de Dependência: `Notifier`

- Define métodos para enviar notificações ao cliente, como `sendNotification(String message)`.

- Várias implementações devem ser criadas para diferentes métodos de notificação, como `EmailNotifier`, `SMSNotifier`, e `PushNotifier`.

4. Classe Principal: `Order`

- Representa um pedido com detalhes como `orderId`, `items`, `totalAmount`, etc.

- Utiliza injeção de dependência para processar pagamentos e enviar notificações. O processamento de pagamento e a notificação devem ser configuráveis, permitindo a substituição de implementações conforme necessário.

5. Configuração de Dependência:

- Utilize um padrão de injeção de dependência (por exemplo, por construtor ou método setter) para fornecer implementações específicas de `PaymentProcessor` e `Notifier` para a classe `Order`.

- Crie um arquivo de configuração ou utilize um contêiner de injeção de dependências (como o Spring, se estiver usando Java) para gerenciar essas dependências.

Tarefa:

- Passo 1: Definir as interfaces `PaymentProcessor` e `Notifier`.

- Passo 2: Implementar as classes `CreditCardProcessor`, `PayPalProcessor`, `BitcoinProcessor`, `EmailNotifier`, `SMSNotifier`, e `PushNotifier`.

- Passo 3: Criar a classe `Order` que recebe instâncias de `PaymentProcessor` e `Notifier` via injeção de dependência.

- Passo 4: Implementar a classe consumidora `RestaurantOrderSystem` que gerencia o ciclo de vida do pedido.

- Passo 5: Configurar a injeção de dependência utilizando um contêiner de dependência ou manualmente.

Resultado Esperado:

Ao executar o sistema, ele deve:

- Criar um pedido com detalhes específicos.

- Processar o pagamento utilizando a implementação injetada.

- Enviar uma notificação utilizando o método configurado.

- Demonstrar a facilidade de substituição de diferentes implementações de pagamento e notificação sem alterar a lógica principal do sistema.

Exercício 2: "Sistema de Automação Residencial"

Objetivo: Desenvolver um sistema de automação residencial que utiliza princípios de orientação a objetos e injeção de dependência para gerenciar dispositivos inteligentes em uma casa. O sistema deve ser flexível para adicionar, remover ou substituir dispositivos e métodos de controle.

Requisitos:

1. Classe Consumidora: `HomeAutomationSystem`

- Esta é a classe principal onde o sistema é executado. Ela deve gerenciar dispositivos inteligentes em uma casa, como lâmpadas, termostatos e câmeras de segurança.

2. Interface de Dependência: `SmartDevice`

- Define métodos genéricos para dispositivos inteligentes, como `turnOn()`, `turnOff()`, e `status()`.

- Implemente várias classes que representam dispositivos inteligentes diferentes, como `SmartLight`, `SmartThermostat`, e `SmartCamera`.

3. Interface de Dependência: `ControlMethod`

- Define métodos para diferentes tipos de controle, como `sendCommand(String command)`.

- Implemente diferentes métodos de controle, como `VoiceControl`, `AppControl`, e `RemoteControl`.

4. Classe Principal: `Room`

- Representa um cômodo da casa que pode conter múltiplos dispositivos inteligentes.

- Utiliza injeção de dependência para definir quais dispositivos estão disponíveis em cada cômodo e quais métodos de controle serão utilizados.

5. Configuração de Dependência:

- Use injeção de dependência para fornecer instâncias de `SmartDevice` e `ControlMethod` para a classe `Room`.

- Utilize um padrão de fábrica ou um contêiner de injeção de dependências para gerenciar a criação de dispositivos e métodos de controle.

Tarefa:

- Passo 1: Definir as interfaces `SmartDevice` e `ControlMethod`.

- Passo 2: Implementar as classes `SmartLight`, `SmartThermostat`, `SmartCamera`, `VoiceControl`, `AppControl`, e `RemoteControl`.

- Passo 3: Criar a classe `Room` que utiliza injeção de dependência para definir os dispositivos e métodos de controle.

- Passo 4: Implementar a classe consumidora `HomeAutomationSystem` que gerencia o controle dos dispositivos em diferentes cômodos.

- Passo 5: Configurar a injeção de dependência utilizando um contêiner de dependência ou manualmente.

Resultado Esperado:

Ao executar o sistema, ele deve:

- Controlar diferentes dispositivos em vários cômodos da casa.

- Demonstrar o controle de dispositivos usando diferentes métodos, como controle por voz, aplicativo ou controle remoto.

- Mostrar a flexibilidade de adicionar, remover ou substituir dispositivos e métodos de controle sem alterar a lógica principal do sistema.