
FIAP

Domain Driven Design
using Java

AGENDA

1. Padrão Adapter, Inversão de Controle e Injeção de Dependência
2. Exercícios

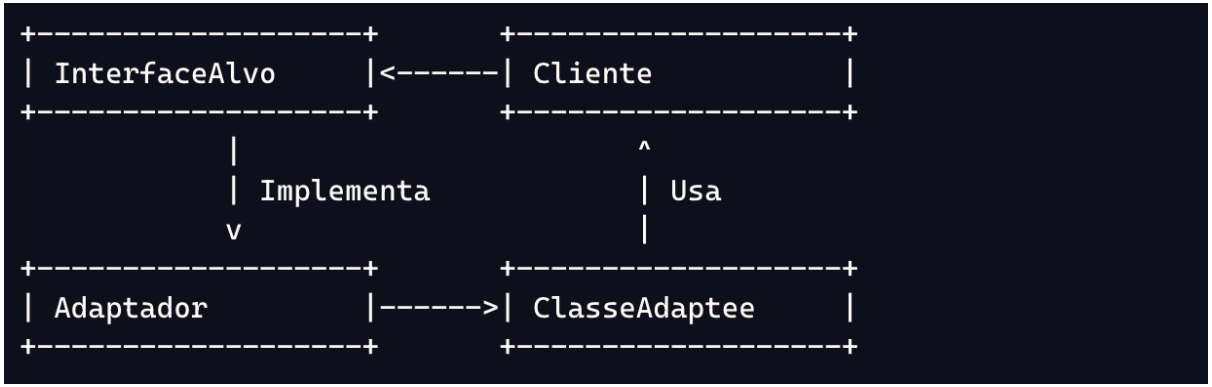
Padrão Adapter

Prof. Evando Borges

O Que é o Adapter?

- Definição:
  - Permite que duas interfaces incompatíveis trabalhem juntas, convertendo a interface de uma classe para a interface esperada por outra.
- Objetivo:
  - Resolver problemas de incompatibilidade de interfaces entre sistemas existentes e novos sistemas.
- Analogia:
  - Um adaptador de tomadas permite que dispositivos com plugs diferentes sejam conectados a uma fonte de energia.

Estrutura do Padrão Adapter



- **Cliente:** Classe que utiliza a interface esperada.
- **InterfaceAlvo:** A interface que o cliente espera.
- **Adaptador:** Adapta a ClasseAdaptee para ser compatível com o cliente.
- **ClasseAdaptee:** Classe existente que será adaptada.

Exemplo em Java – Padrão Adapter

```java
// Interface esperada pelo sistema
public interface PagamentoOnline {
    void processarPagamento(double valor);
}

// Classe existente com uma interface incompatível
public class SistemaPagamentoAntigo {
    public void realizarPagamento(double valor) {
        System.out.println("Pagamento realizado no sistema antigo: R$ "
    }
}
```

Exemplo em Java – Padrão Adapter

```java
// Adapter para conectar as duas interfaces
public class PagamentoAdapter implements PagamentoOnline {
    private SistemaPagamentoAntigo sistemaAntigo;

    public PagamentoAdapter(SistemaPagamentoAntigo sistemaAntigo) {
        this.sistemaAntigo = sistemaAntigo;
    }

    @Override
    public void processarPagamento(double valor) {
        sistemaAntigo.realizarPagamento(valor);
    }
}

// Uso do Adapter
public class Main {
    public static void main(String[] args) {
        SistemaPagamentoAntigo sistemaAntigo = new SistemaPagamentoAntigo()
        PagamentoOnline pagamento = new PagamentoAdapter(sistemaAntigo);
        pagamento.processarPagamento(200.0);
    }
}
```

Benefícios do Padrão Adapter

- **Flexibilidade:** Integra sistemas novos a sistemas antigos sem alterar o código legado.
- **Desacoplamento:** O cliente não precisa conhecer os detalhes da ClasseAdaptee.
- **Reutilização:** Facilita o uso de bibliotecas ou frameworks externos.

Inversão de Controle (IoC) e Injeção de Dependência (DI)

Prof. Evando Borges

Contextualização

- Por que estudar IoC e DI?
  - Projetos modernos exigem código modular, flexível e fácil de manter.
  - IoC e DI são princípios fundamentais para desenvolver sistemas desacoplados e testáveis.

- Principais Objetivos:
  - Reduzir dependências diretas entre classes.
  - Garantir que componentes possam ser configurados e substituídos dinamicamente.

Conceitos Gerais

- Inversão de Controle:
  - Delegação da responsabilidade de instanciar e gerenciar objetos para um framework ou container.
- Injeção de Dependência:
  - Implementação de IoC, onde dependências são fornecidas a uma classe em vez de serem criadas internamente.
- Relacionamento:
  - DI é um mecanismo prático para aplicar IoC.

Inversão de Controle (IoC)

Prof. Evando Borges

O Que Inversão de Controle?

- Definição:
  - Princípio de design que transfere o controle do fluxo de criação e gerenciamento de objetos para um container ou framework.
- Motivação:
  - O código principal não deve se preocupar com instâncias específicas.
  - Promove flexibilidade e facilita substituições.

Exemplo Prático de IoC

- **Sem IoC:** A classe controla diretamente a criação de objetos.

```
public class PedidoService {
    private ClienteService clienteService;

    public PedidoService() {
        this.clienteService = new ClienteService(); // Forte acoplamento
    }

    public void criarPedido() {
        clienteService.processarCliente();
    }
}
```

Exemplo Prático de IoC

- **Com IoC:** Um framework gerencia o controle e cria as instâncias necessárias.

```
public class PedidoService {
    private ClienteService clienteService;

    @Autowired
    public PedidoService(ClienteService clienteService) { // Inversão de controle
        this.clienteService = clienteService;
    }

    public void criarPedido() {
        clienteService.processarCliente();
    }
}
```

- **Framework (Exemplo Spring):** O container gerencia o ciclo de vida dos objetos e injeta dependências automaticamente.

Benefícios do IoC

- **Desacoplamento:** Reduz dependências rígidas entre classes.
- **Flexibilidade:** Permite substituir objetos dinamicamente (ex.: para testes).
- **Centralização:** Frameworks como Spring oferecem controle centralizado sobre instâncias.

Injeção de Dependência (DI)

Prof. Evando Borges

O Que Injeção de Dependência?

- Definição:
  - Técnica usada para fornecer dependências externas a uma classe em vez de criá-las diretamente dentro da classe.
- Principais Tipos de DI:
  - **Construtor:** Dependências são fornecidas via parâmetros do construtor.
  - **Setter:** Dependências são configuradas através de métodos setters.
  - **Campo:** Frameworks injetam dependências diretamente nos atributos da classe (Ex.: *@Autowired* no Spring).

Exemplo Prático de DI

- **DI via Construtor:**

```java
public class EmailService {
    public void enviarEmail(String mensagem) {
        System.out.println("Enviando email: " + mensagem);
    }
}

public class PedidoService {
    private EmailService emailService;

    public PedidoService(EmailService emailService) {
        this.emailService = emailService; // Injeção de dependência
    }

    public void criarPedido(String mensagem) {
        emailService.enviarEmail(mensagem);
    }
}

// Uso
EmailService emailService = new EmailService();
PedidoService pedidoService = new PedidoService(emailService);
pedidoService.criarPedido("Pedido criado com sucesso!");
```

Exemplo Prático de DI

- **DI via Spring Framework:**

```java
@Component
public class EmailService {
    public void enviarEmail(String mensagem) {
        System.out.println("Enviando email: " + mensagem);
    }
}

@Component
public class PedidoService {
    private EmailService emailService;

    @Autowired
    public PedidoService(EmailService emailService) { // DI automatizada
        this.emailService = emailService;
    }

    public void criarPedido(String mensagem) {
        emailService.enviarEmail(mensagem);
    }
}
```

Benefícios do IoC

- **Modularidade:** Substituir dependências é mais simples.
- **Testabilidade:** Permite injetar mocks ou dependências falsas para testes unitários.
- **Manutenção:** Facilita modificações e atualizações no código.

Relacionando IoC e DI

IoC	DI
Princípio geral de delegação de controle.	Aplicação prática do IoC.
Permite que frameworks gerenciem objetos.	Define como as dependências são fornecidas.
Exemplo: Spring Container gerencia objetos.	Exemplo: @Autowired injeta dependências automaticamente.

Implementação IoC e DI com Spring – Exemplo Completo

```java
@SpringBootApplication
public class Aplicacao {
    public static void main(String[] args) {
        SpringApplication.run(Aplicacao.class, args);
    }
}

@Component
public class EmailService {
    public void enviarEmail(String mensagem) {
        System.out.println("Email enviado: " + mensagem);
    }
}
```

Implementação IoC e DI com Spring – Exemplo Completo

```java
@Component
public class PedidoService {
    private EmailService emailService;

    @Autowired
    public PedidoService(EmailService emailService) {
        this.emailService = emailService; // DI e IoC
    }

    public void processarPedido() {
        emailService.enviarEmail("Pedido processado!");
    }
}
```

Implementação IoC e DI com Spring – Exemplo Completo

```java
@RestController
@RequestMapping("/pedidos")
public class PedidoController {
    private PedidoService pedidoService;

    @Autowired
    public PedidoController(PedidoService pedidoService) {
        this.pedidoService = pedidoService;
    }

    @PostMapping
    public ResponseEntity<String> criarPedido() {
        pedidoService.processarPedido();
        return ResponseEntity.ok("Pedido criado com sucesso!");
    }
}
```

Referências

- **Livro:** *Spring in Action* - Craig Walls.
- **Documentação Spring Framework:** https://spring.io
- **Artigo sobre DI:** https://martinfowler.com/articles/injection.html


FIAP