**02 - Padrões de Projetos Estruturais - Respostas**

**João César de Carvalho Júnior**

**Thieres Marcos**

**Introdução**

Neste segundo trabalho, foram aplicados **padrões estruturais** no projeto ClinicaGoF. Esses padrões tratam da composição de classes e objetos para formar estruturas maiores e mais flexíveis, promovendo organização e reutilização de código.

**Padrões Estruturais Utilizados**

**1. Adapter – Integração com sistema legado**

**Aplicação:**

Criado um adaptador SistemaAntigoAdapter para integrar o novo sistema com módulos legados da clínica (ex: sistema de notificações antigo).

**Justificativa:**

Permite o reaproveitamento de código legado, integrando-o sem alterar sua estrutura original.

**Código:**

java

CopiarEditar

// Sistema legado

public class SistemaAntigoNotificacao {

public void enviarEmailAntigo(String msg) {

System.out.println("Enviando pelo sistema antigo: " + msg);

}

}

// Adapter

public interface Notificador {

void enviar(String mensagem);

}

public class SistemaAntigoAdapter implements Notificador {

private SistemaAntigoNotificacao antigo = new SistemaAntigoNotificacao();

public void enviar(String mensagem) {

antigo.enviarEmailAntigo(mensagem);

}

}

**Decorator – Recursos adicionais em consultas**

**Aplicação:**

Adicionado comportamento dinâmico em objetos Consulta, como anexar exames ou relatórios complementares.

**Justificativa:**

Permite extensão de funcionalidades sem alterar a classe base.

**Código:**

java

CopiarEditar

public interface Consulta {

String descrever();

}

public class ConsultaBase implements Consulta {

public String descrever() {

return "Consulta básica";

}

}

public class ExameDecorator implements Consulta {

private Consulta consulta;

public ExameDecorator(Consulta consulta) {

this.consulta = consulta;

}

public String descrever() {

return consulta.descrever() + " + exame adicional";

}

}

**3. Facade – Interface simplificada de agendamento**

**Aplicação:**

Criada a classe AgendamentoFacade que encapsula lógica de agendamento de consulta, verificação de disponibilidade e envio de notificação.

**Justificativa:**

Simplifica o uso do sistema para o programador, ocultando complexidade interna.

**Código:**

java

CopiarEditar

public class AgendamentoFacade {

private Agenda agenda = new Agenda();

private Notificador notificador = new SistemaAntigoAdapter();

public void agendarConsulta(String paciente, String data) {

agenda.marcar(paciente, data);

notificador.enviar("Consulta agendada para " + paciente);

}

}

**4. Composite – Plano de tratamento**

**Aplicação:**

Representação de um plano de tratamento que pode conter múltiplos procedimentos.

**Justificativa:**

Permite tratar componentes individuais e composições de maneira uniforme.

**Código:**

java

CopiarEditar

public interface Procedimento {

void executar();

}

public class Exame implements Procedimento {

public void executar() {

System.out.println("Executando exame...");

}

}

public class ConsultaEspecializada implements Procedimento {

public void executar() {

System.out.println("Consulta especializada...");

}

}

public class PlanoTratamento implements Procedimento {

private List<Procedimento> procedimentos = new ArrayList<>();

public void adicionar(Procedimento p) {

procedimentos.add(p);

}

public void executar() {

for (Procedimento p : procedimentos) {

p.executar();

}

}

}

**Conclusão**

Os padrões estruturais aplicados no projeto ClinicaGoF contribuíram para uma arquitetura mais flexível e desacoplada. Adapter permitiu integração com sistemas legados, Decorator trouxe dinamismo, Facade simplificou o uso de funcionalidades complexas e Composite viabilizou a composição de planos de tratamento mais ricos.

.