**03 - Padrões de Projetos Comportamentais – Respostas**

**Alunos: João César de Carvalho Júnior**

**Thieres Marcos Introdução**

Neste terceiro trabalho, foram implementados **padrões comportamentais**, que tratam da comunicação entre objetos e como responsabilidades são distribuídas dinamicamente.

**Padrões Comportamentais Utilizados**

**1. Observer – Notificação de eventos**

**Aplicação:**

Quando uma consulta é agendada, todos os observadores (paciente, médico, sistema) são notificados.

**Justificativa:**

Facilita o desacoplamento entre emissores e receptores de eventos.

**Código:**

java

CopiarEditar

public interface Observador {

void notificar(String mensagem);

}

public class PacienteNotificador implements Observador {

public void notificar(String msg) {

System.out.println("Paciente: " + msg);

}

}

public class ConsultaSubject {

private List<Observador> observadores = new ArrayList<>();

public void adicionarObservador(Observador o) {

observadores.add(o);

}

public void agendarConsulta(String dados) {

for (Observador o : observadores) {

o.notificar("Consulta marcada: " + dados);

}

}

}

**2. Strategy – Cálculo de pagamento**

**Aplicação:**

Diferentes estratégias de cálculo para pacientes com convênio, particulares, ou com desconto.

**Justificativa:**

Evita condicionais extensas e facilita extensão com novas regras.

**Código:**

java

CopiarEditar

public interface PagamentoStrategy {

double calcular(double valorBase);

}

public class PagamentoParticular implements PagamentoStrategy {

public double calcular(double valorBase) {

return valorBase;

}

}

public class PagamentoConvenio implements PagamentoStrategy {

public double calcular(double valorBase) {

return valorBase \* 0.5;

}

}

**3. Command – Encapsulamento de ações**

**Aplicação:**

Criado comando CancelarConsultaCommand para encapsular a lógica de cancelamento.

**Justificativa:**

Facilita o armazenamento, agendamento e desfazer ações.

**Código:**

java

CopiarEditar

public interface Comando {

void executar();

}

public class CancelarConsultaCommand implements Comando {

private Consulta consulta;

public CancelarConsultaCommand(Consulta consulta) {

this.consulta = consulta;

}

public void executar() {

consulta.cancelar();

}

}

**4. Template Method – Geração de relatórios**

**Aplicação:**

Classe abstrata define a estrutura básica, subclasses implementam partes específicas.

**Justificativa:**

Garante consistência com flexibilidade.

**Código:**

java

CopiarEditar

public abstract class RelatorioTemplate {

public final void gerarRelatorio() {

cabecalho();

conteudo();

rodape();

}

protected abstract void cabecalho();

protected abstract void conteudo();

protected abstract void rodape();

}

public class RelatorioConsultaMedica extends RelatorioTemplate {

protected void cabecalho() { System.out.println("Cabeçalho médico"); }

protected void conteudo() { System.out.println("Corpo do relatório"); }

protected void rodape() { System.out.println("Rodapé"); }

}

**Conclusão**

Com os padrões comportamentais, o sistema ClinicaGoF ganhou modularidade e flexibilidade no controle de fluxos. Observer e Strategy eliminaram acoplamentos, Command facilitou o controle de comandos e Template Method trouxe padronização com liberdade de implementação.