**01 - Padrões de Projetos Criacionais – Respostas**

**João César de Carvalho Júnior**

**Thieres Marcos**

**Introdução**

O projeto ClinicaGoF representa uma clínica médica fictícia desenvolvida em Java, com o objetivo de aplicar os padrões de projeto GoF. Neste primeiro trabalho, foi realizada a aplicação dos padrões **criacionais**, que são responsáveis por abstrair o processo de criação de objetos, promovendo maior flexibilidade, reutilização e independência do sistema em relação à forma como os objetos são instanciados.

**Padrões Criacionais Utilizados**

**1. Singleton – Classe de Configuração de Banco de Dados**

**Aplicação:**

Criamos uma classe DatabaseConnection para garantir que apenas uma instância de conexão com o banco de dados seja utilizada em todo o sistema.

**Justificativa:**

Evita a criação de múltiplas conexões, economizando recursos e mantendo um ponto centralizado de controle.

**Código:**

public class DatabaseConnection {

private static DatabaseConnection instance;

private Connection connection;

private DatabaseConnection() {

// Inicializa conexão

}

public static synchronized DatabaseConnection getInstance() {

if (instance == null) {

instance = new DatabaseConnection();

}

return instance;

}

public Connection getConnection() {

return connection;

}

}

**2. Factory Method – Criação de Pacientes**

**Aplicação:**

Foi implementada uma PacienteFactory para encapsular a lógica de criação de diferentes tipos de pacientes (ex: particular, convênio).

**Justificativa:**

Isola o processo de criação e facilita a extensão do sistema para outros tipos de paciente no futuro.

**Código:**

public abstract class Paciente {

protected String nome;

public abstract String getTipo();

}

public class PacienteParticular extends Paciente {

public String getTipo() {

return "Particular";

}

}

public class PacienteConvenio extends Paciente {

public String getTipo() {

return "Convênio";

}

}

public class PacienteFactory {

public static Paciente criarPaciente(String tipo) {

if (tipo.equals("Particular")) return new PacienteParticular();

else if (tipo.equals("Convenio")) return new PacienteConvenio();

throw new IllegalArgumentException("Tipo inválido");

}

}

**3. Builder – Criação de Relatórios de Consulta**

**Aplicação:**

Aplicado para construir relatórios médicos que exigem muitos parâmetros opcionais.

**Justificativa:**

Melhora a legibilidade e evita construtores com muitos parâmetros (telescoping constructors).

**Código:**

public class RelatorioConsulta {

private String paciente;

private String medico;

private String data;

private String descricao;

private RelatorioConsulta(Builder builder) {

this.paciente = builder.paciente;

this.medico = builder.medico;

this.data = builder.data;

this.descricao = builder.descricao;

}

public static class Builder {

private String paciente;

private String medico;

private String data;

private String descricao;

public Builder paciente(String paciente) { this.paciente = paciente; return this; }

public Builder medico(String medico) { this.medico = medico; return this; }

public Builder data(String data) { this.data = data; return this; }

public Builder descricao(String descricao) { this.descricao = descricao; return this; }

public RelatorioConsulta build() {

return new RelatorioConsulta(this);

}

}

}

**4. Prototype – Clonagem de Consulta para Simulação**

**Aplicação:**

Foi criado um método clone() na classe Consulta para duplicar dados da consulta, útil para simulações ou prévias.

**Justificativa:**

Evita recriação manual de objetos com muitos atributos, útil para cenários de testes.

**Código:**

public class Consulta implements Cloneable {

private String paciente;

private String medico;

private String data;

public Consulta clone() {

try {

return (Consulta) super.clone();

} catch (CloneNotSupportedException e) {

throw new AssertionError();

}

}

}

**Conclusão**

A aplicação dos padrões criacionais no projeto ClinicaGoF trouxe maior organização, flexibilidade e robustez para o sistema, reduzindo o acoplamento entre classes e centralizando a lógica de criação de objetos. Os padrões **Singleton**, **Factory Method**, **Builder** e **Prototype** demonstraram-se ideais para os contextos da clínica, permitindo escalabilidade e facilidade de manutenção.