



## Aplicação de Padrões de Projeto em um Sistema Hospitalar

### 1. Introdução

Neste documento, vamos falar sobre como usar quatro padrões de projeto em um sistema hospitalar. Esses padrões são: Singleton, Builder, Adapter e Strategy. Cada um desses padrões ajuda a resolver problemas diferentes, como gerenciar dados, criar relatórios, integrar sistemas e escolher tratamentos.

### 2. Padrões Criacionais

#### 2.1 Singleton

O Singleton é usado para garantir que só exista uma única instância de uma classe em todo o sistema. No hospital, usamos o Singleton para o Gerenciador de Registros Médicos. Isso garante que todas as partes do sistema usem o mesmo conjunto de informações dos pacientes.

```
meu_projeto_hospital > src > singleton.py > GerenciadorRegistrosMedicos
1 class GerenciadorRegistrosMedicos:
2     _instancia = None
3
4     def __new__(cls):
5         if cls._instancia is None:
6             cls._instancia = super(GerenciadorRegistrosMedicos, cls).__new__(cls)
7             cls._instancia._registros = {}
8             return cls._instancia
9
10    def adicionar_registro(self, id_paciente, dados_paciente):
11        self._instancia._registros[id_paciente] = dados_paciente
12
13    def obter_registro(self, id_paciente):
14        return self._instancia._registros.get(id_paciente)
15
```

#### 2.2 Builder

O Builder ajuda a criar objetos complexos de uma maneira organizada e passo a passo. No hospital, usamos o Builder para criar Relatórios Médicos. Assim, podemos adicionar informações diferentes, como nome do paciente, diagnóstico e prescrição, de maneira flexível.

```
meu_projeto_hospital > src > builder.py > ConstrutorRelatorioMedico
1 class ConstrutorRelatorioMedico:
2     def __init__(self):
3         self.relatorio = {}
4
5     def adicionar_informacoes_paciente(self, nome, idade):
6         self.relatorio['nome'] = nome
7         self.relatorio['idade'] = idade
8         return self
9
10    def adicionar_diagnostico(self, diagnostico):
11        self.relatorio['diagnostico'] = diagnostico
12        return self
13
14    def adicionar_prescricao(self, prescricao):
15        self.relatorio['prescricao'] = prescricao
16        return self
17
18    def construir(self):
19        return self.relatorio
20
```

## 3. Padrão Estrutural

### 3.1 Adapter

O Adapter ajuda a conectar sistemas que não são compatíveis. No hospital, usamos o Adapter para que o Sistema de Monitoramento Novo e o Antigo possam trabalhar juntos. Com o Adapter, ambos os sistemas podem se comunicar sem precisar mudar os sistemas antigos.

```
meu_projeto_hospital > src > adapter.py > SistemaMonitoramentoAntigo
1 class SistemaMonitoramentoAntigo:
2     def __init__(self, paciente):
3         self.paciente = paciente
4
5     def monitorar(self):
6         return f"Monitorando {self.paciente} usando o sistema antigo."
7
8 class SistemaMonitoramentoNovo:
9     def __init__(self, paciente):
10        self.paciente = paciente
11
12    def monitorar(self):
13        return f"Monitorando {self.paciente} usando o sistema novo."
14
15 class AdaptadorMonitoramento:
16     def __init__(self, sistema_monitoramento):
17         self.sistema_monitoramento = sistema_monitoramento
18
19     def monitorar(self):
20         return self.sistema_monitoramento.monitorar()
21
```

## 4. Padrão Comportamental

### 4.1 Strategy

O padrão Strategy permite definir diferentes formas de fazer algo e escolher a melhor quando necessário. No hospital, usamos o Strategy para Gerenciar Estratégias de Tratamento. Dependendo da condição do paciente, podemos escolher entre medicação, cirurgia ou fisioterapia sem mudar a lógica principal do sistema.

```
meu_projeto_hospital > src > strategy.py > EstrategiaFisioterapia
1 class EstrategiaTratamento:
2     def tratar(self, paciente):
3         pass
4
5 class EstrategiaMedicacao(EstrategiaTratamento):
6     def tratar(self, paciente):
7         return f"Tratando {paciente['nome']} com medicação."
8
9 class EstrategiaCirurgia(EstrategiaTratamento):
10    def tratar(self, paciente):
11        return f"Tratando {paciente['nome']} com cirurgia."
12
13 class EstrategiaFisioterapia(EstrategiaTratamento):
14     def tratar(self, paciente):
15         return f"Tratando {paciente['nome']} com fisioterapia."
16
17 class ContextoTratamento:
18     def __init__(self, estrategia):
19         self.estrategia = estrategia
20
21     def executar_tratamento(self, paciente):
22         return self.estrategia.tratar(paciente)
23
```