

## Padrões de Projeto Utilizados no Sistema Hospitalar

### 1. Singleton

O padrão Singleton foi usado na classe “Hospital” para garantir que haja apenas uma única instância do hospital no sistema. Isso é útil porque o hospital precisa centralizar o gerenciamento de pacientes, médicos e consultas. Com o Singleton, temos certeza de que sempre estamos trabalhando com a mesma instância do hospital, evitando inconsistências. Implementa-se com um construtor privado e um método estático que retorna a única instância da classe.

```
3 public class HospitalController { 1 usage  Sarah Camargo
4
5     private static Hospital hospitalInstance; 3 usages
6
7     private HospitalController() {} no usages  Sarah Camargo
8
9     public static Hospital getInstance() { 1 usage  Sarah Camargo
10         if (hospitalInstance == null) {
11             hospitalInstance = new Hospital();
12         }
13         return hospitalInstance;
14     }
15 }
```

### 2. Builder

O padrão Builder foi aplicado na classe “Paciente” para facilitar a criação de objetos que têm muitos atributos. Em vez de usar vários construtores, o Builder permite que você monte o objeto passo a passo de maneira mais clara. Isso torna o código mais legível e fácil de modificar, especialmente quando nem todos os atributos precisam ser definidos na criação do objeto.

Implementa-se como uma classe interna estática dentro da classe “Paciente”, permitindo a configuração de atributos de forma fluente e intuitiva.

```
Hospital.java  HospitalController.java  Paciente.java x
8  public class Paciente { 15 usages  Sarah Camargo
51
52      public static class PacienteBuilder { 5 usages  Sarah Camargo
53          private String nome; 2 usages
54          private int idade; 2 usages
55          private String historicoMedico; 2 usages
56
57          public PacienteBuilder setNome(String nome) { 2 usages  Sarah Camargo
58              this.nome = nome;
59              return this;
60          }
61
62          public PacienteBuilder setIdade(int idade) { 2 usages  Sarah Camargo
63              this.idade = idade;
64              return this;
65          }
66
67          public PacienteBuilder setHistoricoMedico(String historicoMedico) { 2 usages  Sarah Camargo
68              this.historicoMedico = historicoMedico;
69              return this;
70          }
71
72          public Paciente build() { return new Paciente(nome, idade, historicoMedico); }
73      }
74  }
75
76  }
77
```

### 3. Adapter (Estrutural)

O padrão Adapter foi usado na classe “MedicoAdapter” para permitir que a classe “Medico” seja compatível com outros sistemas que precisam de uma interface diferente. O Adapter age como um tradutor, permitindo que “Medico” seja usado onde uma interface específica é necessária, sem modificar o código original da classe.

```
public class MedicoAdapter { 3 usages  Sarah Camargo
    private final Medico medico; 3 usages
>
    public MedicoAdapter(Medico medico) { this.medico = medico; }

    public String getMedicoInfo() { 1 usage  Sarah Camargo
        return "Nome: " + medico.getNome() + ", Especialidade: " + medico.getEspecialidade();
    }
}
```

### 4. Observer (Comportamental)

O padrão Observer foi aplicado na classe “ConsultaManager” para gerenciar notificações quando uma consulta é agendada ou cancelada. Com esse padrão, diferentes partes do sistema podem reagir automaticamente a esses eventos, como enviar um email ou SMS ao paciente. O Observer facilita a adição de novos tipos de notificações no futuro, sem precisar alterar o código existente.