Documento Técnico - Expansão do ClinicaGoF com Padrões de Projeto

Introdução

Este documento detalha a expansão funcional do sistema ClinicaGoF, com foco na aplicação de padrões de projeto comportamentais para suportar novos cenários de atendimento, como consultas por videoconferência e atendimentos domiciliares. A seguir, descrevemos como cada padrão foi aplicado, os benefícios obtidos e exemplos de uso.

Padrões de Projeto Aplicados

1. Chain of Responsibility - Validação de Agendamento

Onde e como foi aplicado:

O padrão Chain of Responsibility foi utilizado para criar uma cadeia de validação para o agendamento de consultas. Cada etapa da validação (disponibilidade do médico, compatibilidade do tipo de consulta, status do paciente e documentação obrigatória) é encapsulada em uma classe separada que implementa a interface `IAgendaValidator`. Cada validador decide se a solicitação pode prosseguir para o próximo validador na cadeia ou se o fluxo deve ser interrompido.

Benefícios obtidos:

- **Desacoplamento:** O código que agenda a consulta não precisa conhecer todos os detalhes da validação. Ele apenas inicia a cadeia de validação.
- **Flexibilidade:** É fácil adicionar, remover ou reordenar validadores sem alterar o código cliente.
- **Reusabilidade:** Cada validador é uma classe independente e pode ser reutilizado em outros contextos.

Exemplo de uso:

```
// Em ConsultaService.cs
var disponibilidadeMedico = new DisponibilidadeMedicoValidator(_consultaRepo);
var compatibilidadeTipoConsulta = new CompatibilidadeTipoConsultaValidator();
var statusPaciente = new StatusPacienteValidator();
var documentacaoObrigatoria = new DocumentacaoObrigatoriaValidator();

disponibilidadeMedico.SetNext(compatibilidadeTipoConsulta);
compatibilidadeTipoConsulta.SetNext(statusPaciente);
statusPaciente.SetNext(documentacaoObrigatoria);

_agendaValidator = disponibilidadeMedico;

// ...

if (!_agendaValidator.Validate(consulta, paciente, medico))
{
    throw new InvalidOperationException("Falha na validação do agendamento.");
}
```

2. Template Method - Fluxo de Consulta

Onde e como foi aplicado:

O padrão Template Method foi usado para definir um fluxo padrão para uma consulta, com etapas como `PreConsulta()`, `ExecutarConsulta()` e `FinalizarConsulta()`. A classe abstrata `ConsultaTemplate` define a estrutura do algoritmo, enquanto as subclasses `ConsultaPresencial` e `ConsultaOnline` implementam as etapas específicas para cada modalidade de consulta.

Benefícios obtidos:

- **Reutilização de código:** O fluxo geral da consulta é definido em um único lugar, evitando duplicação de código.
- **Extensibilidade:** Novas modalidades de consulta podem ser adicionadas facilmente, criando novas subclasses de `ConsultaTemplate`.
- **Controle sobre o fluxo:** O padrão garante que as etapas da consulta sejam executadas na ordem correta.

Exemplo de uso:

```
// Em ConsultaService.cs
ConsultaTemplate consultaProcesso;
if (consulta.TipoConsulta == TipoConsulta.Presencial)
{
    consultaProcesso = new ConsultaPresencial(consulta, paciente, medico);
}
else if (consulta.TipoConsulta == TipoConsulta.Online)
{
    consultaProcesso = new ConsultaOnline(consulta, paciente, medico);
}
else
{
    throw new ArgumentException("Tipo de consulta inválido.");
}
consultaProcesso.ProcessarConsulta();
```

3. Mediator - Coordenação entre Módulos

Onde e como foi aplicado:

O padrão Mediator foi implementado para centralizar a comunicação entre os módulos de Agenda, Notificação, Prontuário e Faturamento. A classe `ClinicaMediator` atua como o mediador, recebendo notificações de eventos (como `ConsultaAgendada` e `ConsultaCancelada`) e coordenando as ações necessárias nos outros módulos. Isso reduz o acoplamento direto entre os componentes.

Benefícios obtidos:

- **Redução do acoplamento:** Os módulos não precisam se conhecer diretamente, apenas conhecem o mediador.
- **Centralização do controle:** A lógica de coordenação entre os módulos está centralizada no mediador, facilitando a manutenção e o entendimento do sistema.
- **Facilidade de extensão:** Novos módulos podem ser adicionados ao sistema sem a necessidade de modificar os módulos existentes.

Exemplo de uso:

```
// Em ConsultaService.cs
_mediator.Notificar("ConsultaAgendada", consulta);
```

4. State - Ciclo de Vida da Consulta

Onde e como foi aplicado:

O padrão State foi utilizado para gerenciar o ciclo de vida de uma consulta. Cada estado (Agendada, Confirmada, Em Andamento, Finalizada, Cancelada) é representado por uma classe que implementa a interface `IEstadoConsulta`. A classe `Consulta` mantém uma referência ao seu estado atual e delega as ações (como `Confirmar()` ou `Cancelar()`) para o objeto de estado correspondente, que por sua vez, é responsável por realizar a ação e a transição para o próximo estado.

Benefícios obtidos:

- **Organização do código:** O comportamento de cada estado é encapsulado em sua própria classe, tornando o código mais limpo e fácil de entender.
- **Facilidade de adicionar novos estados:** Novos estados podem ser adicionados sem modificar as classes de estado existentes ou a classe `Consulta`.
- **Controle de transições:** As transições entre os estados são controladas de forma explícita, evitando estados inválidos.

Exemplo de uso:

```
// Na classe Consulta.cs
public void Confirmar()
{
    _estadoAtual.Confirmar(this);
}

// Na classe EstadoAgendada.cs
public void Confirmar(Consulta consulta)
{
    Console.WriteLine("Confirmando consulta...");
    consulta.MudarEstado(new EstadoConfirmada());
}
```

5. Memento - Histórico do Prontuário

Onde e como foi aplicado:

O padrão Memento foi implementado para permitir o armazenamento e a restauração de versões anteriores das observações no prontuário de um paciente. A classe `Prontuario` (Originator) cria um `ProntuarioMemento` para salvar seu estado interno. A classe `ProntuarioCaretaker` é responsável por armazenar o histórico de

mementos, permitindo que o sistema restaure um estado anterior do prontuário quando necessário.

Benefícios obtidos:

- **Rastreabilidade:** O histórico de alterações no prontuário é mantido, permitindo auditoria e visualização de versões anteriores.
- **Reversibilidade:** É possível restaurar o prontuário para um estado anterior de forma simples e segura.
- **Encapsulamento:** O estado interno do prontuário não é exposto diretamente, mantendo o encapsulamento.

Exemplo de uso:

```
 var prontuario = new Prontuario("Primeira observação.");
var caretaker = new ProntuarioCaretaker();

caretaker.AdicionarMemento(prontuario.SalvarEstado());

prontuario.Observacoes = "Segunda observação.";
caretaker.AdicionarMemento(prontuario.SalvarEstado());

prontuario.RestaurarEstado(caretaker.GetMemento(0)); // Restaura para a
primeira observação
```

Conclusão

A aplicação dos padrões de projeto comportamentais permitiu expandir o sistema ClinicaGoF de forma robusta e flexível, atendendo aos novos requisitos de negócio. O código-fonte resultante é mais organizado, coeso e fácil de manter, demonstrando o poder dos padrões de projeto na construção de software de qualidade.