Alexandre Ioshio Matsuo RA 1840482013036

Paulo Henrique Araujo de Andrade RA 1840482223026

**Documentação do Projeto de Clínica Veterinária**

1. **INTRODUÇÃO**

Este documento apresenta a especificação e o desenvolvimento de um sistema simplificado para uma Clínica Veterinária, com foco na aplicação de boas práticas de engenharia de software e padrões de projeto. O objetivo é demonstrar a arquitetura do software, a segregação de responsabilidades e a flexibilidade para futuras manutenções e evoluções.

O sistema simula operações de agendamento e realização de consultas, diagnóstico e prescrição de medicamentos para animais de estimação. A implementação foi realizada em linguagem Java, adotando os padrões de projeto **Facade**, **Adapter** e **Data Access Object (DAO)**, visando modularidade, baixo acoplamento e alta coesão.

1. **CONTEXTO E JUSTIFICATIVA**

Segundo Gamma et al. (1995), o padrão Facade oferece uma interface simplificada para um subsistema de classes. O gerenciamento de uma clínica veterinária envolve diversas operações, desde o cadastro de pets e clientes até o agendamento, realização de consultas, diagnósticos e prescrições. A falta de uma estrutura de software bem definida pode levar a sistemas monolíticos, de difícil manutenção e evolução.

A aplicação dos padrões de projeto Facade, Adapter e DAO justifica-se pela necessidade de:

* **Simplificar a interface** de uso para o cliente da aplicação (Facade).
* **Adaptar interfaces incompatíveis** entre componentes (Adapter).
* **Abstrair a camada de persistência** de dados, permitindo a fácil substituição da tecnologia de armazenamento (DAO).
* **Promover a segregação de responsabilidades**, garantindo que cada componente do sistema tenha uma única e bem definida função.

1. **ARQUITETURA DO PROJETO**

A arquitetura do projeto segue uma abordagem modular, baseada na segregação de responsabilidades e na aplicação dos padrões de projeto descritos. A estrutura de pacotes (packages) reflete essa organização:

src/

└── main/

└── java/

└── br/

└── com/

└── veterinaria/

├── adapter/ (Padrão Adapter)

├── dao/ (Padrão DAO - Interfaces e Implementações)

├── facade/ (Padrão Facade)

├── model/ (Classes de Modelo/POJOs)

└── Main.java (Ponto de Entrada da Aplicação)

**3.1 Padrão Facade (br.com.veterinaria.facade)**

O pacote facade contém a classe ClinicaFacade, que atua como uma **interface unificada** para um conjunto de interfaces em um subsistema.

* **ClinicaFacade.java**:
  + **Responsabilidade:** Fornecer uma interface simplificada e de alto nível para as operações complexas da clínica (agendamento, atendimento).
  + **Benefícios:** Reduz o acoplamento entre o cliente (Main.java) e o subsistema interno da clínica (DAOs, classes de modelo, adaptadores), tornando o sistema mais fácil de usar e manter. Esconde a complexidade de gerenciar múltiplos objetos e interações.

**3.2 Padrão Adapter (br.com.veterinaria.adapter)**

O pacote adapter contém classes que permitem que interfaces incompatíveis trabalhem juntas.

* **DataAdapter.java**:
  + **Responsabilidade:** Adaptar o formato de dados de java.util.Date para uma representação em String formatada (dd/MM/yyyy HH:mm:ss), conforme a necessidade de exibição ou persistência.
  + **Benefícios:** Permite que componentes que esperam diferentes formatos de dados colaborem sem a necessidade de modificar seus códigos-fonte.

**3.3 Padrão Data Access Object (DAO) (br.com.veterinaria.dao)**

A segregação de responsabilidades, princípio fundamental da engenharia de software (PRESSMAN; MAXIM, 2021), foi aplicada através do padrão DAO. O pacote dao implementa o padrão, que abstrai e encapsula todo o acesso à fonte de dados e O uso de DAO’s permite a abstração da persistência, seguindo princípios de código limpo (MARTIN, 2009).

* **ConsultaDAO.java (Interface):**
  + **Responsabilidade:** Definir o contrato (métodos) para operações CRUD (Create, Read, Update, Delete) relacionadas à entidade Consulta.
  + **Benefícios:** Garante que qualquer implementação de persistência de Consulta siga um contrato bem definido, promovendo a consistência.
* **InMemoryConsultaDAO.java (Implementação):**
  + **Responsabilidade:** Fornecer uma implementação concreta da interface ConsultaDAO que persiste os dados em uma lista na memória do aplicativo.
  + **Benefícios:** Separa a lógica de persistência da lógica de negócio. Facilita a substituição futura por uma implementação baseada em banco de dados (ex: JDBCConsultaDAO, HibernateConsultaDAO) sem impactar a camada de fachada ou as classes de negócio.

**3.4 Classes de Modelo (br.com.veterinaria.model)**

O pacote model contém as classes que representam as entidades de negócio do domínio da Clínica Veterinária. São objetos POJO (Plain Old Java Objects) que transportam dados e contêm a lógica de negócio intrínseca a si mesmos (getters, setters, construtores).

* **Pet.java**: Representa um animal de estimação.
* **Cliente.java**: Representa o proprietário do pet.
* **Veterinario.java**: Representa o profissional veterinário.
* **Medicamento.java**: Representa um medicamento prescrito.
* **Consulta.java**: Representa um agendamento ou evento de consulta. Contém referências a Pet, Cliente e Veterinario, e um id para gerenciamento pelo DAO.
* **Diagnostico.java**: Representa o diagnóstico de uma consulta, associado a uma Consulta.
* **Prescricao.java**: Representa a prescrição de medicamentos, associada a uma Consulta e contendo uma lista de Medicamentos.

1. **IMPLEMENTAÇÃO**

A implementação das classes e a interconexão entre elas foram realizadas em Java. Abaixo, são destacados os pontos chave da implementação.

**4.1 Código-Fonte**

(Nesta seção, você inseriria os trechos de código relevantes. **Para uma documentação real, você não colocaria todo o código aqui, mas referenciaria-o em um anexo ou em um repositório de código como o GitHub.** Aqui, vou apenas listar a estrutura para você saber onde cada trecho se encaixa, como nos exemplos anteriores.)

* **Main.java**: Ponto de entrada que demonstra a interação com a ClinicaFacade.
* **br.com.veterinaria.facade.ClinicaFacade**: Classe central do padrão Facade.
* **br.com.veterinaria.dao.ConsultaDAO**: Interface DAO.
* **br.com.veterinaria.dao.InMemoryConsultaDAO**: Implementação DAO em memória.
* **br.com.veterinaria.adapter.DataAdapter**: Classe do padrão Adapter.
* **br.com.veterinaria.model.\***: Classes de modelo.

**4.2 Fluxo de Execução Simplificado**

1. A classe Main instancia ClinicaFacade.
2. Main invoca o método agendarConsulta da fachada, passando os objetos de modelo necessários.
3. A ClinicaFacade cria o objeto Consulta e o envia para consultaDAO.salvar().
4. A InMemoryConsultaDAO persiste a Consulta em uma lista interna e atribui um ID.
5. A ClinicaFacade utiliza DataAdapter.formatarData() para exibir a data da consulta de forma amigável.
6. Main simula a busca da consulta agendada e chama realizarAtendimento na fachada.
7. A ClinicaFacade cria instâncias de Diagnostico e Prescricao internamente, orquestrando a lógica do atendimento.
8. Os detalhes do atendimento são exibidos, encapsulando as operações complexas.
9. **CONCLUSÃO**

Este projeto demonstra a eficácia da aplicação de padrões de projeto (Facade, Adapter, DAO) e da segregação de responsabilidades no desenvolvimento de software. A estrutura resultante é modular, com baixo acoplamento entre os componentes e alta coesão em suas funcionalidades específicas.

A utilização do DAO, em particular, provê uma camada de abstração para a persistência de dados, facilitando futuras migrações para diferentes tecnologias de banco de dados sem a necessidade de reescrever a lógica de negócio principal. O Facade simplifica a interface para os consumidores do subsistema, enquanto o Adapter resolve incompatibilidades de interface de forma transparente.

Este modelo serve como base para o desenvolvimento de sistemas mais complexos, promovendo a manutenibilidade, escalabilidade e extensibilidade do código.

**REFERÊNCIAS**

GAMMA, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1995.

GAMMA, E. et al. *Padrões de Projeto*: Soluções Reutilizáveis de Software Orientado a Objetos. Tradução de Luis Paulo G. Souto. Porto Alegre: Bookman, 2000. (Ou o ano da edição da sua tradução).

FOWLER, M. *Refactoring*: Improving the Design of Existing Code. 2. ed. Boston: Addison-Wesley, 2019.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software*: Uma Abordagem Profissional. 9. ed. Porto Alegre: AMGH, 2021. (Verifique a edição que você tem acesso ou que é mais atual).

MARTIN, R. C. *Código Limpo*: Habilidades Práticas do Agile Software. Rio de Janeiro: Alta Books, 2009.