RELATÓRIO FINAL – SISTEMA DE GERENCIAMENTO VETERINÁRIO CLÍNICA PETCARE

P2 – Programação Orientada a Objetos

1. Introdução

Este relatório descreve o desenvolvimento do Sistema de Gerenciamento Veterinário, proposto como avaliação P2 da disciplina de Programação Orientada a Objetos. O objetivo do projeto foi criar uma aplicação completa em Java, por linha de comando, capaz de gerenciar proprietários, animais e seus eventos médicos, aplicando os principais conceitos da orientação a objetos:

- Encapsulamento
- Herança
- Polimorfismo
- Classes abstratas
- Composição entre objetos
- Coleções (ArrayList)
- Tratamento de erros (exceções)
- Modularização do código

Além de atender aos requisitos funcionais, o projeto também exigiu organização do código, clareza, responsabilidade das classes e documentação mínima.

2. Escopo Geral do Sistema

O sistema gerencia:

- Proprietários

- Animais
- Consultas
- Cirurgias
- Exames
- Vacinas
- Histórico médico completo de cada animal
- Relatórios diversos

Essas entidades se relacionam por composição:
um proprietário *possui* animais, e um animal *possui* vários eventos médicos.

3. O que foi feito no projeto

Todas as funcionalidades principais propostas na P2 foram implementadas. Entre elas:

3.1 Cadastro de Proprietários

Inclui:

- Nome
- CPF
- Armazenamento de animais

O sistema permite cadastrar quantos proprietários forem necessários.

3.2 Cadastro de Animais

Cada animal possui:

- Nome
- Espécie
- Idade
- Proprietário vinculado

- Listas individuais de:
    - Consultas
    - Cirurgias
    - Vacinas
    - Exames
    - Eventos médicos gerais

Foi implementada validação para impedir idades negativas.

3.3 Sistema de Eventos Médicos (núcleo do projeto)

A classe abstrata EventoMedico estabelece:

- data
- hora
- animal
- status (ativo, cancelado, remarcado)
- métodos abstratos e polimórficos de exibição

As subclasses implementadas foram:

Consulta

- Motivo da consulta
- Data e horário

Cirurgia

- Tipo da cirurgia
- Duração
- Agendamento específico

Exame

- Tipo de exame
- Data e horário

Vacina

- Nome da vacina
- Classificação especial

Todos os eventos são adicionados automaticamente ao histórico do animal.

3.4 Cancelamento de eventos

Todos os tipos de evento implementam:

```
public void cancelar() {
    this.status = "Cancelado";
}
```

3.5 Remarcação de eventos

Cada evento pode ser remarcado, mudando data e hora.

3.6 Relatórios implementados

O sistema possui um menu dedicado para relatórios, incluindo:

- Listar todas as consultas cadastradas
- Listar todas as cirurgias
- Listar todos os exames
- Listar todas as vacinas
- Listar animais de um proprietário
- Mostrar histórico médico completo de um animal

Os relatórios utilizam polimorfismo de toString() nas subclasses de EventoMedico.

4. Funcionalidades Não Implementadas ou Parcialmente Implementadas

O projeto atendeu 100% dos requisitos essenciais, mas alguns recursos extras poderiam ter sido adicionados e ficaram de fora por limite de tempo ou por não serem obrigatórios:

4.1 Persistência em arquivos (salvar o sistema)

O programa não grava dados em:

- Arquivos .txt
- Arquivos .json
- Banco de dados

Motivo:
não foi exigido na P2, e implementá-lo adicionaria complexidade extra ao código.

4.2 Prevenção de choques de horário

O sistema não verifica automaticamente se dois eventos foram marcados no mesmo horário para o mesmo animal.

Motivo:
Funcionalidade extra não obrigatória.

4.3 Interface gráfica

Nenhuma interface gráfica foi implementada (JavaFX ou Swing).

Motivo:
Avaliação exige terminal, e GUI aumentaria muito o tempo de desenvolvimento.

4.4 Sistema de login

O projeto não inclui autenticação de usuários.

Motivo:
Requisito não presente no enunciado da P2.

5. Dificuldades enfrentadas

O desenvolvimento apresentou desafios importantes:

5.1 Organização das classes

Criar uma hierarquia clara entre:

- EventoMedico (abstrata)
- Consulta
- Cirurgia
- Exame
- Vacina

foi um dos maiores desafios do projeto, principalmente para manter consistência e evitar duplicação de código.

5.2 Relacionamento entre Animal e Eventos

Como um animal contém *vários tipos de evento*, foi necessário lidar com:

- Diversas listas diferentes
- Histórico unificado
- Polimorfismo para exibição no relatório

5.3 Tratamento de erros

Foram adicionadas verificações, como idade negativa e propriedades inválidas.
Manter o programa robusto sem travar exigiu testes e ajustes.

5.4 Tamanho do menu da Main

O menu principal cresceu bastante e precisou ser organizado cuidadosamente para evitar confusão ao usuário.

6. O que o aluno aprendeu

Durante a implementação deste projeto, foi possível reforçar diversos conceitos:

Entendimento sólido de herança e polimorfismo

Criar subclasses com comportamentos diferentes usando sobrescrita foi essencial.

Melhor compreensão de classes abstratas

EventoMedico atuou como a espinha dorsal das entidades do sistema.

Modelagem orientada a objetos

Aprendi a dividir responsabilidade entre classes e evitar acoplamento excessivo.

Manipulação de listas e coleções

O projeto utiliza varias listas encadeadas (proprietário → animais → eventos).

Importância de validação e tratamento de exceções

Evitar idade negativa ou dados inválidos melhorou a estabilidade da aplicação.

Organização e modularização

O projeto mostrou a importância de manter cada classe fazendo apenas seu papel.

7. Bibliotecas utilizadas e justificativa

A única biblioteca externa utilizada foi:

java.util.ArrayList

Usada para armazenar:

- Eventos médicos
- Animais dentro do proprietário
- Consultas, exames, cirurgias, vacinas separadamente

Foi escolhida porque:

- É dinâmica (tamanho variável)
- Fácil de percorrer com loops
- Simples de adicionar e remover itens
- É parte da biblioteca padrão do Java

Nenhuma outra biblioteca externa foi necessária ou utilizada.

8. Referências consultadas

Links e conteúdos usados durante o desenvolvimento:

- Documentação oficial do Java: https://docs.oracle.com/javase/8/docs/api/

Tutorial de classes abstratas e interfaces:
https://www.w3schools.com/java/java_abstract.asp

Explicação de ArrayList:
https://www.geeksforgeeks.org/arraylist-in-java/

Polimorfismo em Java:
https://www.javatpoint.com/runtime-polymorphism-in-java

StackOverflow (busca de dúvidas gerais):
https://stackoverflow.com/

Guia de boas práticas de POO:
https://refactoring.guru/

9. Screenshot / Trecho de Classe Importante

A seguir, um trecho da classe EventoMedico, considerada uma das mais importantes do projeto por aplicar herança e polimorfismo:

```
public abstract class EventoMedico {

    protected String data;

    protected String hora;

    protected Animal animal;

    protected String status;


    public EventoMedico(String data, String hora, Animal animal) {

        this.data = data;

        this.hora = hora;

        this.animal = animal;

        this.status = "Ativo";

    }


    public void remarcar(String novaData, String novaHora) {
```

```
        this.data = novaData;
        this.hora = novaHora;
        this.status = "Remarcado";
    }

    public void cancelar() {
        this.status = "Cancelado";
    }

    @Override
    public abstract String toString();
}
```

Este trecho resume os conceitos centrais do projeto:

- Classe abstrata
- Herança
- Métodos concretos e abstratos
- Polimorfismo (toString sobrescrito nas subclasses)
- Encapsulamento de dados

10. Conclusão

O projeto atendeu a todos os requisitos da P2, implementando um sistema completo, funcional e modular de gerenciamento veterinário. Foram utilizados os principais conceitos de Programação Orientada a Objetos de forma prática e integrada.

O desenvolvimento foi desafiador, especialmente no design das classes e na integração entre os eventos médicos e os animais. No entanto, o aprendizado adquirido foi

significativo, especialmente na organização do código, modularização e boas práticas de POO.

O sistema está pronto para ser executado e avaliado.