Faculdade Insted

Tecnólogo em Análise e Desenvolvimento de Sistemas

Desenvolvimento Orientado a Objetos e Padrões de Projeto

ATIVIDADE DE PORTFÓLIO – CONECTAPET

Vitória Caetano Faria da Silva

Prof. Thiago Gonçalves de Almeida

Campo Grande – MS
27/10/2025

Sumário

0.1 Aplicação de Boas Práticas de Programação

0.1.1 Convenções de Escrita

No desenvolvimento do sistema *ConectaPet*, foram utilizadas as convenções de escrita recomendadas pela PEP 8, guia oficial de estilo do Python.

As classes foram nomeadas em `CamelCase` (ex.: `Tutor`, `Veterinario`, `Clinica`), e as funções e variáveis em `snake_case` (ex.: `salvar_linha`, `cadastrar_tutor`, `listar_veterinarios`).

Essa padronização melhora a legibilidade e a consistência do código, além de facilitar futuras manutenções.

0.1.2 Princípios de Desenvolvimento – SOLID

Durante o desenvolvimento do *ConectaPet*, foram aplicados os princípios SOLID, que visam tornar o código mais organizado, flexível e de fácil manutenção:

- **S – Single Responsibility Principle (Responsabilidade Única):**
  Cada classe possui apenas uma responsabilidade. Por exemplo, `Tutor` lida somente com dados de tutores, enquanto `Clinica` gerencia cadastros e persistência.
- **O – Open/Closed Principle (Aberto/Fechado):**
  O sistema permite a extensão das classes sem a necessidade de alterar o código original, como na herança entre `Pessoa`, `Tutor` e `Veterinario`.
- **L – Liskov Substitution Principle (Substituição de Liskov):**
  As subclasses `Tutor` e `Veterinario` podem substituir `Pessoa` sem alterar o comportamento do sistema.
- **I – Interface Segregation Principle (Segregação de Interface):**
  Cada classe possui apenas os métodos essenciais à sua função, evitando sobrecarga de responsabilidades.
- **D – Dependency Inversion Principle (Inversão de Dependência):**
  O sistema depende de abstrações — por exemplo, o módulo `storage.py` fornece funções genéricas de leitura e escrita que podem ser reutilizadas sem alterar o restante do código.

0.1.3 Arquitetura de Software – Padrão MVC

O projeto *ConectaPet* foi estruturado de acordo com o padrão de arquitetura MVC (*Model-View-Controller*), que separa o sistema em três camadas principais:

- **Model (Modelo):** Contém as classes que representam os dados e regras de negócio (`Pessoa`, `Tutor`, `Veterinario`, `Animal`, `Prescricao`, `Consulta`).

- **View (Visão):** Representada pelo menu textual no arquivo `main.py`, responsável pela interação com o usuário.
- **Controller (Controle):** Implementado em `controllers/clinica.py`, mediando as ações entre a camada de modelo e a interface do usuário.

Essa separação facilita a manutenção e evolução do código, além de seguir boas práticas de arquitetura amplamente utilizadas no mercado de software.

0.1.4 Estilo e Qualidade de Código

O código do *ConectaPet* segue uma estrutura modular clara e bem organizada, com:

- nomes descritivos para variáveis e métodos,
- comentários breves e objetivos,
- menus e funções com responsabilidades definidas,
- e tratamento de exceções no módulo `storage.py` através de blocos `try/except`, prevenindo falhas durante a leitura de arquivos inexistentes.

Essas práticas tornam o sistema mais estável e profissional.

0.2 Nome e Contexto do Sistema

Nome: *ConectaPet* – Sistema de Gestão de Clínica Veterinária

Descrição e Objetivo:

O *ConectaPet* é um sistema desenvolvido para gerenciar uma clínica veterinária, permitindo o cadastro de tutores, veterinários, animais e consultas, além do registro de prescrições médicas.

Seu objetivo é demonstrar a aplicação prática dos princípios da Programação Orientada a Objetos (POO) e boas práticas de desenvolvimento, simulando um sistema real de gestão.

0.3 Modelagem e Classes

A modelagem do *ConectaPet* aplica os principais conceitos da POO:

- **Herança:** As classes `Tutor` e `Veterinario` herdam atributos e métodos da classe `Pessoa`.
- **Agregação:** Um `Tutor` possui uma lista de `Animal`, representando a relação entre o dono e seus animais.
- **Composição:** A `Consulta` contém uma instância de `Prescricao`, que só existe dentro dela.

- **Encapsulamento:** Os dados são armazenados de forma controlada, com métodos específicos de acesso e escrita (`to_txt` e `from_txt`).

Principais classes do projeto:

- `Pessoa`: Classe base com dados comuns (nome, telefone, e-mail).
- `Tutor`: Herda de `Pessoa` e representa o dono dos animais.
- `Veterinario`: Herda de `Pessoa` e representa o profissional responsável pela consulta.
- `Animal`: Armazena as informações do animal (nome, espécie, raça, idade).
- `Prescricao`: Detalha medicamentos e observações da consulta.
- `Consulta`: Relaciona tutor, animal e veterinário, podendo conter uma prescrição.
- `Clinica`: Controla os cadastros e coordena as operações do sistema.

0.4 UML – Diagrama e Conceitos

A UML (*Unified Modeling Language*) é uma linguagem de modelagem usada para representar graficamente os elementos de um sistema.

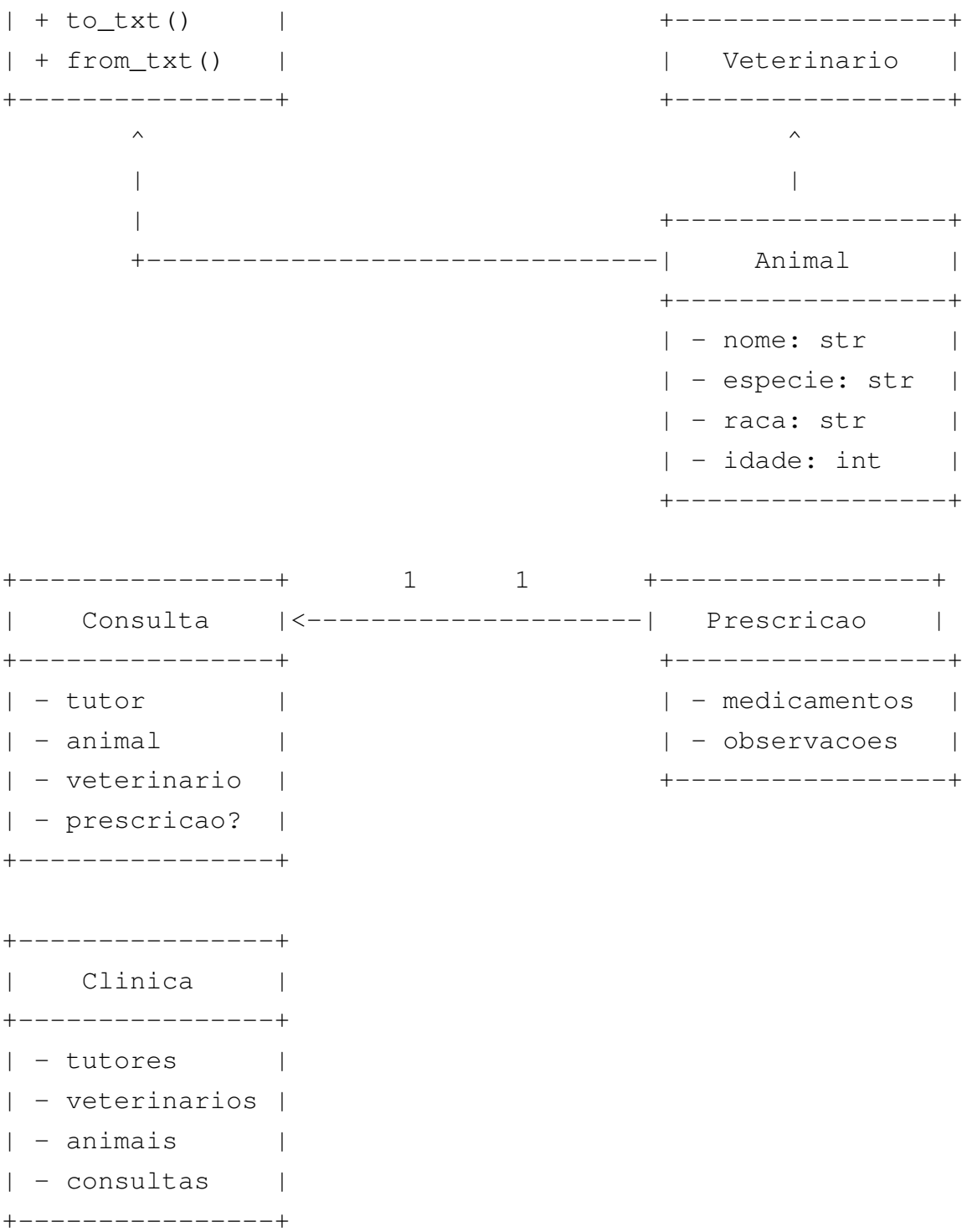
Os principais diagramas utilizados em projetos orientados a objetos são:

- **Diagrama de Casos de Uso:** mostra as interações entre o usuário e o sistema.
- **Diagrama de Classes:** exibe as classes, atributos e relacionamentos.
- **Diagrama de Sequência:** ilustra o fluxo de chamadas entre objetos.
- **Diagrama de Atividades:** representa o fluxo de execução de processos.

No *ConectaPet*, foi utilizado o **Diagrama de Classes** para representar as relações entre as entidades.

0.4.1 Diagrama de Classes (em texto – ASCII UML)

```
+----------------+       1      *       +-----------------+
|     Pessoa     |<---------------------|     Tutor       |
+----------------+                      +-----------------+
| - nome: str    |                      | - animais: list |
| - telefone: str|                      +-----------------+
| - email: str   |                              ^
+----------------+                              |
```



Este diagrama textual representa:

- `Pessoa` como superclasse de `Tutor` e `Veterinario` (herança).
- `Tutor` agregado a vários `Animal` (1 para *).
- `Consulta` composta por uma `Prescricao` (1 para 1, opcional).
- `Clinica` gerenciando todas as entidades.

0.5 Implementação e Persistência

O projeto foi organizado na seguinte estrutura de pastas:

```
ConectaPet/
          models/
                pessoa.py
                tutor.py
                veterinario.py
                animal.py
                prescricao.py
                consulta.py
          controllers/
                clinica.py
          persistence/
                storage.py
          dados/
                tutores.txt
                veterinarios.txt
                animais.txt
                consultas.txt
          main.py
          README.md
```

Listing 1 – Estrutura de pastas do projeto ConectaPet

Cada classe foi implementada em um arquivo próprio para garantir modularização e facilitar a manutenção.

O `main.py` é o ponto de entrada do sistema, exibindo um menu para o usuário cadastrar e listar tutores e veterinários.

Os dados são armazenados em arquivos `.txt` localizados na pasta `dados/`, utilizando as funções `salvar_linha()` e `ler_linhas()` do módulo `storage.py`.

Esse método de persistência garante que as informações cadastradas sejam salvas de forma simples e acessível, mantendo o funcionamento mesmo após o fechamento do programa.

0.6 Conclusão

O sistema *ConectaPet* demonstra de maneira prática a aplicação dos conceitos da Programação Orientada a Objetos, como herança, agregação, composição e encapsulamento, aliados a boas práticas de modularização e organização de código.

A adoção do padrão MVC, dos princípios SOLID e do uso de persistência em arquivos tornam o projeto um exemplo funcional e bem estruturado de aplicação POO em Python.

O *ConectaPet* cumpre o objetivo proposto de unir teoria e prática, apresentando um sistema funcional, escalável e bem documentado.