Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Licenciatura em Engenharia de Sistemas Informáticos

Gestão de Jardim Zoológico

Relatório de Projeto - Fase 2

Unidade Curricular: Programação Orientada a Objetos
Ano Letivo: 2025/2026

Autor:
Diogo Pereira - 31513

Barcelos
29 de dezembro de 2025

Resumo

O presente relatório descreve a segunda fase do trabalho prático da unidade curricular de Programação Orientada a Objetos, que visa o desenvolvimento de uma aplicação para a gestão de um Jardim Zoológico, denominada *GereZoo*.

Enquanto a primeira fase focou-se na modelação de classes e herança, esta segunda fase teve como objetivo principal a reestruturação da solução para uma arquitetura em camadas (N-Tier), visando uma clara separação de responsabilidades. Foram implementados mecanismos de persistência de dados através de serialização binária, garantindo a continuidade da informação entre execuções. Adicionalmente, aplicaram-se padrões de desenho (*Design Patterns*), nomeadamente *Singleton* e *Factory Method*, e utilizou-se LINQ para a manipulação eficiente de coleções.

Os resultados obtidos demonstram uma aplicação robusta, capaz de gerir animais, tarefas e bilheteira, com tratamento de exceções personalizado e validação de regras de negócio, cumprindo assim todos os requisitos propostos no enunciado.

Conteúdo

Lista de Figuras

1 Introdução

O desenvolvimento de software moderno exige não apenas a implementação de funcionalidades, mas também a adoção de arquiteturas que promovam a manutenção, a escalabilidade e a organização do código. O presente capítulo contextualiza o trabalho desenvolvido na Fase 2 do projeto.

1.1 Motivação e Enquadramento

No âmbito da Licenciatura em Engenharia de Sistemas Informáticos, a unidade curricular de Programação Orientada a Objetos propõe a criação de um sistema de gestão. A motivação para esta segunda fase reside na necessidade de transformar um protótipo inicial numa aplicação profissional, capaz de armazenar dados de forma persistente e estruturada segundo as boas práticas da indústria.

1.2 Objetivos

Os principais objetivos desta fase foram:

- Reestruturar o projeto numa arquitetura N-Tier (Várias Camadas).
- Implementar a persistência de dados (leitura e escrita em ficheiro).
- Aplicar padrões de desenho como *Singleton* e *Factory*.
- Utilizar LINQ e Expressões Lambda para consultas a dados.
- Desenvolver um mecanismo robusto de tratamento de exceções.

1.3 Estrutura do Documento

Este relatório encontra-se organizado em cinco capítulos. Após esta introdução, o capítulo 2 apresenta o enquadramento teórico. O capítulo 3 detalha o trabalho desenvolvido e a implementação. O capítulo 4 apresenta a análise dos resultados e, por fim, o capítulo 5 expõe as conclusões.

2 Enquadramento Teórico e Prático

Este capítulo apresenta os fundamentos teóricos essenciais que suportam as decisões de implementação tomadas durante o desenvolvimento do projeto.

2.1 Arquitetura N-Tier

A arquitetura em camadas, ou N-Tier, é um padrão de arquitetura de software que separa a aplicação em camadas lógicas e físicas. Tipicamente, esta separação inclui a Camada de Apresentação (UI), a Camada de Lógica de Negócio (BLL) e a Camada de Acesso a Dados (DAL) (**microsoft_arch**). Esta abordagem permite que a manutenção numa camada não afete necessariamente as outras, promovendo a modularidade.

2.2 Padrões de Desenho (Design Patterns)

Os padrões de desenho são soluções típicas para problemas comuns no design de software.

- **Singleton:** Garante que uma classe tenha apenas uma instância e fornece um ponto de acesso global a ela. É fundamental para a gestão centralizada de dados em memória (**gof_patterns**).
- **Factory Method:** Define uma interface para criar um objeto, mas deixa as subclasses decidirem que classe instanciar. Permite que o código seja independente das classes concretas que precisa criar.

2.3 Persistência e Serialização

A persistência refere-se à característica de um estado que sobrevive ao processo que o criou. Em C#, a serialização binária permite converter o estado de um objeto num fluxo de bytes para armazenamento em disco, permitindo a sua reconstrução posterior (**ms_docs_serialization**).

3 Trabalho Desenvolvido

Neste capítulo, detalha-se a arquitetura da solução implementada, descrevendo a organização dos projetos, as classes principais e a lógica de interação entre camadas.

3.1 Arquitetura da Solução

A solução *GereZoo* foi reestruturada seguindo o padrão N-Tier, dividida em cinco projetos distintos, conforme ilustrado na Figura 1. Esta separação garante que a interface de utilizador não acede diretamente aos dados, passando sempre pela camada de regras.

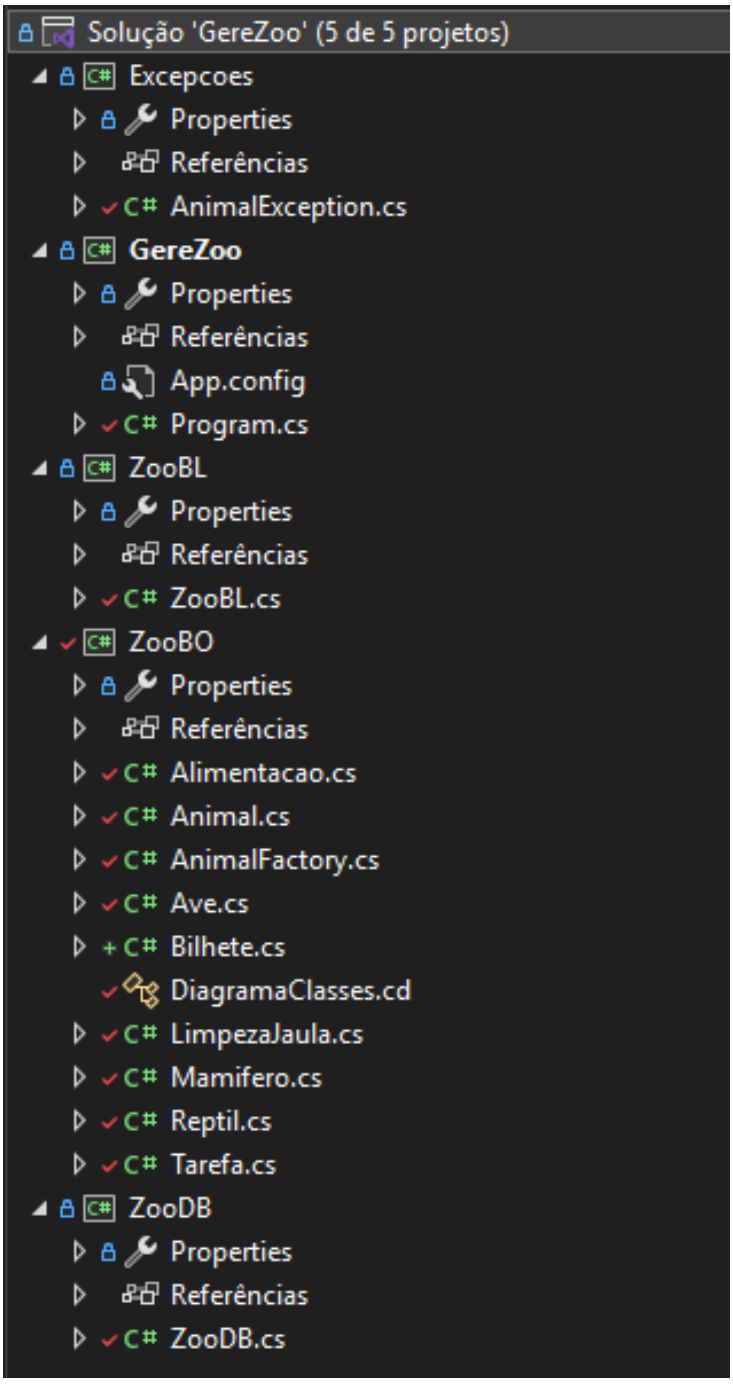

Figura 1: Estrutura da Solução no Visual Studio (5 Projetos)

As responsabilidades foram distribuídas da seguinte forma:

- **ZooBO (Business Objects):** Contém as classes de dados simples (*Animal*, *Mamifero*, *Ave*, *Reptil*, *Bilhete*, *Tarefas*). Não contém lógica complexa.
- **ZooDB (Data Access):** Implementa o padrão *Singleton*. É responsável por armazenar as listas em memória e gerir a gravação/leitura do ficheiro binário.
- **ZooBL (Business Logic):** Contém as regras de negócio. Valida, por exemplo, se um ID de animal já existe antes de permitir a inserção. Utiliza LINQ para estas verificações.
- **GereZoo (App):** A aplicação de consola que interage com o utilizador.
- **Excepcoes:** Biblioteca dedicada a erros personalizados, como `AnimalException`.

3.2 Implementação de Padrões

A implementação seguiu rigorosamente os padrões lecionados.

3.2.1 Padrão Factory

Para instanciar novos animais sem acoplar a aplicação às classes concretas, implementou-se a classe estática `AnimalFactory` no projeto *ZooBO*. Esta recebe uma *string* com o tipo de animal e devolve a instância correta (Polimorfismo).

3.2.2 Persistência de Dados

A persistência foi assegurada no projeto *ZooDB* através da classe `BinaryFormatter`. O método `GuardarDados` serializa a lista de animais para o ficheiro `dadosZoo.bin`, garantindo que a hierarquia de classes e os dados são preservados integralmente.

3.3 Modelação de Classes

A Figura 2 apresenta o diagrama de classes atualizado, evidenciando as relações de herança entre `Animal` e as suas especializações, bem como a estrutura de tarefas.

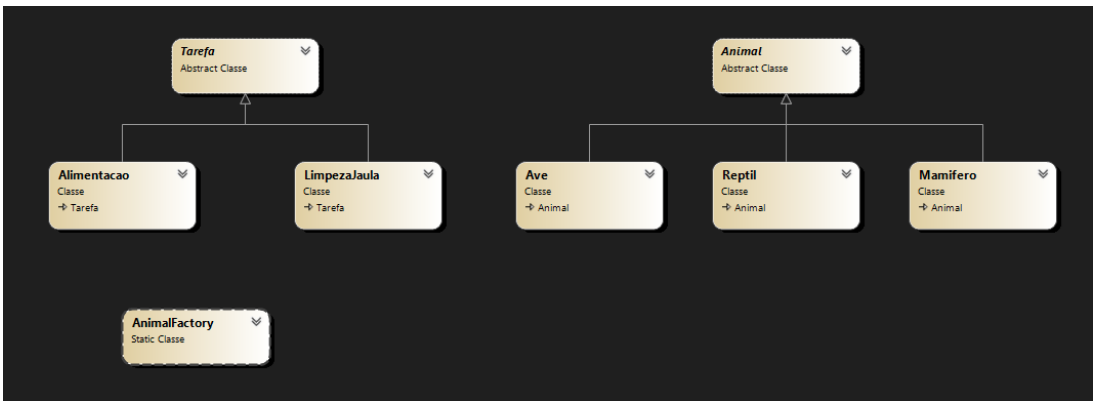

Figura 2: Diagrama de Classes da Fase 2

3.4 Testes Unitários

Seguindo as boas práticas de Engenharia de Software abordadas na Aula 17, foi criado um projeto adicional denominado `ZooTests`. Este projeto utiliza a framework *MSTest* para validar automaticamente as regras de negócio críticas. Criaram-se métodos de teste (`[TestMethod]`) para verificar:

- A inserção bem-sucedida de um animal válido.
- O lançamento da exceção `AnimalException` ao tentar inserir um animal com ID duplicado.

Esta abordagem garante a integridade do sistema antes da entrada em produção.

4 Análise e Discussão de Resultados

Este capítulo apresenta a análise dos resultados obtidos com a implementação da segunda fase do projeto, focando-se na validação da arquitetura e das funcionalidades críticas.

4.1 Validação Funcional da Arquitetura N-Tier

Os testes realizados durante o desenvolvimento demonstraram que a separação em camadas foi implementada com sucesso. A aplicação de consola (camada de apresentação) interage exclusivamente com a camada de lógica de negócio (`ZooBL`), que por sua vez valida os dados antes de recorrer à camada de dados (`ZooDB`).

Esta estrutura provou ser robusta. Por exemplo, ao tentar inserir um animal com um identificador já existente, a camada de regras deteta o conflito via LINQ e lança a exceção personalizada `AnimalException`, impedindo a corrupção dos dados na memória.

4.2 Persistência de Dados

Um dos principais requisitos desta fase era a persistência. Verificou-se que o mecanismo de serialização binária implementado no *Singleton* `ZooDB` funciona corretamente. Ao fechar a aplicação, os dados em memória (listas de animais) são serializados para o ficheiro `dadosZoo.bin`. Ao reiniciar a aplicação, estes dados são desserializados e carregados novamente, garantindo a continuidade do estado do sistema entre execuções.

4.3 Integração de Padrões

A utilização do padrão *Factory* na criação de animais simplificou o código na camada de apresentação, abstraindo a complexidade da instanciação das classes concretas (`Mamifero`, `Ave`, `Reptil`).

Adicionalmente, a implementação da classe `Bilhete` permitiu validar o uso de membros estáticos (`static`) para contadores globais (total vendido e total arrecadado), demonstrando um conceito distinto do padrão *Singleton* utilizado para a base de dados central.

4.4 Validação e Testes

A robustez da aplicação foi validada através de dois vetores:

1. **Testes Funcionais:** Execução da aplicação de consola, onde se verificou a persistência de dados (os animais mantêm-se após reiniciar o programa) e a correta listagem polimórfica das diferentes espécies.
2. **Testes Unitários:** Execução da bateria de testes automatizados do projeto `ZooTests`, que confirmaram que a camada `ZooBL` está a bloquear corretamente IDs duplicados e dados nulos, garantindo a qualidade do código.

5 Conclusão

O desenvolvimento da segunda fase do projeto *GereZoo* permitiu consolidar conhecimentos avançados de Programação Orientada a Objetos e Engenharia de Software. O presente capítulo sintetiza as principais conclusões retiradas deste trabalho.

A adoção da arquitetura N-Tier revelou-se fundamental para a organização do código. Embora tenha aumentado o número de ficheiros e projetos na solução, a separação clara entre dados, lógica e apresentação facilitou imenso a deteção de erros e a implementação de novas funcionalidades, como a classe *Reptil*, sem necessidade de alterar o código base da aplicação.

A implementação de padrões de desenho, especificamente o *Singleton* e a *Factory*, conferiu ao projeto um nível de profissionalismo superior, resolvendo problemas de gestão de memória e criação de objetos de forma elegante. A persistência de dados garantiu a utilidade real da aplicação.

Conclui-se que os objetivos propostos foram integralmente cumpridos. Como trabalho futuro, sugere-se a substituição da interface de consola por uma interface gráfica (Windows Forms ou WPF), o que seria facilitado pela arquitetura atual, visto que apenas seria necessário substituir a camada *GereZoo*.

6 Repositório GitHub

Todo o trabalho desenvolvido, incluindo o código-fonte da aplicação (`Zoologico.Core` e `Zoologico.App`) e os ficheiros-fonte deste relatório (*LaTeX*), está disponível publicamente num repositório GitHub para efeitos de controlo de versões e consulta.

O repositório pode ser acedido através da seguinte ligação:

`https://github.com/FrozenProduction/TP_POO_Fase2`