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 1

Unidade Curricular: Programação Orientada a Objetos
Docentes: Prof. Luís G. Ferreira

Autor:
Diogo Pereira
31513
a31513@alunos.ipca.pt

Barcelos, 12 de novembro de 2025

Resumo

O presente trabalho, desenvolvido no âmbito da Unidade Curricular de Programação Orientada a Objetos, foca-se na aplicação dos conceitos do paradigma orientado a objetos para resolver um problema de gestão de tarefas de um jardim zoológico.

Este relatório documenta a Fase 1 do projeto, que compreende a análise de requisitos, o desenho da arquitetura e a implementação essencial das classes. A solução foi estruturada numa biblioteca de classes (*DLL*) para promover a modularidade, e uma aplicação de consola para demonstração.

O desenho da solução assenta em hierarquias de classes abstratas (como `Animal` e `Tarefa`) que utilizam os pilares da abstração, herança e polimorfismo para gerir as diferentes entidades e operações do parque, como a alimentação, assistência veterinária e gestão de bilhetes. O resultado desta fase é um núcleo de *software* funcional que cumpre todos os requisitos essenciais definidos.

Conteúdo

1 Introdução

O presente capítulo expõe a contextualização do projeto. Inicia-se com a motivação e o enquadramento do trabalho, seguindo-se a definição dos objetivos traçados para esta primeira fase. Por fim, é detalhada a metodologia de trabalho adotada e a estrutura deste documento.

1.1 Motivação e Enquadramento

A gestão de um jardim zoológico moderno é uma operação logística complexa que envolve o cuidado de múltiplas espécies (animais), cada uma com necessidades distintas, e a coordenação de diversas tarefas diárias, desde a alimentação e limpeza até à assistência veterinária e realização de espetáculos. A ausência de um sistema de gestão centralizado e extensível pode levar a ineficiências e erros.

O paradigma da Programação Orientada a Objetos (POO) apresenta-se como a metodologia ideal para modelar este problema, permitindo criar representações digitais de entidades do mundo real (como `Animal` ou `Tarefa`) e gerir as suas interações.

Este projeto é desenvolvido no contexto da Unidade Curricular de Programação Orientada a Objetos, da Licenciatura em Engenharia de Sistemas Informáticos do Instituto Politécnico do Cávado e do Ave.

1.2 Objetivos

Para a conclusão da Fase 1 deste projeto, definiram-se os seguintes objetivos:

- Analisar os requisitos do problema de "Gestão de Jardim Zoológico".
- Desenhar uma arquitetura de *software* modular, separando a lógica de negócio (*Core*) da aplicação de teste (Consola).
- Identificar e modelar as classes e hierarquias essenciais através de um diagrama UML.
- Implementar a estrutura de classes base, aplicando corretamente os pilares da POO (Abstração, Encapsulamento, Herança e Polimorfismo).
- Utilizar estruturas de dados adequadas para a gestão das coleções de animais e tarefas.

1.3 Metodologia de Trabalho

A metodologia adotada foi o desenvolvimento orientado a objetos. O processo iniciou-se pela análise dos requisitos (palavras-chave do enunciado), seguindo-se a fase de desenho e modelação, onde as entidades e as suas relações foram definidas num diagrama de classes. Posteriormente, procedeu-se à implementação em C#, criando uma biblioteca de classes reutilizável e uma aplicação de consola para testes, garantindo que a "implementação essencial"estivesse funcional.

1.4 Estrutura do Documento

Este documento está organizado em cinco capítulos. O Capítulo 1 (o atual) introduz o projeto. O Capítulo 2 aborda o enquadramento teórico, focando-se nos pilares da POO. O Capítulo 3 detalha todo o trabalho desenvolvido, desde a arquitetura da solução até à implementação de cada pilar. Por fim, o Capítulo 4 apresenta a conclusão desta fase e aponta o trabalho futuro a desenvolver na Fase 2.

2 Enquadramento Teórico

Neste capítulo são revistos os fundamentos teóricos do Paradigma de Programação Orientada a Objetos (POO), que constituem a base metodológica sobre a qual este projeto foi desenvolvido. A correta aplicação destes conceitos é um dos objetivos centrais do trabalho.

2.1 Paradigma de Programação Orientada a Objetos

A POO é um paradigma de programação baseado no conceito de "objetos", que podem conter dados (atributos) e código (métodos). O objetivo é modelar o *software* de uma forma mais próxima da realidade, permitindo uma maior reutilização de código, flexibilidade e facilidade de manutenção. Este paradigma assenta em quatro pilares fundamentais, descritos de seguida.

2.2 Pilares da POO

2.2.1 Abstração

A abstração consiste em focar-se nos aspetos essenciais de uma entidade, ignorando os detalhes supérfluos. Em C#, isto é frequentemente implementado através de classes abstratas (*abstract class*) ou interfaces. Uma classe abstrata define um "contrato"do que uma classe deve ser, mas não pode ser instanciada diretamente, obrigando outras classes a herdar dela.

2.2.2 Encapsulamento

O encapsulamento é o mecanismo que restringe o acesso direto aos dados de um objeto, protegendo-os de modificações indevidas. Os dados (atributos) são definidos como privados (*private*) e o acesso a eles é controlado através de métodos públicos (*public*), como propriedades (*getters* e *setters*).

2.2.3 Herança

A herança permite que uma classe (subclasse ou classe filha) adquira os atributos e métodos de outra classe (superclasse ou classe pai). Isto promove a reutilização de código e a criação de hierarquias lógicas. A classe filha pode especializar a classe pai, adicionando novos métodos ou modificando (através de *override*) os existentes.

2.2.4 Polimorfismo

O polimorfismo (do grego, "muitas formas") é a capacidade de um objeto assumir diferentes formas. Em termos práticos, permite que uma variável de um tipo base (superclasse) possa referenciar um objeto de um tipo derivado (subclasse). Isto é fundamental para tratar coleções de objetos diferentes de forma homogénea, como uma lista de `Tarefa` que contém objetos `Alimentacao` e `LimpezaJaula`.

3 Trabalho Desenvolvido

Este capítulo detalha a solução desenhada e implementada para a Fase 1, justificando as decisões tomadas com base nos requisitos do enunciado e nos conceitos teóricos do capítulo anterior.

3.1 Análise e Especificação

A primeira etapa consistiu na análise do problema e no desenho da arquitetura geral da solução.

3.1.1 Requisitos

Com base no tema "Gestão de jardim zoológico", foram identificados os seguintes requisitos essenciais (palavras-chave) para a Fase 1:

- Gestão de **animais** e das suas **informações**.
- Gestão de um **calendário** de tarefas.
- Definição de tarefas específicas: **alimentação** (com **tipos de comida**), **assistência veterinária**, **limpeza de jaulas** e **espetáculos**.
- Gestão da venda de **bilhetes**.

3.1.2 Arquitetura da Solução

Para cumprir o requisito de "Reutilização de código através de bibliotecas (DLL)", a solução foi dividida em dois projetos:

- **Zoologico.Core**: Um projeto do tipo "Biblioteca de Classes"(.dll) que contém toda a lógica de negócio, incluindo as classes de animais, tarefas e a classe principal de gestão.
- **Zoologico.App**: Um projeto "Aplicação de Consola"(.exe) que serve como "Aplicação demonstradora". Este projeto adiciona uma referência ao `Zoologico.Core` para poder instanciar e testar as suas classes.

Esta separação garante um baixo acoplamento e uma clara separação de responsabilidades.

3.1.3 Modelação (Diagrama de Classes)

O desenho da solução foi modelado num diagrama de classes UML, conforme exigido. A Figura 1 ilustra as principais classes e as suas relações de herança e associação.

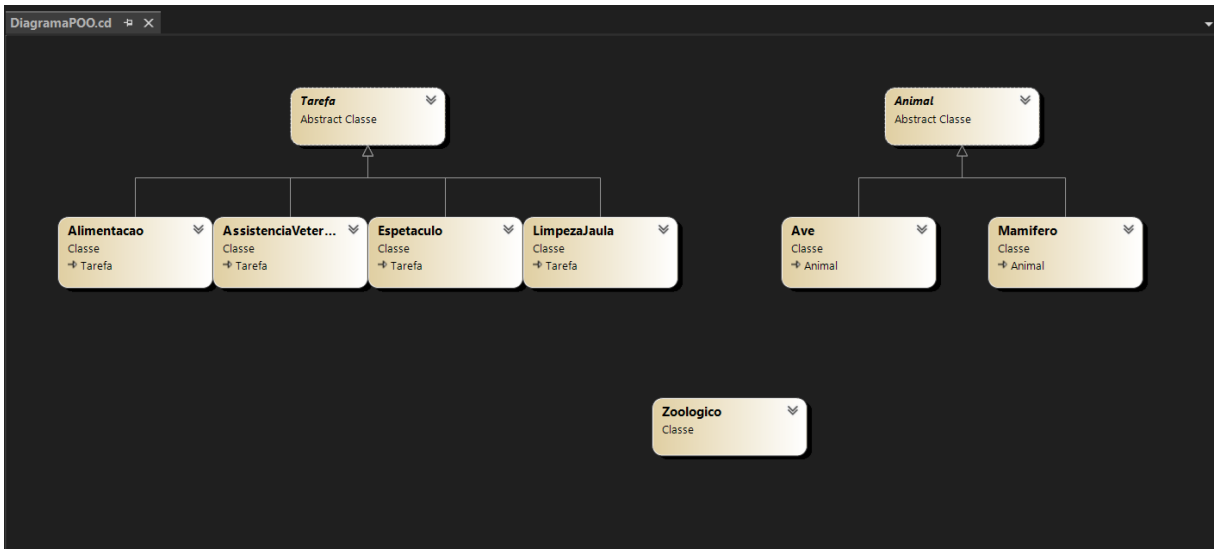

Figura 1: Diagrama de Classes UML da solução.
Fonte: Ficheiro `DiagramaPOO.cd` do projeto.

3.2 Implementação e Decisões de Desenho

A implementação focou-se na aplicação correta dos pilares da POO para garantir uma solução robusta e extensível.

3.2.1 Abstração (Animal e Tarefa)

Decidiu-se que nem `Animal` nem `Tarefa` deveriam ser instanciados diretamente. Definiu-se a classe `Animal` como *abstract*, pois um "animal"genérico não existe; o que existe são mamíferos, aves, etc. Esta classe define métodos abstratos como `GetInformacao()`, forçando as classes filhas a implementá-los.

O mesmo princípio foi aplicado à classe `Tarefa`. Ela define o que é comum a todas as tarefas (uma data e um estado de conclusão), mas a lógica de `Executar()` é específica de cada subtipo.

3.2.2 Herança (Animais e Tipos de Tarefa)

A herança foi usada para criar especializações. As classes `Mamifero` e `Ave` herdam de `Animal`, reutilizando as propriedades `Nome` e `Idade`, mas implementando de forma única o método `GetInformacao()`.

De igual modo, as classes `Alimentacao`, `LimpezaJaula`, `AssistenciaVeterinaria` e `Espetaculo` herdam de `Tarefa`, partilhando o construtor base e as suas propriedades.

3.2.3 Polimorfismo (Gestão de Tarefas)

O pilar do polimorfismo é central na gestão do zoológico. A classe `Zoologico` gere um `List<Tarefa>` (o "calendário"). No método `ExecutarTarefasPendentes()`, o sistema itera sobre esta lista e invoca o método `t.Executar()`. Graças ao polimorfismo, o sistema executa a versão correta do método (*override*) para cada objeto, seja ele `Alimentacao` ou `LimpezaJaula`, sem necessidade de verificar o seu tipo específico.

3.2.4 Encapsulamento (Proteção de Dados)

O encapsulamento foi aplicado para proteger o estado interno dos objetos. Na classe `Animal`, os atributos `nome` e `dataNascimento` são *private*. O acesso é controlado por propriedades *public*, como `Nome` (que permite leitura e escrita) e `Idade` (que permite apenas leitura e calcula o valor).

3.2.5 Métodos de Classe (Static)

Para o requisito "bilhetes", decidiu-se que o número total de bilhetes vendidos é uma informação global do zoológico, e não de uma instância particular. Para tal, usou-se um atributo *private static int totalBilhetesVendidos* na classe `Zoologico`. Os métodos `VenderBilhete()` e `GetTotalBilhetesVendidos()` foram implementados como *public static*, permitindo a sua invocação diretamente a partir da classe (`Zoo.VenderBilhete()`).

4 Conclusão

Neste capítulo, são apresentadas as conclusões retiradas do trabalho desenvolvido nesta primeira fase, bem como as linhas de orientação para o trabalho futuro a desenvolver na Fase 2.

4.1 Conclusões da Fase 1

A Fase 1 do projeto permitiu estabelecer as fundações de um sistema robusto para a gestão de um jardim zoológico. Todos os objetivos propostos para esta fase foram alcançados: a arquitetura de *software* (DLL + Aplicação de Consola) foi implementada com sucesso, cumpre os requisitos de modularidade e reutilização.

As hierarquias de classes desenhadas (visíveis na Figura 1) demonstraram ser eficazes na modelação do problema, e a aplicação dos pilares da POO (Abstração, Herança, Encapsulamento e Polimorfismo) permitiu criar um código limpo, extensível e de fácil manutenção, cobrindo todos os requisitos funcionais essenciais do enunciado.

4.2 Trabalho Futuro (Fase 2)

A Fase 2 incidirá em completar a solução, adicionando funcionalidades avançadas e garantindo a qualidade do *software*, conforme os critérios de avaliação. As principais tarefas a desenvolver serão:

- **Persistência de dados:** Implementar a capacidade de guardar e carregar o estado do zoológico (listas de animais e tarefas) em ficheiros.
- **Tratamento de Exceções:** Implementar exceções personalizadas (ex: `AnimalNaoEncontradoExc` para um tratamento de erros robusto.
- **Testes Unitários:** Desenvolver um projeto de testes unitários para validar a lógica de negócio, visando uma cobertura de código significativa.
- **Utilização de LINQ:** Refatorar o código de procura e gestão de listas para utilizar expressões LINQ, como pedido.
- **Aplicação Demonstradora:** Melhorar a aplicação de consola para se tornar uma interface de utilizador mais interativa, que demonstre todos os serviços implementados.

5 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_Fase1`