

**Instituto Politécnico do Cávado e do Ave**

**Escola Superior de Tecnologia**

**Licenciatura**

**de**

**Engenharia de Sistemas Informáticos**

**Trabalho Pratico de**

**Programação orientada aos Objetos**

Hugo Ferreira Baptista 23279

Nuno da Cunha Faria Gajo 23002

**Dezembro de 2023**

Índice

[2 Introdução 1](#_Toc153800930)

[3 Desenvolvimento 2](#_Toc153800931)

[3.1 Classes 3](#_Toc153800932)

[3.2 Estruturação do Código 4](#_Toc153800933)

[3.3 Exemplo de Objeto: 6](#_Toc153800934)

[3.4 Exemplo de Dados: 6](#_Toc153800935)

[3.5 Exemplo de uma regra de Negócio: 7](#_Toc153800936)

[3.6 Exemplo de uma Interface: 7](#_Toc153800937)

[3.7 Métodos usados nas classes 8](#_Toc153800938)

[3.8 Serialize 11](#_Toc153800939)

[3.9 Exceções 12](#_Toc153800940)

[4 Conclusão 13](#_Toc153800941)

# Introdução

No âmbito da unidade curricular de Programação Orientada aos Objetos (POO) fomos desafiados a criar uma solução prática em C# para resolver problemas do mundo real com uma complexidade moderada. Para atingir esse objetivo, nosso foco será identificar e criar classes relevantes, estabelecer estruturas de dados eficientes e implementar os processos essenciais que sustentarão essa solução. Em resumo, nossa tarefa consiste em desenvolver um programa robusto e funcional que aborde desafios reais, utilizando os princípios da programação orientada a objetos em C#.

O desafio que nos escolhemos foi a gestão de um jardim zoológico.

# Desenvolvimento

Iniciamos este desafio com um passo crucial: a identificação das classes essenciais e seus métodos correspondentes. Este processo é a base estrutural do nosso projeto em Programação Orientada a Objetos (POO), pois estabelece os alicerces sobre os quais construiremos uma solução sólida e eficiente para os problemas propostos.

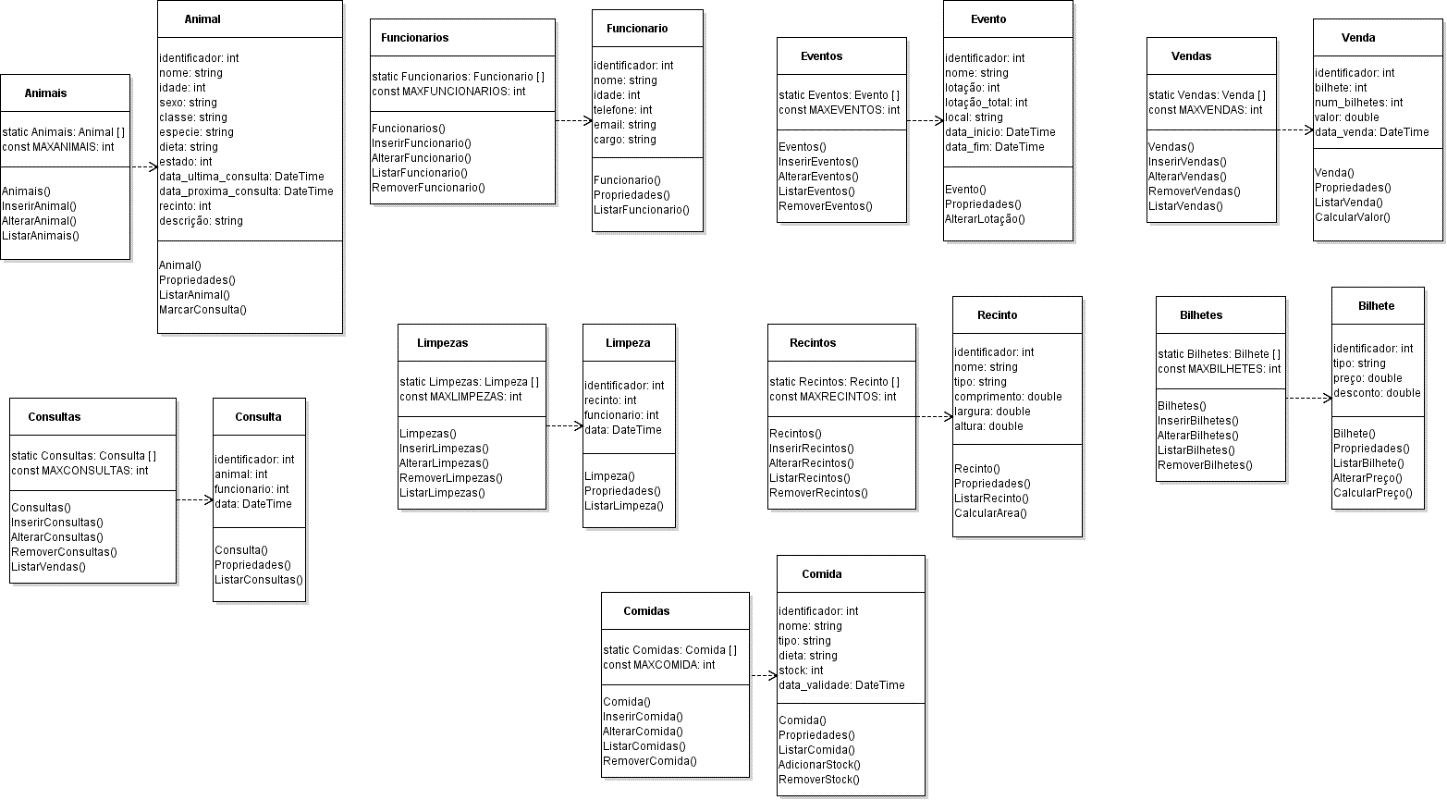
Primeiramente, dedicamo-nos à análise aprofundada dos requisitos do projeto, buscando compreender a natureza dos problemas apresentados. A partir dessa compreensão, identificamos entidades ou conceitos-chave que naturalmente se traduzem em classes. Cada classe é concebida para representar uma abstração do mundo real, encapsulando dados e comportamentos relacionados.

Com as classes identificadas, concentramo-nos na definição meticulosa de seus métodos. Estes são os procedimentos ou ações que cada classe pode executar, desempenhando um papel vital na implementação das funcionalidades desejadas. A escolha cuidadosa dos métodos é essencial para garantir que as classes interajam de maneira harmoniosa e eficaz.

No âmbito da linguagem de programação C#, garantimos que a implementação dos métodos seja consistente com os princípios da POO, priorizando a coesão interna de cada classe e a minimização de dependências externas. Isso não apenas facilita a manutenção do código, mas também promove a reutilização em futuras iterações do projeto.

Em resumo, a identificação criteriosa de classes e métodos marca o ponto de partida crucial para o desenvolvimento do nosso projeto em C#. Este é apenas o começo de uma jornada que promete traduzir conceitos abstratos em soluções tangíveis, guiadas pelos princípios sólidos da Programação Orientada a Objetos.

## Classes



As nossas classes são:

Animais: Esta classe é para gerir os animais do zoológico. Tem como atributos, informações que identifiquem o animal como o nome, a idade, o sexo, a espécie, a dieta, o estado, as datas das consultas, o recinto onde ele se encontra e uma descrição. Existem métodos para inserir, alterar, listar e marcar consultas para os animais.

Funcionários: É a classe para gerir os funcionários. Contém informações básicas como nome, idade, telefone, cargo, e email, além de métodos para inserir, alterar e listar os funcionários.

Eventos: Esta classe é usada para gerir eventos dentro do zoológico. Os atributos são as informações de um evento como o nome, a lotação e lotação máxima, o local onde é realizado e a data de início e fim do evento. Tem um método para inserir, alterar, listar e remover eventos.

Comidas: É uma classe para gerir o stock de comida, como atributos são as informações da comida como o nome, o tipo, a dieta a que pertence, o stock e a data de validade. Tem métodos para adicionar, alterar, listar e remover itens de comida, além de adicionar e remover stock. Isso é essencial em um ambiente onde muitos tipos diferentes de dietas de animais precisam ser geridos.

Recintos: Esta classe é para gerir os recintos dos animais, com atributos como nome, largura, comprimento e altura. Os métodos permitem inserir, alterar e listar recintos, além de calcular a área.

Bilhetes: A classe de bilhetes gere o tipo de bilhetes. Com métodos para inserir, alterar e listar bilhetes, além de alterar preço e calcular possíveis descontos.

Vendas: A classe de vendas gere as vendas de bilhetes com atributos de informações como o bilhete, o número de bilhetes e o valor da venda e com métodos para inserir, alterar, remover e listar vendas.

Limpezas: Esta classe é para gerir as limpezas aos recintos dos animais. Como atributos temos informações relativas à limpeza como o recinto onde foi realizada, o funcionário que realizou e a data de quando foi.

Consultas: A classes de consultas gere as consultas veterinárias dos animais, por isso como atributos são informações relativas à consulta como o funcionário que realizou a consulta, o paciente que é o animal e a data da consulta. Tem métodos

Cada classe tem uma lista estática de objetos, indicando que o sistema mantém um registo de todos os objetos criados.

## Estruturação do Código

Ao desenvolver o nosso projeto, adotamos uma abordagem estruturada que visava uma organização eficiente e uma manutenção simplificada do código. Essa estratégia consistiu na divisão clara e distinta entre objetos de negócio, regras de negócio e dados, proporcionando uma arquitetura robusta e coesa.

Primeiramente, identificamos os objetos de negócio relacionados ao domínio da aplicação. Estes objetos representam entidades específicas e suas interações dentro do contexto do sistema. Cada objeto de negócio foi encapsulado em uma classe, garantindo uma representação fiel e isolada de suas características e comportamentos. Essa abordagem orientada a objetos facilitou a compreensão do código, além de promover a reutilização de código e a manutenção modular.

Em seguida, dedicamos uma atenção especial às regras de negócio. Estas foram mapeadas de forma clara e concisa, separando-as das implementações técnicas. Organizamos as regras de negócio em classes específicas, onde cada classe era responsável por um conjunto de funcionalidades relacionadas. Isso não apenas simplificou a lógica de negócios, mas também permitiu a adaptação fácil a mudanças nos requisitos, sem comprometer a integridade do sistema.

A terceira camada essencial foi dedicada aos dados. Aqui, implementamos uma estrutura eficiente para o gerenciamento e manipulação dos dados utilizados pelo sistema. Utilizamos classes e estruturas de dados apropriadas para representar as informações de forma coesa e organizada. A separação clara entre objetos de negócio e dados assegurou a integridade dos dados e facilitou a manutenção do código, pois alterações na representação dos dados não afetavam diretamente a lógica de negócios.

Essa divisão cuidadosa em objetos de negócio, regras de negócio e dados não apenas melhorou a legibilidade e manutenção do código, mas também proporcionou uma base sólida para futuras expansões e otimizações. A clareza na estrutura do código não só beneficia o desenvolvedor que o criou, mas também qualquer pessoa que venha a interagir ou dar continuidade ao projeto no futuro.

Objetos de negócio: Dados: Regras de negócio:

A black screen with white text

Description automatically generatedUma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

## Exemplo de Objeto:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

## Exemplo de Dados:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

## Exemplo de uma regra de Negócio:

A screen shot of a computer

Description automatically generated

## Exemplo de uma Interface:

No decorrer do desenvolvimento do nosso projeto, buscamos elevar a flexibilidade e a extensibilidade do código, incorporando o conceito de interfaces de forma estratégica. A utilização de interfaces desempenhou um papel crucial na promoção da modularidade e na facilitação da integração de novos componentes.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

## Métodos usados nas classes

Contar:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Listar:

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Inserir:

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Alterar:

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Remover:

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Existe:

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Procurar:

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

## Serialize

Durante a elaboração do nosso projeto, percebemos a importância de otimizar o armazenamento e a troca de dados, tornando-os mais eficientes e compatíveis entre diferentes partes do sistema. Diante dessa necessidade, escolhemos integrar a serialização em várias classes, criando assim uma abordagem eficaz para a preservação de objetos e a facilitação da comunicação entre diversos componentes. Essa decisão revelou-se fundamental para garantir uma gestão de dados mais eficiente e uma interação mais fluida entre os elementos do projeto.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

## Exceções

Ao longo do desenvolvimento do nosso projeto, reconhecemos a importância de fortalecer a robustez e a confiabilidade do código, especialmente ao lidar com situações excecionais. Diante dessa perceção, implementamos um sistema abrangente de exceções, aprimorando significativamente a capacidade do programa de lidar com condições inesperadas e erros.

Exemplo:

A computer code on a black background

Description automatically generated

# Conclusão

No decorrer deste projeto de Programação Orientada a Objetos (POO), empenhamo-nos em criar uma solução que não apenas atendesse aos requisitos funcionais, mas que também incorporasse princípios fundamentais de design e boas práticas de desenvolvimento. A adoção de uma arquitetura baseada em objetos de negócio, regras de negócio, dados e interfaces revelou-se crucial para a construção de um sistema robusto e flexível.

A divisão clara em objetos de negócio proporcionou uma representação fiel das entidades do domínio, promovendo a encapsulação e reutilização de código. As regras de negócio foram tratadas de forma distinta, permitindo adaptações ágeis e independência das implementações técnicas. A gestão eficaz dos dados, separando-os das lógicas de negócio, assegurou a integridade e a manutenção facilitada do sistema.

Além disso, a integração de interfaces proporcionou um nível adicional de flexibilidade e extensibilidade ao projeto. A definição de contratos abstratos permitiu a substituição transparente de implementações, simplificando a manutenção e abrindo espaço para futuras melhorias.

Ao concluir este projeto, reconhecemos a importância da abordagem orientada a objetos na construção de sistemas mais coesos e adaptáveis. A experiência adquirida durante este processo de desenvolvimento fortaleceu a nossa compreensão sobre os benefícios de uma arquitetura bem estruturada e reforçou a necessidade de considerar não apenas os requisitos atuais, mas também a evolução futura do sistema.

Este projeto não é apenas o resultado de códigos e algoritmos, mas representa uma abordagem consciente e estratégica para a resolução de problemas complexos. Ao aplicar os princípios da POO, conseguimos criar um sistema que não apenas funciona, mas que é sustentável, modular e pronto para enfrentar os desafios que podem surgir ao longo do tempo.