IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

RELATÓRIO TRABALHO PRÁTICO – 1ª FASE

Diogo Alexandre Alves da Silva nº 31504

UC: Programação Orientada a Objetos

Professor: Luís Ferreira

Novembro, 2025

Indice

Índice de figuras

Glossário e Siglas ordenado por ordem alfabética

POO – Programação Orientada a Objetos

UC- Unidade Curricular

txt- Texto

Doxygen- Ferramenta que gera documentação automática do código

Resumo

Este trabalho foi desenvolvido no âmbito da *UC* de *POO*, com o objetivo de aplicar e consolidar conceitos de objetos, nomeadamente classes, atributos e métodos e os pilares de POO, como herança, polimorfismo, abstração e encapsulamento.

O problema proposto consiste na criação de um sistema que permite a gestão de um hospital.

O projeto produziu a criação de uma biblioteca em C#, que possui diferentes classes, com os seus atributos e métodos que permitem a implementação de uma gestão de um Centro de Saúde. Também para melhorar a compreensibilidade do projeto desenvolveu-se uma documentação no código.

Introdução

O presente capítulo pretende contextualizar o trabalho realizado ao apresentar a motivação para o seu desenvolvimento, o seu enquadramento e os principais objetivos.

Motivação

No contexto deste projeto foi proposto o desenvolvimento de uma biblioteca C# que permita gerir as operações básicas de um centro de saúde: marcação de consultas, registo de diagnósticos, prescrições, exames, internamentos e cuidados de enfermagem. A resolução deste problema permitiu aplicar, de forma prática e consolidada, os pilares da POO – encapsulamento, herança, polimorfismo e abstração – sem depender de estruturas de dados complexas, focando-se antes na modelação correta de classes, na relação entre elas e na organização modular do código, aspetos essenciais no curso.

Enquadramento

Este projeto foi desenvolvido no âmbito da unidade curricular de Programação Orientada a Objetos (POO), integrada no curso de Licenciatura em Engenharia Informática, sob orientação do Professor Luís Ferreira. Trata-se de um trabalho prático individual, inserido na 1.ª fase de avaliação da unidade curricular, com o objetivo de aplicar e consolidar os conhecimentos adquiridos sobre os princípios fundamentais da programação orientada a objetos.

O trabalho consistiu na implementação de uma biblioteca em C#, que modela um sistema de gestão de um centro de saúde, permitindo a realização de operações como marcação de consultas, registo de diagnósticos, exames, internamentos e cuidados de enfermagem. O desenvolvimento focou-se na aplicação dos pilares da POO, encapsulamento, herança, polimorfismo e abstração, bem como na organização modular do código e na documentação técnica.

Estado da Arte

O presente capítulo pretende contextualizar as ferramentas e estruturas utilizadas ao longo do projeto para realizar o trabalho proposto.

Conceitos / Fundamentos Teóricos

C#

"C# é uma linguagem de programação, multiparadigma, de tipagem forte, desenvolvida pela Microsoft como parte da plataforma .NET. A sua sintaxe orientada a objetos foi baseada no C++ mas inclui muitas influências de outras linguagens de programação, como Object Pascal e, principalmente, Java. O código fonte é compilado para Common Intermediate Language (CIL) que é interpretado pela máquina virtual Common Language Runtime (CLR))."(Referência: C# : Wikipédia, 15-11-2025)

manutenção em diversas linguagens de programação e projetos escalas." (Referência traduzida com Google Tradutor:Abstração :Rocketseat, 2025)

Programação Orientada a Objetos

"Programação orientada a objetos (POO, ou OOP segundo as suas siglas em inglês) é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos." (Referência: Programação orientada a objetos: Wikipédia 2025)

Classes

"uma classe é um Tipo abstrato de Dados (TAD); ou seja, uma descrição que abstrai um conjunto de objetos com características similares (um projeto do objeto), é um código da linguagem de programação orientada a objetos que define e implementa um novo tipo de objeto, que terão características (atributos) que guardaram valores e, também funções específicas para manipular estes." (Referência: Classe: Wikipédia, 2025)

Abstração

"A abstração é um princípio fundamental da Programação Orientada a Objetos que envolve a identificação e a modelagem das características e comportamentos essenciais de um objeto, ignorando os detalhes irrelevantes ou secundários para o contexto em questão. Em essência, a abstração permite aos desenvolvedores criar modelos simplificados de entidades complexas do mundo real, focando apenas nos aspectos que são importantes para a aplicação sendo desenvolvida." (Referência: Abstração :Rocketseat, 2025)

Problema

O problema apresentado foi o seguinte: Criar um sistema que faça a gestão de um hospital (Mas na primeira fase, foram feitas as classes, os fundamentos do resto)

Abordagem

A abordagem escolhida para a resolução do problema foi a implementação de classes bem compostas como se verá posteriormente. Essa escolha deve-se à sua eficiência e simplicidade frente às exigências do projeto nas funções. Caso as classes estejam bem definidas o resto do projeto será mais fácil pois a base estará bem preparada para a carga que for colocada em cima.

Implementação

Para iniciar este projeto projetou-se as classes necessárias para o sistema anteriormente referido.

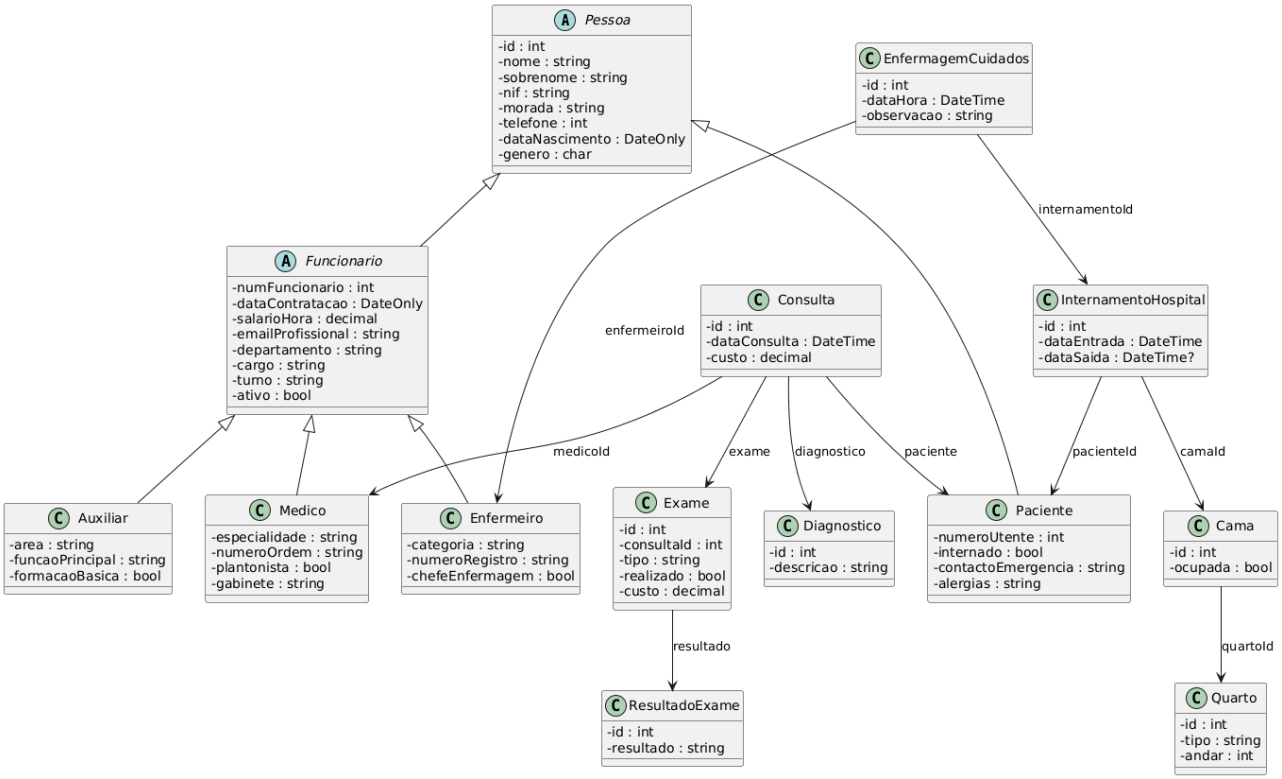

Figura 1 - Diagrama de Classes do projeto[1]

Como se pode observar na Figura 1 - Diagrama de Classes do projeto, todas as classes encontram-se conectadas para poderem realizar o seu propósito de servir como meio para alguém gerir um hospital para isso procede-se as suas funcionalidades e conexões.

Desenvolvimento

O presente capítulo pretende mostrar as soluções e o desenvolvimento ao longo do projeto.

Para iniciar, este projeto teve de decidir como iria definir as regras/instruções de utilização do programa sendo elas:

- O hospital apenas atende clientes com ficha (ou em caso de urgência realizar a ficha depois)
- Todas as relações são únicas pois ainda não foram abordadas estruturas de dados
- Como o projeto ainda se encontra na primeira fase encontra-se problemas de implementação pelo facto de não ser capaz de se inserir estruturas de dados, então desenvolveu-se apenas a estrutura básica
- O projeto ficou com meia implementação de devolução de códigos de erros que será mais aprofundada na segunda fase

Classes

Pessoas.cs

Pessoa

Resumo: Classe abstrata base para todas as pessoas do hospital.

Atributos: id, nome, sobrenome, nif, morada, telefone, dataNascimento, genero.

Métodos principais: NomeCompleto, calculaIdade, maiorIdade, ValidarNIF, GeneroExtenso.

Overrides: ToString(), GetTipo() (abstrato).

Operadores: ==, !=.

Funcionario

Resumo: Classe abstrata base para funcionários.

Atributos: numFuncionario, dataContratacao, salarioHora, emailProfissional, departamento, cargo, turno, ativo.

Métodos principais: ObterAnosServico, AumentarSalario.

Overrides: ToString(), GetTipo().

Operadores: ==, !=.

Paciente

Resumo: Representa um paciente.

Atributos: numeroUtente, internado, contactoEmergencia, alergias.

Métodos principais: AdicionarAlergia, RemoverAlergia, ObterUltimaConsulta, AptoAAlta, DarAlta.

Overrides: GetTipo().

Medico

Resumo: Representa um médico do hospital.

Atributos: especialidade, numeroOrdem, fazUrgencias, gabinete.

Métodos principais: EstaDisponivel, ObterPacientesDoDia, AdicionarEspecialidade, ConsultasMedico, AdicionarCuidadosEnfermeiro, RemoverCuidadosEnfermeiro.

Overrides: ToString(), GetTipo().

Enfermeiro

Resumo: Representa um enfermeiro.

Atributos: categoria, chefeEnfermagem.

Métodos principais: PodeSerChefe, CuidadosEnfermeiro, AdicionarCuidadosEnfermeiro, RemoverCuidadosEnfermeiro.

Overrides: ToString(), GetTipo().

Auxiliar

Resumo: Representa funcionário auxiliar (limpeza, TI, etc.).

Atributos: area, funcaoPrincipal.

Métodos principais: —

Overrides: ToString(), GetTipo().

Consultas.cs

Consulta

Resumo: Representa uma consulta médica, ligando paciente, médico, exames, diagnósticos e custo associado.

Atributos: id, paciente, medicoId, dataConsulta, exame, diagnostico, custo.

Métodos principais: AdicionarExame, RemoverExame, ListarExames, AdicionarDiagnostico, RemoverDiagnostico, ListarDiagnostico, CalcularCustoTotal.

Overrides: ToString().

Operadores: ==, !=.

Diagnostico

Resumo: Guarda um diagnóstico associado a uma consulta.

Atributos: id, descricao.

Métodos principais: —

Overrides: ToString().

Exame

Resumo: Representa um exame solicitado numa consulta.

Atributos: id, consultaId, tipo, resultado, realizado, custo.

Métodos principais: —

Overrides: ToString().

Operadores: ==, !=.

ResultadoExame

Resumo: Representa o resultado de um exame.

Atributos: id, resultado.

Métodos principais: —

Overrides: ToString().

InternamentoHospital.cs

Quarto

Resumo: Representa um quarto do hospital.

Atributos: id, tipo, andar.

Métodos principais: AdicionarCama, RemoverCama.

Overrides: ToString().

Operadores: ==, !=.

Cama

Resumo: Representa uma cama num quarto.

Atributos: id, quartoId, ocupada.

Métodos principais: —

Overrides: ToString().

Operadores: ==, !=.

InternamentoHospital

Resumo: Representa um internamento associando paciente, cama e datas.

Atributos: id, pacienteId, camaId, dataEntrada, dataSaida.

Métodos principais: —

Overrides: ToString().

Operadores: ==, !=.

EnfermagemCuidados

Resumo: Guarda observações feitas por um enfermeiro a um paciente internado.

Atributos: id, internamentoId, enfermeiroId, dataHora, observacao.

Métodos principais: —

Overrides: ToString().

Operadores: ==, !=.

Funcionalidade Geral

Para melhor compreensão da necessidade de cada classe, observa-se na Figura 1 - Diagrama de Classes do projeto[1] as suas ligações.

A classe *Pessoa* é a classe principal mas uma classe abstrata onde as que herdam de si eram reutilizar os seus atributos, em que as classes que herdam da mesma é a classe *Paciente*, que serve para guardar e gerir os pacientes do hospital. Outra classe que herda de *Pessoa* é *Funcionario*, outra classe abstrata que serve para ser herdada por: *Auxiliar, Medico e Enfermeiro*. A classe *Auxiliar* serve para guardar auxiliares (Ex: Limpeza, Informatica etc), a classe Médico conecta-se a *Consulta* (uma consulta pode ter um médico mas um médico pode ter várias consultas) e a classe Enfermeiro conecta-se a *EnfermagemCuidados*, que serve para fazer observações de serviços prestados a pessoas internadas no hospital. Uma consulta resulta em vários diagnósticos(classe *Diagnostico*) e vários exames e cada *Exame* possui um resultado. Cada *Internamento* possui um paciente e uma *Cama* que pertence a um *Quarto*.

Repositório GitHub:

Repositório GitHub

Conclusão

Este projeto representou uma aplicação prática e integrada dos conceitos fundamentais da Programação Orientada a Objetos, demonstrando como princípios como encapsulamento, herança, polimorfismo e abstração podem ser traduzidos em soluções reais e funcionais. A modelação de um sistema de gestão hospitalar exigiu não só a definição rigorosa de classes e relações, mas também uma reflexão constante sobre a eficiência, modularidade e escalabilidade do código.

A escolha do C# pela organização do código como práticas profissionais essenciais. Ao longo do desenvolvimento, consolidaram-se competências técnicas, mas também se reforçou a capacidade de resolver problemas complexos com soluções simples, limpas e bem estruturadas.

Em suma, este trabalho não só cumpriu os objetivos da unidade curricular para a primeira fase, como também serviu como um ponto de partida sólido para o desenvolvimento de futuros projetos mais avançados, onde a qualidade do código e a correta aplicação dos paradigmas de programação serão ainda mais cruciais.

Referências

Imagem do diagrama (Figura 1 - Diagrama de Classes do projeto[1]) gerada através do site : PlantUml

C# : Wikipédia, 15-11-2025

Abstração :Rocketseat, 2025

Classe: Wikipédia, 2025

Programação orientada a objetos: Wikipédia 2025