

Relatório de trabalho prático

Relatório do Trabalho Pratico LP2

Luís Martins

Aluno nº 16980

Daniel maltez

Aluno nº 17015

Trabalho realizado sob a orientação de:

Luís Ferreira

**Linguagens de Programação II**

**Licenciatura em Engenharia de Sistemas Informáticos**

Barcelos, Abril de 2020

Índice

[1 Introdução 1](#_Toc44092244)

[1.1 Contextualização 1](#_Toc44092245)

[1.2 Objetivos 1](#_Toc44092246)

[1.3 Estrutura 2](#_Toc44092247)

[2 Desenvolvimento 3](#_Toc44092248)

[2.1 Diagrama de Classes e Classes 3](#_Toc44092249)

[3 Conclusão 13](#_Toc44092250)

[Bibliografia 15](#_Toc44092251)

Lista de Figuras

Figura 1: Diagrama de Classes

Figura 2: Propriedades da classe Pessoa

Figura 3: Enumerador Sexo de uma pessoa

Figura 4: Propriedades da classe Paciente

Figura 5: Enumerador que determina condição de paciente

Figura 6: Propriedades da classe Paciente

Figura 7: Diagrama de Classes da camada Data Layer

Figura 8: Inicialização de listas para armazenar pacientes

Figura 9: Inicialização da lista para armazenar médicos

Figura 10: Diagrama de Classes da camada Business Rule Layer

# Introdução

## Contextualização

Neste trabalho visamos desenvolver uma aplicação que consiga gerir a secção das urgências de um hospital detetando os níveis de urgência na condição de um paciente e atribuindo certos casos a certos médicos, como também guardar e organizar os dados de pacientes e médicos em certos ficheiros.

Para a execução deste projeto utilizamos a linguagem de programação C#, e as funcionalidades dadas pelo mesmo.

## Objetivos

No início do projeto foi-nos apresentado certos objetivos a abranger ao longo do semestre para incentivar uma maior exploração da linguagem C# e da sua diferente aplicabilidade em senários diferentes, sendo estes:

* Consolidar conceitos basilares do Paradigma Orientado a Objetos (POO);
* Desenvolver capacidades de programação em C#;
* Potenciar a experiência no desenvolvimento de software;
* Assimilar o conteúdo da Unidade Curricular;

## Estrutura

Para este projeto foi utilizada a arquitetura “NTier”, ou seja, o código foi implementado de forma a ser organizado em 4 camadas diferentes:

1. **Business Objects –** Esta camada contém as definições de todos os objetos utilizados no projeto (Pessoa, Paciente, etc);
2. **Data Layer –** Esta cama é definida por conter as bibliotecas que contem as estruturas de dados utilizadas no projeto como também os seus métodos;
3. **Business Rule Layer –** Esta é a camada que separa as camadas discutidas anteriormente da camada de apresentação, como também a camada que gere os comandos a serem usados;
4. **Presentation Layer –** Finalmente esta camada é a que vai apresentar e traduzir as tarefas e os resultados para algo que o utilizador consiga compreender;

# Desenvolvimento

## Diagrama de Classes e Classes

Uma imagem com captura de ecrã

Descrição gerada automaticamente

*Figura 1 – Diagrama de Classes da camada Business Objects*

Esta camada é composta por 3 classes, Pessoa, Paciente e Medico com as duas últimas sendo classes filho da classe Pessoa pois derivam certos dados da mesma.

**Pessoa:**

É constituída pelos atributos:

* Idade
* Nif
* Nome
* Sexo

E contem propriedades que permitem manipular os próprios atributos:

Uma imagem com captura de ecrã, monitor, preto, sentado

Descrição gerada automaticamente

*Figura 2 – Propriedades da classe Pessoa*

Para determinar o sexo de uma pessoa usamos um enumerador para diferenciar entre os estados:

Uma imagem com telefone, telemóvel, preto, rua

Descrição gerada automaticamente

*Figura 3 – Enumerador Sexo de uma pessoa*

**Paciente:**

É constituída pelos atributos:

* Altura
* Condição
* Data de Entrada
* Data de Saida
* Peso

E contem propriedades que permitem manipular os próprios atributos:

Uma imagem com captura de ecrã

Descrição gerada automaticamente

*Figura 4 – Propriedades da classe Paciente*

Semelhante a classe Pessoa, a classe Paciente também usa um enumerador para determinar a condição de um paciente:

Uma imagem com fotografia, preto, sentado, telefone

Descrição gerada automaticamente

*Figura 5 – Enumerador que determina condição de paciente*

**Medico:**

É constituída pelos atributos:

* Especialidade
* Lista de Pacientes

E contem propriedades que permitem manipular os próprios atributos:

Uma imagem com captura de ecrã, monitor, ecrã, preto

Descrição gerada automaticamente

*Figura 6 – Propriedades da classe Paciente*

*Uma imagem com captura de ecrã

Descrição gerada automaticamente*

*Figura 7 – Diagrama de Classes da camada Data Layer*

Esta camada é constituída por 2 classes, Pacientes e Médicos, estas sendo as camadas responsáveis pelas estruturas de dados usadas para armazenar os dados relacionados aos pacientes e médicos.

**Pacientes:**

Esta classe inicializa as listas que armazenam os dados de todos os pacientes que frequentaram o hospital, como também listas diferentes que separam os pacientes dependendo da condição dos mesmos.

Uma imagem com mesa, sentado, preto, portátil

Descrição gerada automaticamente

*Figura 8 –* *Inicialização de listas para armazenar pacientes*

**Métodos:**

* **AddPaciente(Paciente paciente) -** Adiciona paciente à lista de pacientes global, também compara a condição do paciente e adiciona o mesmo na lista dessa condição.
* **ListarTodosPacientes() –**Lista o histórico de pacientes que frequentaram o hospital (Outros métodos parecidos listam os pacientes com certas condições).
* **ObterPosPaciente(string nomPaciente) –** Obtém a posição de um paciente numa lista.
* **RemovePaciente(string nomePaciente) –** Remove um paciente de uma lista dependendo da condição do mesmo.
* **ExistePaciente(string nomePaciente) –** Verifica se existe um paciente numa lista.
* **SavePacientes() –** Guarda dados da lista atual para um ficheiro “.dat”.
* **LoadPacientes() –** Carrega dados de pacientes de um ficheiro “.dat”
* **ListPacientesPorCondicao() –** Usando um Comparable<T>, organiza os pacientes por condição (Urgente).

**Médicos:**

Esta classe inicializa a lista que armazena os dados de todos os médicos no hospital.

Uma imagem com propriedade, telefone, sentado, mesa

Descrição gerada automaticamente

*Figura 9 – Inicialização da lista para armazenar médicos*

**Métodos:**

* **AddMedico(Medico medico) –** Adiciona um medico à lista de médicos.
* **ObterPosMedico(string nomMedico) –** Obtém a posição de um medico na lista.
* **AtribuirPacientes(string nomeMedico, string nomePaciente) –** Atribui um paciente a um medico na lista.
* **RemoveMedico(string nomeMedico) –** Remove um medico da lista.
* **ListarTodosMedicos() –** Lista todos os médicos guardados na lista.
* **ExisteMedico(string nomeMedico) –** Verifica se um medico existe na lista.
* **SaveMedicos() –** Guarda dados de médicos na lista para um ficheiro “.dat”.
* **LoadMedicos() –** Carrega dados de médicos de um ficheiro “.dat”

Uma imagem com captura de ecrã

Descrição gerada automaticamente

*Figura 10 – Diagrama de Classes da camada Business Rule Layer*

Esta camada é constituída por apenas uma classe, esta sendo a classe que recebe os métodos todos da camada de dados para serem usado na camada Main.

# Conclusão

Com a conclusão deste trabalho pratico conseguimos obter um melhor conhecimento da linguagem de programação C#, dos conceitos basilares do Paradigma Orientado a Objetos (POO) e como também as vantagens da utilização da arquitetura “NTier” para a organização e melhor compreensão do código escrito.

Relativamente a problemas encontrados durante a realização do projeto, os mais notáveis consistiam na escolha de estrutura de dados a utilizar para armazenar os dados de pacientes e médicos e como adaptar o projeto já feito a esses critérios novos.

Em conclusão relativamente à unidade curricular, visto que a proposta do projeto foi mais flexível do que o normal achamos este projeto muito interessante de se realizar, pois, nos permitiu maior criatividade no tema do projeto e mais formas de explorar as possibilidades de C#.

# Bibliografia

* Código disponibilizado no Moodle
* https://stackoverflow.com/