INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

RELATÓRIO DE TRABALHO PRÁTICO

Gestão de Pessoas infetadas numa situação de crise de saúde pública

CARLOS FARIA

ALUNO Nº 16946

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, maio de 2020

Índice

1 Síntese **Erro! Marcador não definido.**

2 Introdução 3

2.1 Motivação e Objetivos 3

2.2 Estrutura do programa 3

2.2.1 Organização de Código N-Tier 5

2.2.2 Classe Pessoa 6

2.2.3 Classe Caso 6

2.2.4 Classe Casos 6

2.2.5 Classe Distrito 7

2.2.6 Classe Distritos 7

2.2.7 Classe Hospital 7

2.2.8 Classe Hospitais 8

2.2.9 Enum Estado 8

3 Implementação 9

3.1 Descrição do Problema 9

3.2 Solução 9

3.3 Persistência de dados através de ficheiros 10

4 Conclusão 13

4.1 Lições Aprendidas 13

4.2 Apreciação Final 14

1 Síntese

Este projeto consiste em criar um programa para **gerir pessoas infetadas numa situação de crise de saúde pública**. Com esta aplicação pretende-se registar casos de infeção, editar o estado do caso registado, contabilizar os casos e eliminar casos se necessário, tendo em conta que mesmo que um caso seja dado como recuperado não deva ser eliminado por questões de análise. A aplicação pretende também oferecer várias funcionalidades tais como consultar casos por região, sexo, idades, entre outros… verificar a quantidade de recuperados, mortes e internados, calcular a taxa de crescimento de infetados entre duas datas e criar uma tabela das regiões mais infetadas, entre outros valores.

2 Introdução

2.1 Motivação e objetivos

O programa tem como objetivo **facilitar a gestão de uma crise de saúde pública**, através do tratamento de dados automático e inserção e manipulação de casos durante a crise, para que dessa forma seja mais rápida e intuitiva a perceção dos números para que possam ser feitos estudos à cerca dos mesmos de maneira a chegar a conclusões acerca da situação de crise em questão, para que assim, se possam propor soluções mais eficazes num espaço reduzido de tempo.

2.2 Estrutura do Programa

A estrutura do programa até à data consiste num projeto com **5 classes**:

- Classe **Pessoa**
- Classe **Caso** que **herda** da classe Pessoa
- Classe **Casos** que possui uma Lista **de objetos do tipo Caso**
- Classe **Distrito**
- Classe **Distritos**
- Classe **Hospital** e Classe **Hospitais**
- Classe **Program**
- Enum **Estado** que **define o estado de infeção** de uma pessoa
- Classe **Regras** que **define regras de negócio** que permitem executar ou não métodos da camada de Dados chamados indiretamente pela camada de apresentação.

2.2.1 Organização de Código N-Tier

O código deste projeto está organizado segundo a arquitetura N-Tier, sendo que este **contém 4 camadas**:

- **Camada de Apresentação**:
    - Nesta camada encontra-se a classe **Program,** sendo que esta contém o **Main** que irá gerar a interface de interação com o utilizador e permite mostrar resultados.
- **Camada de Regras de Negócio**:
    - Esta camada possui uma classe chamada **Regras**, serve para definir regras de negócio na utilização de métodos da camada Dados. Esta camada tem como referência outras duas camadas do projeto, sendo estas a camada dos Dados e dos Objetos de Negócio.
- **Camada de Objetos de Negócio**:
    - Esta camada contém três classes, a **classe Pessoa** e a **classe Caso** que deriva da classe Pessoa, a classe Distrito e a classe Hospital. Estas classes não contêm estruturas de dados visto que os valores das mesmas serão guardados nas **classes Casos, Distritos e Hospitais** que estão presentes na camada dos Dados, onde existirá uma "**List**" em cada uma dessas classes que será automaticamente guardada num ficheiro pelo programa e carregada no início do programa caso o ficheiro exista.
- **Camada dos Dados**:
    - Esta camada tem 3 classes, as **classes Casos, Distritos e Hospitais,** estas classes contêm **Listas do tipo Caso, Distrito e Hospital respetivamente e todas elas têm um inteiro "totId"** que possui o valor ID que irá ser atribuído ao próximo objeto que for registado. Nesta camada são guardados todos os dados do programa que precisem de ser colocados em ficheiros de forma a serem persistentes e não "desaparecerem" assim que se feche a aplicação. Esta camada tem como referência a camada dos "Objetos de Negócio (BO)" visto que nessa camada encontram-se as **classes Caso, Distrito e Hospitais** que são do tipo presente na Listas já enunciada.

2.2.2 Classe Pessoa

- Possui **5 variáveis que definem uma pessoa**, essas variáveis guardam informações como o nome da pessoa, data de nascimento, idade, género e distrito onde vive.
- Contém **dois construtores**, sendo que um deles é um construtor por defeito que não recebe qualquer parâmetro e outro é um construtor que recebe como parâmetros todos os atributos da **classe Pessoa** menos o atributo de idade, sendo que este é calculado automaticamente através da data de nascimento.
- Cada atributo da **classe Pessoa** possui **uma *"property"*** correspondente.
- Possui também **um método** que calcula a idade da pessoa automaticamente através da sua data de nascimento

2.2.3 Classe Caso

- **A classe Caso herda todos os atributos da classe Pessoa**
- Possui **dois construtores**, um que não possui qualquer atributo e outro que possui **todos os atributos** da **classe Pessoa** tirando a idade (calculada automaticamente) e **todos os atributos** da **classe Caso**
- Contém também **duas *"properties"*** que definem os dois atributos da **classe Caso**

2.2.4 Classe Casos

- A classe Casos possui **dois atributos**, sendo que **ambos são variáveis *"static",*** uma das variáveis é um **array do tipo "Caso"** e a outra variável **conta a quantidade de casos** sendo que incrementa sempre que se adiciona um novo caso
- Esta classe tem apenas **um construtor**, esse construtor **apenas inicializa a variável casos** presente nos seus atributos com um array de tamanho 100000, esta estrutura será revista pois uma lista será mais eficaz
- A classe possui **apenas uma *"property".*** Esta property apenas define e devolve valores da variável que conta a quantidade de casos, chamada "totalCasos"
- Esta classe possui até à data **11 métodos**. Entre eles um que permite registar um caso, um que permite editar um caso e outro que permite remover um caso, que deverá ser utilizado em situações pontuais, como remover um caso que tenha sido inserido incorretamente.

2.2.5 Classe Distrito

- A classe distrito possui duas variáveis, nome e id;
- Possui um construtor por defeito e outro com o parâmetro “nome”, o id não é atribuído pelo construtor;
- Cada atributo possui uma property correspondente;
- Possui um método “override” que reescreve a função “Equals” da classe Distrito, para que a comparação entre dois distritos siga as regras definidas neste método.

2.2.6 Classe Distritos

- A classe Distritos possui dois atributos, uma lista do tipo “Distrito” e um inteiro “totId” que guarda o valor do próximo “id” a atribuir a um distrito que seja registado.
- Possui um construtor “static” por defeito que inicializa os dois atributos
- Não possui qualquer property
- Contém 7 métodos que permitem registar, remover, editar distritos que façam parte da lista, entre outros…
- Esta classe tem também a capacidade de guardar os seus dados de forma permanente com os seus métodos “Save” e “Load”

2.2.7 Classe Hospital

- A classe hospital possui 5 variáveis, nome, id, capacidade, lotacaoAtual e distrito;
- Possui um construtor por defeito e outro com os parâmetros “nome”, “capacidade” e “distrito”
- Cada atributo possui uma property correspondente;
- Possui um método “override” que reescreve a função “Equals” da classe Hospital, para que a comparação entre dois hospitais siga as regras definidas neste método.

2.2.8 Classe Hospitais

- A classe Hospitais possui dois atributos, uma lista do tipo “Hospital” e um inteiro “totId” que guarda o valor do próximo “id” a atribuir a um hospital que seja registado.
- Possui um construtor “static” por defeito que inicializa os dois atributos
- Não possui qualquer property
- Contém 6 métodos que permitem registar, remover, editar hospitais que façam parte da lista, entre outros…
- Esta classe tem também a capacidade de guardar os seus dados de forma permanente com os seus métodos “Save” e “Load”

2.2.9 Enum Estado

- Este enum possui **4 estados** e a cada estado está atribuído um número
- **Recuperado** = 0
- **Infetado** = 1
- **Internado** = 2
- **Morto** = 3

3 Implementação

3.1 Descrição do Problema

Numa situação de crise de saúde pública é pertinente registar dados de forma contínua e obter vários tipos de dados/tabelas/cálculos automáticos rapidamente, para que a interpretação e análise dos dados permita chegar a conclusões que possam levar a soluções mais eficazes no tratamento da crise.

3.2 Solução

Para ajudar na gestão de uma crise de saúde pública, a aplicação irá permitir registar, editar e eliminar casos, tal como fornecer vários tipos de funcionalidades que permitam apresentar tabelas de dados e cálculos relevantes. Para que dessa forma, se possam criar soluções que poderão abrandar a crise e erradicar o problema com mais rapidez.

3.3 Persistência de Dados

Para tornar os dados importantes do programa persistentes, recorreu-se à utilização de ficheiros. O programa gera seis ficheiros.

- No ficheiro **casos.bin** é guardada a lista de casos da classe Casos presente na camada de Dados. Se este ficheiro já existir antes da abertura do programa, a lista presente nessa classe irá receber a informação desse ficheiro de modo a que possa ser utilizada com valores de iterações anteriores.
- No ficheiro **ids.bin** é guardado o maior número ID até ao momento, esse número irá ser atribuído ao próximo caso que for registado na lista dos casos e de seguida incrementado para que não existam dois casos com o mesmo ID.
- No ficheiro **distritos.bin** é guardada a lista de distritos da classe Distritos presente na camada de Dados. Se este ficheiro já existir antes da abertura do programa, a lista presente nessa classe irá receber a informação desse ficheiro de modo a que possa ser utilizada com valores de iterações anteriores.
- No ficheiro **idsDistritos.bin** é guardado o maior número ID até ao momento, esse número irá ser atribuído ao próximo distrito que for registado na lista dos distritos e de seguida incrementado para que não existam dois distritos com o mesmo ID.
- No ficheiro **hospitais.bin** é guardada a lista de hospitais da classe Hospitais presente na camada de Dados. Se este ficheiro já existir antes da abertura do programa, a lista presente nessa classe irá receber a informação desse ficheiro de modo a que possa ser utilizada com valores de iterações anteriores.
- No ficheiro **idsHospitais.bin** é guardado o maior número ID até ao momento, esse número irá ser atribuído ao próximo hospital que for registado na lista dos hospitais e de seguida incrementado para que não existam dois hospitais com o mesmo ID.

4 Conclusão

4.1 Lições Aprendidas

Com o trabalho desenvolvido até ao momento, aprendi acerca da **importância da utilização de heranças** quando o programa assim o permite. Até à data, o programa desenvolvido possui **uma herança entre a classe Pessoa e Caso**, sendo que um Caso é uma pessoa infetada, logo é uma pessoa e por isso herda os atributos dessa classe. Aprendi também a **organizar as classes de uma forma intuitiva com a utilização de regions**. Antes deste trabalho não sabia qual era o interesse da utilização de **enums** mas como pode ser observado, este projeto **contém um enum que classifica o estado do caso de uma pessoa**, em que a cada estado está atribuído um número correspondente, a utilização deste enum veio a simplificar este processo. Utilização de "**Generics"** e a suas vantagens, utilização de uma **arquitetura de programação**, neste caso a arquitetura **N-Tier**, com este trabalho consegui já perceber as vantagens de organizar o código de um programa por camadas, sendo que desta forma ficará mais fácil de manusear/alterar aspetos em relação ao código sem precisar de alterar outras camadas ou o código na totalidade. Aprendi também a criar **persistência de dados através da utilização de ficheiros**.

4.2 Apreciação Final

Com o trabalho realizado até ao momento já foram desenvolvidas as classes mais **pertinentes** do projeto, sendo que com estas classes é possível realizar as operações mais importantes no que toca à gestão de uma crise de saúde pública, pois estar a par da quantidade de pessoas infetadas, o estado das mesmas, a sua região, idade, etc… é algo que torna a capacidade de resposta de uma nação a uma situação de crise de saúde pública mais rápida e eficaz, pelo menos até que seja encontrada uma cura ou vacina. O projeto contém uma quantidade reduzida de métodos, pois estes ainda vão ser desenvolvidos. Com as pesquisas que precisei de fazer e as revisões que efetuei, sinto agora uma maior capacidade de programar na linguagem c#, ainda há muito que aprender, mas sinto que este trabalho me vai fazer evoluir enquanto programador. A implementação da **arquitetura N-Tier** melhorou bastante a organização do código e a capacidade de poder ser alterado sem afetar outras camadas senão aquela que está a ser alterada, com esta arquitetura torna-se também mais fácil melhorar o código e as suas funcionalidades e perceber onde estão os problemas caso eles existam. O código tem a capacidade de **armazenar os seus dados relevantes em ficheiros**, visto que é algo necessário tendo em conta que o programa precisa dos dados da sua última utilização de forma a ser possível obter análises estatísticas com todos os dados inseridos até à data e não apenas com os dados colocados na iteração atual.

Bibliografia

Durante o desenvolvimento do trabalho, foram visitados os seguintes websites:

- https://stackoverflow.com/
- https://docs.microsoft.com/en-us/dotnet/api/system.enum.equals?view=netcore-3.1
- https://docs.microsoft.com/en-us/dotnet/api/system.datetime.compare?view=netcore-3.1
- https://stackify.com/n-tier-architecture/