Licenciatura em Sistemas Informáticos

Linguagens de Programação II

Relatório do Trabalho Prático

André Cardoso & José Cosgrove
18848 & 18826

Barcelos, 22 de maio de 2020

Contents

1 Introdução

Foi proposto para a unidade curricular de Linguagens de Programação II, o desenvolvimento de uma solução em C# para problemas reais de complexidade moderada. Para tal é necessário provar:

- Compreensão dos conceitos básicos de programação segundo o Paradigma Orientado a Objetos.
- Capacidade de Análise de problemas reais.

1.1 Motivação

Pretende-se o desenvolvimento de soluções em C# para problemas reais, trabalhando o paradigma de POO.

Numa primeira fase foi nos requerido uma implementação essencial das classes a utilizar, uma boa documentação do código, a criação do seu diagrama, a utilização de software gestor de versões e ainda a redação de um relatório de qualidade.

Numa segunda fase, já devemos mostrar um pouco mais com aquilo que o programa final será, tento em conta conceitos como, heranças, encapsulamento, abstração e polimorfismo, interfaces, exeptions, generics, estrutura de código por camadas e tratamento de ficheiros.

Para finalizar com a terceira parte, devemos ter o programa já concluído com todas as suas funcionalidades e bem estruturado.

2 POO - Programação Orientada a Objetos

A POO é um dos grandes paradigmas da programação.

A principal forma de utilização desta técnica é a determinação de estruturas e operações referentes a um objeto (classes). Desta forma podemos determinar que um objeto obedece a um determinado conjunto de regras, podendo ter atributos (Properties) e comportamentos(Methods).

A POO acenta ainda em 4 pilares fundamentais:

- Heranças
- Encapsulamento
- Abstração
- Polimorfismo

Estes 4 pilares são bem demonstrados no projeto. A herança através da classe-pai Pessoa e as classes-filho Árbitro e Jogador. O encapsulamento e abstração pelos atributos e métodos sendo privates, sendo estes acessíveis através das proprieties e não diretamente à variável.

3 Enumeráveis

Para as classes Arbitro e Jogador é necessário indicar uma ASSOCIACAO e CATEGORIA, e uma POSICAO, respetivamente.

- ASSOCIACAO
  Contém todas as associações de Árbitros de Portugal.
- CATEGORIA
  Discriminação dos níveis profissionais que um árbitro pode alcançar.
- POSICAO
  Enumerável com as posições 'gerais' que um jogador pode ocupar no decorrer do jogo.

4 MVC - Model View Controller

Para uma primeira fase foi-nos requerido uma implementação básica e essencial de classes. No entanto, nesta segunda fase, apresentamos uma estrutura de classes avançada. O Programa encontra-se estruturado segundo o padrão de MVC. De modo a relacionarmos as Classes utilizamos Interfaces para cada membro MVC das classes. Vamos falar do modo que a Interface mantém as relações entre os membros MVC a seguir.

De salientar que cada um dos seguintes continuam a ser classes e devem manter:

- Variáveis de Membro –> Aqui são inicialmente definidas as variáveis como private.
- Construtores –> Os construtores são onde definimos quais os parâmetros que vão corresponder às variáveis, que podem ser definidas pelo utilizador, pelo programa, ou ficarem sem valor
- Propriedades –> São definidos os get e set para cada atributo, podendo alguns sofrer algumas verificações.
- Métodos –> Funções próprias da classe, definidas para serem utilizadas com a classe.

4.1 Model

Os models são as classes onde estão definidos cada tipo de dados e quais as suas variáveis. A sua principal função é ser a estrutura para os tipos de dados definidos.

4.2 View

A view nada mais é do que o Input/Output entre a aplicação e o utilizador, sendo definidas aqui todas as ações a que o utilizador tem acesso na sua tela, fazendo desta forma uma ligação entre o utilizador e os controllers.

4.3 Controller

O controller é onde estão definidos os requests e responses entre toda a aplicação. É feito um request pelo utilizador através da view, o controller processa esse pedido acendendo aos models, enviando um response, de forma a dar resposta à pergunta.

5 Interfaces

Uma interface é implementada para cada classe criada. Em cada interface deve estar definido o método em utilização.Por convenção, é adicionado um 'I' antes do nome da classe que vai ser utilizada e a classe que vai utilizar a interface como herança. Deste modo, para qualquer classe model, view e controller existe uma interface correspondente à classe.

6 Generics

Numa primeira fase do trabalho eram utilizados arrays para fazer conjuntos de objetos, o que iria claramente trazer um grande problema: o espaço de memória, uma vez que ao declarar um array temos também de declarar o número de espaço a reservar na memória.

Os Generics vieram para levar esse problema e ainda dar auxilio através de funções já pré-definidas.

O generic utilizado nesta aplicação foi o List<>, onde podemos utilizar métodos já pré-definidos, tais como:

- .Add() –> Insere um novo objeto à lista.
- .Remove() –> Remove o objeto da lista.
- .Contains() –> Verifica se o bjeto já se encontra na lista.
- .ForEach() –> Percorre toda a lista, elemento a elemento.
- .IndexOf(Object) –> Retorna a posição na lista do objeto.

Alguns exemplos onde são utilizadas listas são ao construir uma equipa, onde existe um conjunto de jogadores.

7 Exceptions

As exceptions trazem auxilio no que toca ao tratamento de erros. Existem 2 tipos de exceptions:

- System Exceptions
- Aplication Exceptions

Tal como o próprio nome indica, um trata de resolver os erros executados pelo sistema, e a outra, de resolver os problemas gerados pela aplicação.

Dentro dos erros de sistemas existem várias formas de apresentar os erros. Os dois mais utilizados nesta aplicação foram:

- FormatException –> Utilizado para precaver erros quando existe a conversão de um tipo de dados para outro tipo.
- OverflowException –> Utilizado quando uma variável excede a sua capacidade máxima.

As exceptions são utilizadas neste programa na sua forma mais simples, uma vez que poderiamos construir classes para dar uma melhor resposta ao tipo de erros. Poderia ser uma melhoria futura neste projeto.

8 Tratamento de Ficheiros

Sobre tratamento de ficheiros, utilizamos uma classe guarda<Model> onde estão inseridas funções de:

- Escrita –> Utiliza as classes "Stream" com "File" para criar um ficheiro binário. Se salientar que as classes model têm de ser Serialize, pois caso não sejam, fica impossível guardar uma lista desse model;
- Leitura –> Utiliza novamente as mesmas classes para fazer a leitura do ficheiro e atribui a uma lista (variável de membro) os valores;
- Procura –> Nada mais faz do que receber como parâmetro uma variável e compara-la à lista, assim que for encontrada uma semelhança é retornado o valor encontrado.

Existem ainda outras funções como o Add() ou GiveList() que, respetivamente, adiciona um novo elemento à lista e retorna toda a lista.

Os ficheiros estão a ser guardados, por defeito, no diretório bin/Debug.

9 Aplicacao

Esta aplicação corre em linha de comando, tendo como base a gestão de competições, onde são, por consequência, geridos também Jogadores, Arbitros, Equipas e Jogos.

Esta aplicação não utiliza para todos os deletes remoções físicas, mas sim lógicas, para que dessa forma exista sempre consistência de dados. Para isso foi adicionado em cada model um atributo active do tipo bool para mostrar que aquele elemento está "eliminado" ou não.

Não são utilizados id's para fazer corresponder a cada elemento de uma lista, pois verificamos que pudemos fazer corresponder um id ao index da lista onde o objeto se encontra inserido. Isto não seria possível se não utilizássemos eliminações lógicas.

As principais funcionalidades de cada um destes são:

9.1 Gestão de Jogadores

Através do modelo Pessoa, o jogador tem como variáveis o nome, data de nascimento, peso, altura, nacionalidade e também alcunha, número e posição. Cada vez que é adicionado um jogador novo, é feito um pedido ao controller para adicionar um novo jogador, o controller chama a view e o utilizador insere os dados respetivos, estes dados são enviados ao controller que é o responsável por fazer chegar estes dados ao model guardaJogador, que fica por sua vez, responsável por inserir na lista de jogadores este novo elemento.

Este processo repete-se para o mostrar todos os jogadores, apresentando apenas dados básicos ou para mostrar um jogador de forma individual, apresentando todos os dados relativos ao jogador.

Temos ainda o editar e remover - funções que as restantes classes também possuem - que não vão ser enviados para o model guardarJogador, mas sim para o próprio model Jogador, uma vez que para editar, apenas vai substituir os valores daquele objeto e para remover, uma vez que não trabalhamos com remoções físicas, mas sim lógicas, altera o atributo do tipo booleano Active para false.

9.2 Gestão de Árbitros

Tal como os Jogadores, a classe Árbitro deriva da classe pessoa pelo que têm essas propriedades em comum. No entanto, é-lhe único a sua categoria e associação. Através dos Menus é possibilitado o pedido de Inserção de um Árbitro, efectuado por uma view onde inserimos todos os dados necessários, ao que de seguida são enviados para o controller os dados inseridos, que são reencaminhados para o model guardaArbitro, responsável pela adição do novo Árbitro na sua lista.

O processo é sempre igual independentemente da função desempenhada, View -> Controller -> Model. No entanto, quando é pedido a apresentação dos dados, há um retorno inerente a esse pedido ao que é entregue ao controller os dados e de seguido o controller entrega esses mesmo dados ao view.

9.3 Gestão de Equipas

Neste modelo é guardado o nome e data de fundação da equipa e ainda uma lista com os id's de todos os jogodores que os constituem. Para alem do CRUD já normal são ainda criados mais um editar para a lista de jogadores e um remover, que desta vez utiliza mesmo uma eliminação física.

Alguns factos a notar são que a aplicação não permite inserir jogadores que já estejam registados em outros clubes.

9.4 Gestão de Jogos

Para os Jogos é necessário o nome das duas equipas que se vão defrontar, duas variáveis para o resultado do jogo, e uma lista para os árbitros participantes no jogo. Para alem do CRUD já normal são ainda criados mais um remover (remoção física) e inserção para a lista de árbitros.

Na altura da inserção do jogo, é questionado a que competição ele pertence, à qual é associado imediatamente.

9.5 Gestão de Competição

Este é capaz de ser o ponto mais critico do trabalho, uma vez que engloba num modelo tudo o que já teve de ser inserido anteriormente e é o primeiro elemento de relação entre todos os models.

Aqui é possível criar uma competição, dando-lhe um nome, data de inicio e fim, adicionar as equipas que fazem parte da competição e os jogos correspondentes a essa competição.

10 Conclusão

Agora que chegamos ao fim de um trabalho e de um semestre de muito trabalho e que, de certa forma nos abalou por causa de todas as adversidades que este ano nos trouxe, chega o momento de reflexão.

Sentimo-nos postos à prova a cada vez que tinhamos de dar continuação ao trabalho, ao pensar em como dar a volta às adversidades cada vez que elas surgiam e também ao ter de aprender cada vez mais para puder cada vez mais melhorar o trabalho.

Apesar de o trabalho não ter tido a conclusão que queriamos, pois pretendiamos criar ainda mais 2 funções, uma que apresentasse a classificação das equipas numa determinada competição e outra que mostrasse estatisticas de uma competição, por exemplo melhor marcador.

Apesar de tudo isto, acreditamos que adquirimos os valores essenciais desta cadeira e que fizemos o melhor que sabíamos, podendo por vezes ter tido um desacordo ou outro, mas respeitando sempre toda a gente e trabalhando para achar uma solução.

11 Webgrafia

1. Sitio oficial da microsoft, Exceptions
   `https://docs.microsoft.com/en-us/dotnet/api/system.exception`
2. Sitio oficial da microsoft, Generic Lists
   `https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1`
3. Sitio oficial da microsoft, Interfaces
   `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface`