Programação Orientada aos Objetos

Grupo 45

Junho 2021

A89572
José Ferreira

A89532
Guilherme Martins

A89487
Rúben Lucas

Resumo

O presente relatório possui como finalidade a descrição das soluções implementadas para os desafios impostos pela equipa docente da unidade curricular Programação Orientada aos Objetos.
O propósito deste trabalho consiste no desenvolvimento de um jogo ao estilo de títulos como *Football Manager*. As soluções implementas no projeto são o resultado do aplicação dos conhecimentos adquiridos nas aulas teóricas e práticas da UC ao longo do semestre.

Índice

1 Introdução

No âmbito da Unidade Curricular de Programação Orientada aos Objetos, foi-nos proposto o desenvolvimento de um programa semelhante ao conhecido jogo *Football Manager*. Pretendia-se criar um sistema de gestão e simulação de equipas e jogos de um determinado desporto. O nosso projeto usou o futebol como desporto que serve de base ao jogo.

O foco prioritário deste projeto foi o encapsulamento das estruturas de dados utilizadas, respeitando assim a metodologia da programação orientada aos objetos.

Ao longo deste relatório vamos entrar em detalhe em relação ao desenvolvimento das várias componentes da solução implementada para responder ao enunciado proposto.

2 Arquitetura de Classes

De modo a desenvolver a aplicação é necessário organizar dados numa estrutura que, apesar de compacta, seja de rápido acesso.

As classes elementares da nossa estrutura representam Jogadores, Equipas e Jogos.

Com o avançar do projeto, verificamos que seria necessária uma classe que agregasse as classes anteriormente referidas para que estas pudessem interagir entre si e, ainda, precisaríamos de uma classe capaz de interagir com o utilizador. Assim, decidimos proceder à implementação do modelo MVC, não só para obter independência das várias camadas (Modelo, Vista e Controlador), mas também para poder interagir com o utilizador de forma segura.

Assim, todos os dados destas classes são agregados numa Base de Dados geral à qual chamamos de GestFootManager, que controla o Modelo do programa. Além disso, temos a classe Interpretador responsável pelo controlo do programa e a classe apresentação responsável pela vista.

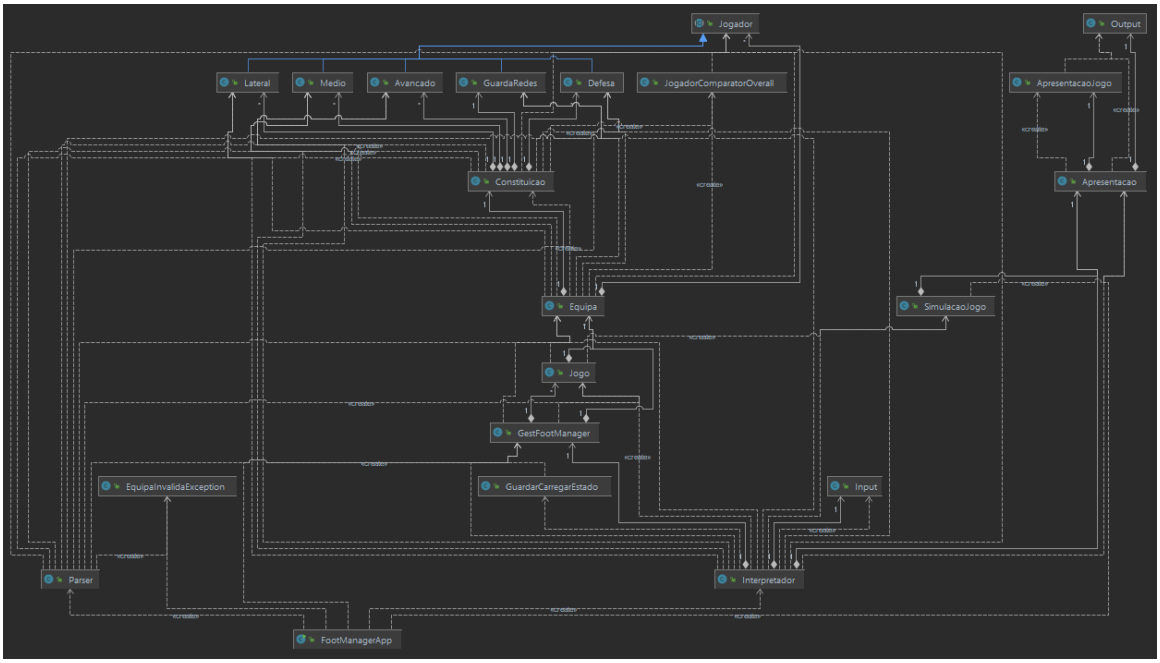

Figure 1: Diagrama de Classes da Solução

2.1 FootManagerApp

Esta classe contém a main que corre o programa. É a classe que contêm os vários Módulos do MVC e que executa o interpretador.

```
GestFootManager gfm = new GestFootManager();  // Modelo
Interpretador i;                              // Controlador
Apresentacao a = new Apresentacao();          // Vista
Parser parser = new Parser();                 // Parser dos Logs
```

2.2 GestFootManager

Esta é a classe responsável pelo Modelo do programa. Aqui encontram-se todas as estruturas do nosso programa. Esta classe implementa a interface IGestFootManager.

```
// Map que contem as equipas registadas no sistema
private Map<String,Equipa> equipas;
// Map que contem os jogos registados no sistema
private Map<LocalDate, List<Jogo>> jogos;
// String com o nome da Equipa que o user controla
private String minhaEquipa;
```

Decidimos utilizar Maps em vez de Sets nas nossas estruturas devido à enorme vantagem de, durante a execução dos métodos, apenas guardarmos os códigos de cada elemento (nas equipas o seu nome e nos jogos a sua data) e utilizarmos isso como key para aceder ao Map.

Além disso, optamos pela utilização de HashMaps em vez de TreeMaps porque a ordenação dos TreeMaps não nos traz vantagens neste caso, sendo então os tempos dos HashMaps mais vantajosos.

Temos também uma String que guarda o nome da Equipa que o utilizador está a controlar. Esta é utilizada para sabermos que o utilizador apenas pode fazer a Gestão da sua equipa.

2.2.1 Equipa

Esta é a classe com a informação de uma equipa.

```
// Nome da Equipa
private String nome;
// Map com os jogadores da equipa
private Map<Integer,Jogador> jogadores;
// Rating da Equipa
private int overall;
// Constituicao da Equipa Titular
private Constituicao constituicao;
```

Para guardar os jogadores de uma dada equipa, utilizamos um Map em que a key será o número da camisola do jogador. O overall é calculado aquando do registo da equipa e da saída ou entrada de um jogador. A constituição representa o escalonamento da equipa titular.

2.2.2 Jogador

Esta é a classe com a informação de um jogador. De notar que esta classe é abstrata, uma vez que vai ser super-classe para as classes que definem um Jogador pela sua posição, descritas adiante neste relatório.

```
// Nome do jogador
private String nome;
// Rating do Jogador
private int overall;
// Numero da Camisola do Jogador
private int numeroCamisola;
// Set com os Nomes das Equipas onde o Jogador ja jogou
private Set<String> historico;
// String com o nome da equipa do Jogador
private String equipaAtual;

// Atributos
private int velocidade;
private int resistencia;
private int destreza;
private int impulsao;
private int jogodecabeca;
private int remate;
private int passe;
```

No histórico, guardamos os nomes de todas as equipas em que um Jogador já jogou. Decidimos utilizar um Set uma vez que não temos interesse em que hajam repetições de Strings no histórico de um Jogador.

De seguida são mostradas as classes que descrevem cada Posição. De notas que algumas destas classes adicionam atributos aos atributos base descritos na classe Jogador.

2.2.2.1 Guarda-Redes

Esta classe extende a classe Jogador para descrever um jogador cuja posição é Guarda-Redes.

```
public class GuardaRedes extends Jogador implements Serializable {
    private int elasticidade;
```

```
3
4       /**
5        * Metodos da classe
6        */
7        (...)
8   }
```

2.2.2.2 Defesa

Esta classe extende a classe Jogador para descrever um jogador cuja posição é Defesa. Os defesas não terão nenhum atributo diferente da classe Jogador, logo apenas irão herdar as variáveis da sua super-classe.

```
1   public class Defesa extends Jogador implements Serializable {
2       /**
3        * Metodos da classe
4        */
5        (...)
6   }
```

2.2.2.3 Lateral

Esta classe extende a classe Jogador para descrever um jogador cuja posição é Lateral. Consideramos como Laterais os Defesas Laterais e Extremos.

```
1   public class Lateral extends Jogador implements Serializable {
2       private int cruzamentos;
3
4       /**
5        * Metodos da classe
6        */
7        (...)
8   }
```

2.2.2.4 Medio

Esta classe extende a classe Jogador para descrever um jogador cuja posição é Médio.

```
1   public class Medio extends Jogador implements Serializable {
2       private int recuperacao;
3
4       /**
5        * Metodos da classe
6        */
7        (...)
8   }
```

2.2.2.5 Avançado

Esta classe extende a classe Jogador para descrever um jogador cuja posição é Avançado. Os avançados não terão nenhum atributo diferente da classe Jogador, logo apenas irão herdar as variáveis da sua super-classe.

```
public class Lateral extends Jogador implements Serializable {
    /**
     * Metodos da classe
     */
     (...)
}
```

2.2.2.6 JogadorComparatorOverall

Em algumas ocasiões no nosso programa, vimos a necessidade de organizar uma lista de Jogadores. Para tal, criamos este comparador, que será usado para que uma Lista de Jogadores fique organizada em ordem ao rating dos Jogadores, por ordem decrescente. Assim, o primeiro elemento da lista, será o Jogador com maior rating.

```
public class JogadorComparatorOverall implements Comparator<Jogador>
    {
    public int compare(Jogador o1, Jogador o2) {
        return o2.getOverall()-o1.getOverall();
    }
}
```

2.2.3 Constituicao

Como foi dito anteriormente, esta classe representa o escalonamento da equipa titular de uma dada equipa. Nesta classe definimos uma coleção de constantes chamada Tatica, que pode ter os valores *'QTT'* e *'QQD'* (4-3-3 e 4-4-2, respetivamente). A partir desta coleção, cada Constituição tem uma tática que vai controlar o número de Defesas, Laterais, Médios, Extremos e Avançados na Constituição.

```
public class Constituicao implements Serializable {
// Colecao de Constantes que representam os sistemas taticos
private enum Tatica {QTT, QQD}
// Guarda-Redes Titular
private GuardaRedes guardaRedes;
// Array com os Defesas Titulares
private Defesa[] defesas;
// Array com os Medios Titulares
private Medio[] medios;
// Array com os Avancados Titulares
```

```
11 private Avancado[] avancados;
12 // Array com os Defesas Laterais Titulares
13 private Lateral[] laterais;
14 // Array com os Extremos Titulares
15 private Lateral[] extremos;
16 // Assume um dos valores definidos nas constantes
17 private Tatica tatica;
18
19     /**
20      * Metodos da classe
21      */
22      (...)
23
24 }
```

Para cada posição, decidimos usar um array para armazenar os jogadores titulares, uma vez que assim conseguimos definir um tamanho máximo para o array, não permitindo que o sistema tático seja derespeitado.

2.2.4 Jogo

Esta é a classe com a informação de um jogo.

```
1  // Equipa que joga em casa
2  private Equipa equipaCasa;
3  // Equipa que joga fora
4  private Equipa equipaFora;
5  // Total de golos da equipa que joga em casa
6  private int scoreCasa;
7  // Total de golos da equipa que joga fora
8  private int scoreFora;
9  // Data do jogo
10 private LocalDate data;
11 // Lista com os jogadores da equipa que joga em casa
12 private List<Integer> jogadoresCasa;
13 // Map com as substituicoes da equipa que joga em casa
14 private Map<Integer,Integer> subsCasa;
15 // Lista com os jogadores da equipa que joga fora
16 private List<Integer> jogadoresFora;
17 // Map com as substituicoes da equipa que joga fora
18 private Map<Integer,Integer> subsFora;
```

Para guardar a informação das substituições de ambas as equipas decidimos utilizar um Map em que conseguimos associar uma key para o jogador que sai e um value para o jogador que entra.

2.3 Files

No package Files incluímos duas classes responsáveis por tratamento de ficheiros e uma *exception*.

2.3.1 Parser

A classe Parser é responsável pela leitura, tratamento e armazenamento dos dados de um ficheiro de logs.

```
1 public class Parser {
2
3     public void parse(GestFootManager gfm) throws
    FicheiroInvalidoException {...}
4
5     public static List<String> lerFicheiro(String nomeFich) {...}
6 }
```

2.3.2 GuardarCarregarEstado

A classe GuardarCarregarEstado apresenta apenas dois métodos em que o primeiro trata te guardar o estado da aplicação em modo binário e que irá obrigar a todas as classes que serão persistidas a implementar a interface *Serializable*. O segundo, por sua vez, é responsável por ler e carregar o ficheiro guardado em modo binário.

```
1 public class GuardarCarregarEstado {
2
3     public static int guardaDados(String fileName, GestFootManager
    gfm) {...}
4
5     public static GestFootManager carregaDados(String fileName)
    throws IOException, ClassNotFoundException {...}
6 }
```

2.3.3 FicheiroInvalidoException

Esta *exception* foi criada para ser usada no programa sempre que o formato de um Ficheiro de Logs for inválido. Isto é, uma linha do ficheiro de Logs deve começar com uma das seguintes Strings:

- "Equipa"
- "Guarda-Redes"

- ”Defesa”
- ”Lateral”
- ”Medio”
- ”Avancado”
- ”Jogo”

Se uma linha do ficheiro não começar por nenhuma destas *keywords*, vamos dar throw à excepção *FicheiroInvalidoException*.

2.4 Controlador

2.4.1 Interpretador

Esta é a classe responsável pelo controlador do programa, ou seja, é a classe que interage com o utilizador. Esta classe implementa a interface IInterpretador.

```
public class Interpretador implements IInterpretador {
    // Respons vel pela leitura de Input do utilizador
    private final Input in;
    // Modelo da Aplicacao
    GestFootManager gfm;
    // Vista da Aplicacao
    Apresentacao apresentacao;

    /**
     * Metodos da classe
     */
     (...)
}
```

2.4.2 Input

Esta classe contém dois métodos que permitem ler inteiros e ler linha a linha que servirá de auxílio à classe Interpretador ao ler do input.

```
public class Input {

    public int lerInt(){...}

    public String lerLinha() {...}
}
```

2.4.3 SimulacaoJogo

Esta classe é responsável pela simulação completa do jogo, sendo detalhado todos os momentos importantes, incluindo os golos que atualizam sempre o resultado final.

```
// String com o nome da equipa que joga em casa
private String casa;
// Rating da equipa que joga em casa
private int ratingCasa;
// Numero de golos da equipa que joga em casa
private int golosCasa;
// String com o nome da equipa que joga fora
private String fora;
// Rating da equipa que joga fora
private int ratingFora;
// Numero de golos da equipa que joga fora
private int golosFora;
// Classe responsavel pelos prints
Apresentacao apresentacao;
```

Utilizamos uma string para guardar os nomes de ambas as equipas para nos facilitar o output de mensagens contendo os nomes das equipas, tornando o jogo mais rico.

Os ratings das equipas influenciarão as probabilidades de acontecimentos no decorrer do jogo, pois quanto maior o rating, maior o número de acontecimentos (golos, penaltis e cantos).

A chamada da classe apresentação é importante, já que nos permite imprimir mensagens durante a simulação do jogo.

2.5 Vista

2.5.1 Apresentacao

Esta é a classe responsável por apresentar conteúdo ao utilizador. Esta classe implementa a interface IApresentacao.

```
public class Apresentacao implements IApresentacao {

    // Imprime os outputs ao utilizador
    private final Output out;
    // Apresentacao responsavel pelos outputs relacionados com um
   jogo
    private ApresentacaoJogo aj;

    /**
     * Metodos da classe
     */
```

```
    (...)
}
```

2.5.2 Output

Esta classe implementa três métodos que vão ser usados para passar informação ao utilizador.

```
public class Output {
    /**
     * Apresenta mensagem numa linha com um '\n' no final
     * @param message mensagem
     */
    public void printMessage(String message) {...}

    /**
     * Apresenta mensagem numa linha
     * @param s mensagem
     */
    public void printPrompt(String s) {...}

    /**
     * Limpa o ecra tornando a informacao mais legivel para o utilizador
     */
    public static void clearScreen() {...}
}
```

2.5.3 ApresentacaoJogo

Esta classe é responsável por passar ao utilizador os eventos ocorridos durante a simulação de um Jogo.

3 Descrição da Aplicação

O modo de interação com o utilizador escolhido foi uma interface de linha de comandos.

3.1 Mensagem de Entrada e Menu Inicial

Quando o utilizador entra no programa, depara-se com uma Mensagem de boas-vindas. Daí, basta pressionar *enter* para exibir o Menu Inicial.

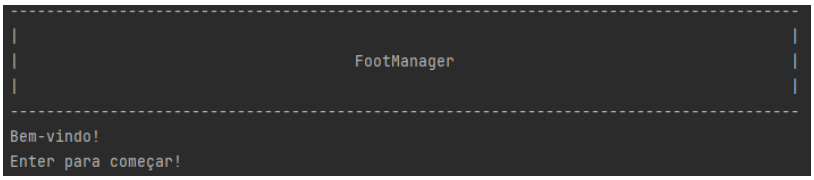

Figure 2: Mensagem de Entrada

Figure 3: Menu Inicial

No menu princial o utilizador tem as seguintes opções:

1. Criar um novo jogo.
2. Carregar o estado de um jogo anterior.
3. Sair do programa

3.1.1 Criar um Novo Jogo

Quando o utilizador escolhe começar um jogo novo, é-lhe apresentada a lista de todas as equipas registadas no sistema para que escolhe a equipa que quer controlar.

3.1.2 Carregar Jogo

Quando o utilizador escolhe carregar o estado de um jogo que foi gravado anteriormente, é-lhe apresentado um prompt a pedir o nome do ficheiro de save.

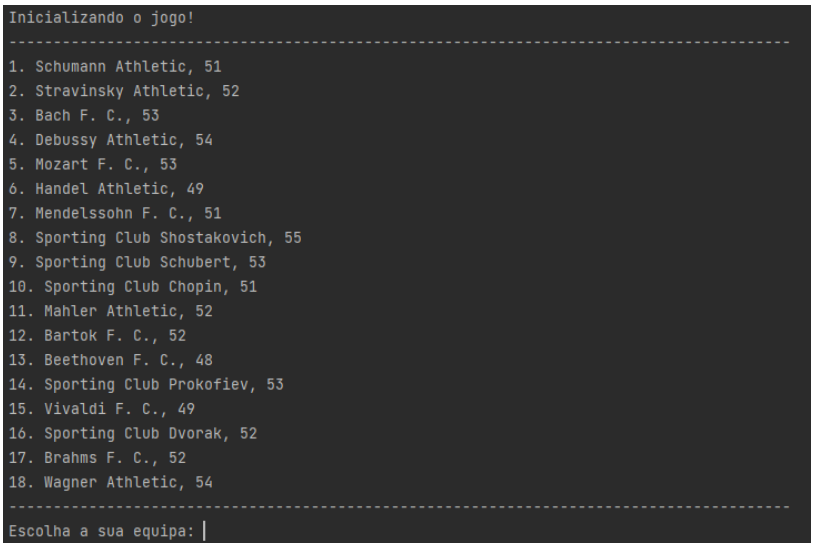

Figure 4: Criar um Jogo Novo

3.2 Menu Principal

Iniciado um novo jogo ou carregado um estado anterior, vai ser apresentado ao utilizador o menu principal do jogo.

Figure 5: Menu Principal

3.2.1 Gestão de Equipa

O menu de gestão de Equipa permite ao utilizador:

- Editar a constituição da sua equipa
- Transferir jogadores
- Ver a constituição atual da sua equipa
- Ver os resultados dos jogos realizados pela sua equipa

- Consultar o plantel da sua equipa

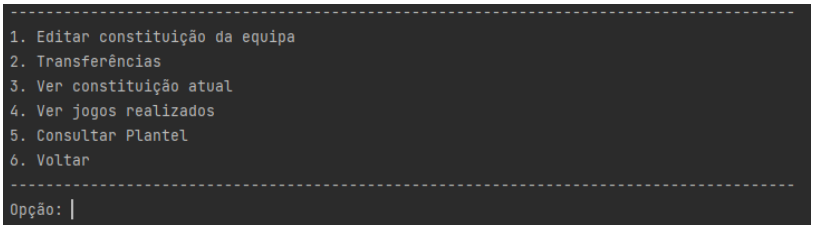

Figure 6: Menu Gestão de Equipa

3.2.1.1 Editar Constituição da Equipa

Escolhendo editar a constituição da equipa, o utilizador pode ver a constituição atual e alterá-la.

Se escolher ver a constituição, ser-lhe-á apresentada a tática e os jogadores titulares em cada posição.

Se escolher alterar a constituição, será questionado sobre a tática que pretende para a sua equipa. Ao escolher uma tática, o sistema gera o melhor onze titular. O utilizador será questionado se pretende ver a nova constituição e/ou alterá-la. Se decidir alterá-la, percorrerá as várias posições, podendo trocar jogadores titulares com titulares ou suplentes.

3.2.1.2 Transferir Jogadores

O utilizador pode escolher vender um jogador da sua equipa para outra equipa ou comprar um jogador de outra equipa para a sua.

Na venda de jogadores, é apresentada uma lista dos jogadores do plantel da equipa do utilizador. O utilizador escolhe o jogador a vender e é-lhe apresentada a lista de todas as equipas do sistema. Escolhe a equipa para onde transferir o jogador e efetua a transferência.

Na compra de jogadores, é apresentada uma lista de todas as equipas para o utilizador escolher a qual vai comprar um jogador. De seguida, é apresentada a lista de jogadores da equipa selecionada. Aqui o utilizador escolhe o jogador a transferir e efetua a transferência.

De notar que ao realizar estras transferências, os atributos *equipaAtual* e *historico* de cada jogador são alterados conforme os eventos, tal como o Map de jogadores das equipas envolvidas.

3.2.1.3 Ver Constituição da Equipa

Se escolher ver a constituição da sua equipa, ser-lhe-á apresentada a tática e os jogadores titulares em cada posição.

3.2.1.4 Ver Jogos Realizados

Escolhendo ver os jogos realizados, serão apresentados todos os resultados dos jogos realizados pela equipa do utilizador, ordenados por ordem cronológica.

3.2.1.5 Consultar Plantel

Escolhendo consultar o plantel da sua equipa, será apresentado ao utilizador a lista de todos os jogadores da sua equipa, podendo o utilizador escolher um jogador para ver em específico.

3.2.2 Realizar Jogo

Para realizar um jogo, o utilizador tem duas opções:

- Simulação Rápida
- Simulação Completa

Nas duas opções o utilizador deve escolher as duas equipas participantes do jogo.

3.2.2.1 Simulação Rápida

Na simulação rápida, apenas é apresentado o resultado de uma simulação do jogo entre duas equipas. Esta simulação é feita através de cálculos probabilísticos e dos ratings das equipas.

3.2.2.2 Simulação Completa

Na simulação completa, é corrido um método de uma classe Simulação Jogo que simula ticks de um jogo, e caso haja um evento, imprime essa notificação ao utilizador. Esta simulação é feita através de cálculos probabilísticos e dos ratings das equipas.

3.2.3 Guardar Jogo

Quando o utilizador escolhe gravar o estado do jogo, é-lhe questionado o nome do ficheiro a gravar com o estado do jogo atual.

4 Conclusão e Reflexão Crítica

Com o desenvolvimento deste projeto ao longo do semestre sentimos que nos foi permitido aprofundar todos os aspetos e os conhecimentos adquiridos face à linguagem de programação Java e também de todo o paradigma de programação orientada aos objetos. Somos totalmente da opinião que desenvolver uma aplicação deste género, em termos de abstração de código, reutilização ou até mesmo modularidade, é extremamente facilitada recorrendo ao uso desta linguagem e deste paradigma, ao contrário de se tivessemos desenvolvido a aplicação tendo como base a linguagem C, por exemplo.

A possibilidade de utilização de interfaces e classes abstratas, facilitou imenso o nosso percurso no decorrer do semestre, já que a adição de novas funcionalidades não é uma complicação devido a toda uma flexibilidade criada pela herança.

Em suma, o grupo considera que o trabalho foi bem conseguido, uma vez que atingimos todos os objetivos pretendidos, não nos ficando apenas pelos requisitos base e correspondendo à totalidade do enunciado.