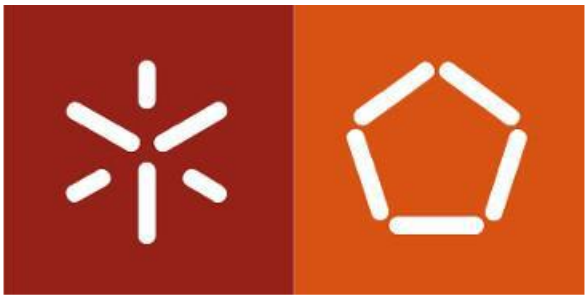

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Programação Orientada aos Objetos

Projeto: “Football Manager”

Grupo 102

Duarte Nuno Pereira Moreira (a93321)

Lucas da Silva Carvalho (a93176)

Ano 2020/21

Índice

Introdução e desafios

Este projeto foi realizado no âmbito da disciplina de POO e consiste na construção de um programa que seja semelhante ao conhecido jogo Football Manager. A linguagem de programação utilizada é o Java.

De entre os principais desafios destaca-se o facto de não ter sido feita uma boa gestão do tempo de forma que o projeto tivesse sido terminado. Para além disso acreditamos que o passo mais difícil tenha sido iniciar o projeto tendo uma visão global da estrutura a utilizar. Depois de ultrapassar este último problema, toda a construção do trabalho fluiu sem grandes obstáculos.

Classes

a. FootballManager

Esta é a classe principal e apenas tem a função de arrancar o programa. Para tal o método main invoca o método run da classe Controller.

```
public class FootballManager {
    public static void main(String[] args) {
        Controller.run();
    }
}
```

b. Controller

A classe Controller, como o próprio nome indica controla o fluxo do programa e gere os comandos dados pelo utilizador interagindo com as outras classes.

```
while(!exit) {
    int option = Menu.MenuInicial();
    switch (option) {
        case  0 -> {...} // leave
        case  1 -> {...} // create player
        case  2 -> {...} // create team
        case  3 -> {...} // joins player to a team
        case  4 -> {...} // change player team
        case  5 -> {...} // consult player
        case  6 -> {...} // consult team
        case  7 -> {...} // calc player hab
        case  8 -> {...} // calc team hab
        case  9 -> {...} // save
        case 10 -> {...} // load
    }
}
```

c. Estado

A classe Estado é responsável por guardar o estado interno do programa. Entende-se como estado interno o seguinte: lista com todos os Jogos, e duas tabelas (Maps), uma com todas as Equipas e outra com todos os Jogadores. A utilização de Maps permite uma maior eficiência aquando da procura e acesso de equipas e/ou jogadores.

```
/* VARIAVEIS DE INSTANCIA */

    private List<Jogo> jogos;
    private Map<String, Equipa > equipas; // nome, equipa
    private Map<String, Jogador> jogadores; // nome, jogador
```

d. Jogo

A classe Jogo foi criada de forma a representar uma partida de futebol. No entanto esta apenas é usada caso um ficheiro que seja carregado contenha jogos no seu estado interno (ou então, quando este estado é guardado posteriormente). Isto acontece, uma vez que não fomos capazes de construir opções de criação de jogos e a realização dos mesmos.

```
/* VARIAVEIS DE INSTANCIA */

    private String equipaCasa;
    private String equipaFora;
    private int golosCasa;
    private int golosFora;
    private LocalDate date;
    private List<Integer> jogadoresCasa;
    private List<Integer> jogadoresFora;
    private Map<Integer, Integer> substituicoesCasa;
    private Map<Integer, Integer> substituicoesFora;
```

e. Equipa

Classe Equipa, representa uma equipa de jogadores. Nesta classe é usada uma estratégia de composição uma vez que uma equipa contém uma lista de jogadores.

```
    private static final int MAX = 20;

/* VARIAVEIS DE INSTANCIA */

    private String nome;
    private List<Jogador> jogadores;
    private int ocupacao;    // Numero atual de jogadores na equipa
```

f. Jogador

Jogador, uma classe abstrata. Esta escolha foi feita uma vez que será útil nas classes seguintes. Nesta classe e nas próximas é utilizada a relação de herança e de especialização, o que originou a criação de métodos abstratos.

```
/* VARIAVEIS DE INSTANCIA */

    // Características
    private String nome;
    private int num; // Numero da camisola
    private int velocidade, resistencia, destreza, impulsao, cabeca, remate, passe;

    // Historial
    private List<String> equipas; // Equipas por onde já passou
```

```
/* METODOS */

    public abstract Jogador clone();
    public abstract int calculaHabilidade();
    public abstract String toFile();
```

I. Avancado
II. Lateral
III. Medio
IV. Defesa
V. Guarda-Redes

Estas cinco classes “dividem” uma instância de Jogador em vários tipos, formando assim a relação de hierarquia anteriormente mencionada. Estas classes têm como informação especial, fatores que influenciam de diferentes formas o cálculo da sua habilidade, e uma característica única/individual. Dando o exemplo de um Avancado:

```java
public class Avancado extends Jogador {

    // Fatores que influenciam a habilidade de um jogador (avançado)
    private static final double velFactor = 0.6;
    private static final double resFactor = 0.8;
    private static final double desFactor = 0.6;
    private static final double impFactor = 0.5;
    private static final double cabecaFactor = 1.4;
    private static final double remateFactor = 2.4;
    private static final double passeFactor = 0.7;

    // Característica única dos avançados
    private int precisao;
```

g. Menu

A classe Menu, tem como função tratar da representação visual de menus, mensagens de erro/sucesso, entre outros. Resumindo, tem como objetivo mostrar o conteúdo relevante ao utilizador de forma que este tome decisões e escolha opções, que são recebidas pelo Controller.

h. LinhaIncorretaException

Esta é uma classe de Exceção que realiza o tratamento de erros, neste caso, aquando da má formatação de um ficheiro. É utilizada no método que trata da leitura de um ficheiro em modo texto.

```
public class LinhaIncorretaException extends Exception{
    public LinhaIncorretaException(String s) { super(s); }
}
```

Estrutura

Este projeto foi desenvolvido segundo a estrutura Model View Controller (MVC), e por isso, encontra-se dividido em três principais camadas, independentes umas das outras:

- Camada de dados e algoritmos, designada MODEL;
- Camada visual de interação com o utilizador, como por exemplo, menus, listas de resultados, etc, designado VIEW;
- Camada responsável pela ordem de execução correta da aplicação em função dos desejos do utilizador, designada CONTROLLER;

Model: constituído pelas classes Estado, Jogo, Equipa, Jogador, Avancado, Lateral, Medio, Defesa e Guarda-Redes.

View: constituído apenas pela classe Menu.

Controller: constituído pelas classes FootballManager e Controller.

Toda a construção do projeto foi pensada de forma a respeitar as normas de programação orientada aos objetos, nomeadamente o encapsulamento, a abstração de implementação assim como a capacidade de a aplicação evoluir de forma controlada, o que esperamos ter realizado com sucesso.

Ilustração

A nossa aplicação é capaz de responder aos requisitos mínimos e básicos requisitados no enunciado do trabalho prático, são estes: guardar/carregar o estado para/de ficheiro, tanto em modo texto como binário, criar e consultar jogadores e equipas assim como calcular o valor das suas habilidades, associar jogadores que não tenham equipas a uma e também alterar jogadores de equipa registando no seu historial a equipa que abandonou.

Menu inicial:

```
|-------------------------------------------|
|-----------FOOTBALL MANAGER MENU-----------|
|-------------------------------------------|

 1) Criar jogador.
 2) Criar equipa.
 3) Associar jogador a uma equipa.
 4) Mudar jogador de equipa.
 5) Consultar jogador.
 6) Consultar equipa.
 7) Calcular habilidade de um jogador.
 8) Calcular habilidade global de uma equipa.
 9) Guardar estado atual do jogo.
10) Carregar estado de um ficheiro.
 0) Sair.

Selecione uma opção:
```

Criação de um Jogador:

```
|-------------------------------------------|
|-------------BUILD PLAYER MENU-------------|

1) Guarda-Redes.
2) Defesa.
3) Médio.
4) Lateral.
5) Avançado.

Escolha um tipo de jogador: 5
Nome do jogador: Duarte Moreira
Número da camisola do jogador: 13
Parâmetro velocidade do jogador (1-100): 70
Parâmetro resistência do jogador (1-100): 45
Parâmetro destreza do jogador (1-100): 55
Parâmetro impulsão do jogador (1-100): 30
Parâmetro jogo de cabeça do jogador (1-100): 20
Parâmetro remate do jogador (1-100): 95
Parâmetro capacidade de passe do jogador (1-100): 60
Parâmetro precisão do jogador (1-100): 75
Jogador Duarte Moreira criado com sucesso.
```

Criação de uma Equipa:

```
|-----------------------------------------|
|--------------BUILD TEAM MENU--------------|

Nome da equipa: MIEI 2°Ano
Equipa MIEI 2°Ano criada com sucesso.
```

Associação de um Jogador “livre” a uma Equipa com lotação menor que 20:

```
Lista de jogadores:
Duarte Moreira
Lucas Carvalho

Nome do jogador: Duarte Moreira

Lista de equipas:
Grupo102 POO
MIEI 2°Ano

Nome da equipa: Grupo102 POO
Jogador adicionado com sucesso.
```

Mudança da Equipa de um Jogador:

```
Lista de jogadores:
Duarte Moreira
Lucas Carvalho

Nome do jogador: Lucas Carvalho

Lista de equipas:
Grupo102 POO

Nome da equipa: Grupo102 POO
Mudança executada com sucesso.
```

Consulta de um Jogador:

```
Lista de jogadores:
Lucas Carvalho
Duarte Moreira

Nome do jogador: Lucas Carvalho
 N°Camisola = 77; Nome = Lucas Carvalho; Habilidade = 52; Historial de equipas: [MIEI 2°Ano]
```

Consulta de uma Equipa:

```
Lista de equipas:
Grupo102 POO
MIEI 2°Ano

Nome da equipa: Grupo102 POO
| Equipa Grupo102 POO |
 Habilidade Global = 57
 Ocupação = 2
 Lista de jogadores: [Duarte Moreira, Lucas Carvalho]
```

Cálculo da habilidade de um Jogador:

```
Lista de jogadores:
Lucas Carvalho
Duarte Moreira

Nome do jogador: Lucas Carvalho
Lucas Carvalho tem uma habilidade de 52
```

Cálculo da habilidade global de uma Equipa:

```
Lista de equipas:
Grupo102 POO
MIEI 2°Ano

Nome da equipa: Grupo102 POO
Grupo102 POO tem uma habilidade global de 57
```

Guardar estado do jogo:

```
Selecione uma opção: 9
1) Modo binário.
2) Modo texto.
Escolha uma opção: 2
Insira o nome do ficheiro: ilustracao
Estado guardado com sucesso para o ficheiro ilustracao.
```

```
1  Equipa:Grupo102 POO
2  Avancado:Duarte Moreira,13,70,45,55,30,20,95,60,75
3  Medio:Lucas Carvalho,77,23,65,55,87,93,43,54,42
4  Equipa:MIEI 2ºAno
5
```

Conclusão

Como jeito de conclusão, tendo em conta que não conseguimos acabar todos os objetivos do programa (calcular o resultado e efetuar a simulação de um jogo), sabemos que o nosso projeto podia estar mais completo, mas, no entanto, podemos afirmar que os requisitos básicos aqui alcançados estão bem conseguidos.

Aproveitamos para salientar o facto de que é necessário ter uma boa noção dos conceitos composição, agregação, herança, encapsulamento, entres outros, para conseguir respeitar as suas regras e assim construir um programa capaz de evoluir. Cremos que este tenha sido uns dos principais objetivos e desafios do projeto.

Diagrama de Classes

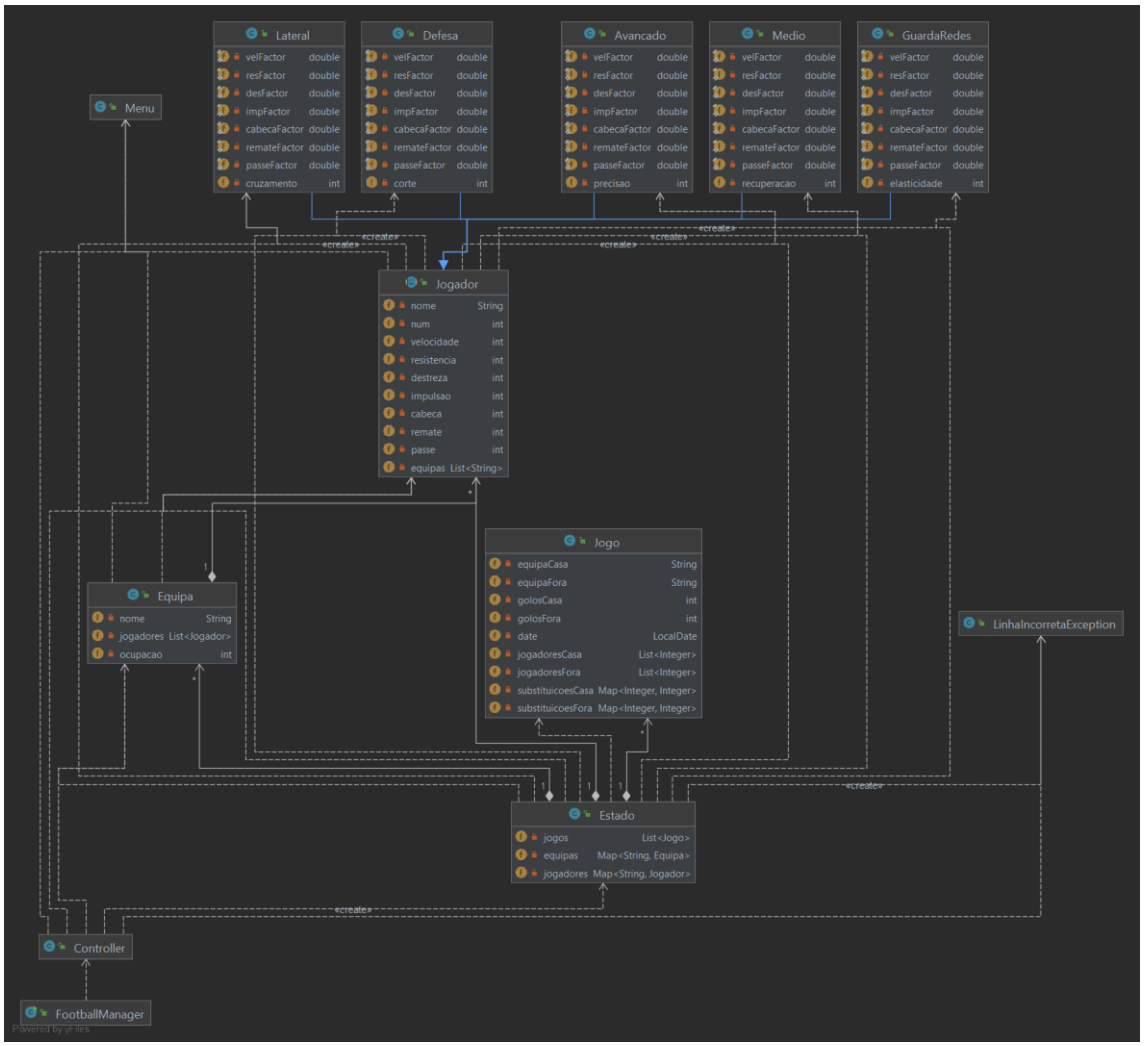

Figura 1: Diagrama de classes do programa, by **Intellij**