



# Universidade do Minho

Mestrado Integrado em Engenharia Informática

## Programação Orientada aos Objetos

TP - Football Manager

Grupo 65



Daniela Fernandes  
(A73768)



João Vale  
(A93219)



Pedro Freitas  
(A84164)

12 de junho de 2021

## Resumo

Este documento diz respeito ao projeto desenvolvido em *Java* para a unidade curricular de Programação Orientada aos Objectos, do curso de Engenharia Informática.

Para este trabalho prático foi-nos pedida a elaboração de um programa que simule um jogo, chamado *Football Manager*.

Ao longo deste relatório vamos apresentar todas as decisões que tomamos para os Requisitos base da gestão da Aplicação e para a Calcular o resultado de um jogo.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Descrição do Problema</b>	<b>3</b>
<b>3</b>	<b>Resolução do Problema</b>	<b>4</b>
3.1	Diagrama de Classes . . . . .	4
3.2	Classes . . . . .	5
3.2.1	Jogador . . . . .	5
3.2.2	Defesa . . . . .	5
3.2.3	Lateral . . . . .	6
3.2.4	Avançado . . . . .	6
3.2.5	Médio . . . . .	6
3.2.6	Guarda-Redes . . . . .	7
3.2.7	Equipa . . . . .	7
3.2.8	Jogo . . . . .	7
3.2.9	Log . . . . .	8
3.2.10	Menu . . . . .	8
3.2.11	FM . . . . .	9
<b>4</b>	<b>Considerações</b>	<b>10</b>
<b>5</b>	<b>Manual de utilização</b>	<b>11</b>
<b>6</b>	<b>Conclusão</b>	<b>12</b>

# 1 Introdução

Para este trabalho prático foi-nos pedido a elaboração de um sistema que simule as características de um jogo semelhante ao *Football Manager*.

No qual podemos visualizar os jogadores, as equipas e os jogos e também criar cada um deles. Para além disso, associar um jogador a uma equipa, trocar o jogador de equipa e criar um jogo entre duas equipas.

Na criação do jogo podemos visualizar qual foi a equipa que ganhou apartir da habilidade total de cada uma das equipas.

# 2 Descrição do Problema

Tal como referimos anteriormente, neste trabalho prático pretendemos simular um jogo semelhante ao *Football Manager*. Esta aplicação tem como principal objetivo suportar todas as funcionalidades de gestão das entidades jogadores, jogos e equipas e também calcular o resultado de um jogo.

O programa deverá cumprir os seguintes requisitos básicos:

- Carregar e gravar o estado de um programa para um ficheiro;
- Carregar um ficheiro texto;
- Criar jogadores ;
- Criar equipas;
- Associar jogadores a equipas;
- Consultar jogadores;
- Consultar equipas;
- Calcular a habilidade de cada um dos jogadores;
- Calcular a habilidade de uma equipa;
- Mudar jogador de equipa e adicionar a equipa onde estava a um histórico;
- Criação de um jogo entre duas equipas;
- Na criação do jogo, definir titulares, substituições e data;
- Em função das equipas calcular o resultado de um jogo.

Para além destes, foram-nos propostos outros requisitos relativos à simulação de um jogo que não conseguimos implementar.

### 3 Resolução do Problema

## 3.2 Classes

### 3.2.1 Jogador

A superclasse *Jogador* foi implementada com o intuito de conter a informação comum aos cinco atores do sistema (defesa, lateral, avançado, médio e guarda-redes), o que permitiu evitar a repetição das mesmas variáveis nas cinco classes referidas de seguida.

Para além disso, vai ter uma variável onde vai ser a habilidade que vai ser calculada para cada tipo de jogador.

```
1 public class Jogador implements Serializable {  
    private int id;  
3    private int numeroCamisola;  
    private String nome;  
5    private int impulsao;  
    private int velocidade;  
7    private int resistencia;  
    private int destreza;  
9    private int jogocabeca;  
    private int remate;  
11    private int capacidade_passe;  
    private List<Equipa> historicoEquipas;  
13    private int habilidade;  
    ...  
15 }
```

### 3.2.2 Defesa

Para além dos dados definidos na classe *Jogador*, o **defesa** contém uma informação específica relativa somente a si e, portanto, foi necessário a criação de uma nova subclasse de *Jogador* para acrescentar essa informação.

```
1 public class Defesa extends Jogador{  
    private int posse_bola;  
3    ...  
}
```

### 3.2.3 Lateral

Para além dos dados definidos na classe *Jogador*, o **lateral** contém uma informação específica relativa somente a si e, portanto, foi necessário a criação de uma nova subclasse de *Jogador* para acrescentar essa informação.

```
public class Lateral extends Jogador{  
2     private int cruzamentos;  
    ...  
4 }
```

### 3.2.4 Avançado

Para além dos dados definidos na classe *Jogador*, o **avanzado** contém um conjunto de informações específicas relativa somente a si e, portanto, foi necessário a criação de uma nova subclasse de *Jogador* para acrescentar essa informação.

```
public class Avancado extends Jogador{  
2     public int forca;  
     public int agilidade;  
4     public int golos;  
     ...  
6 }
```

### 3.2.5 Médio

Para além dos dados definidos na classe *Jogador*, o **médio** contém uma informação específica relativa somente a si e, portanto, foi necessário a criação de uma nova subclasse de *Jogador* para acrescentar essa informação.

```
public class Medio extends Jogador{  
2     private int recuperacao_bolas;  
     ...  
4 }
```

### 3.2.6 Guarda-Redes

Para além dos dados definidos na classe *Jogador*, o **guarda-redes** contém uma informação específica relativa somente a si e, portanto, foi necessário a criação de uma nova subclasse de *Jogador* para acrescentar essa informação.

```

public class GuardaRedes extends Jogador {
2     private int elasticidade;
        ...
4 }

```

### 3.2.7 Equipa

Para além dos *Jogadores* de todas as posições definidas anteriormente, o sistema também contempla *Equipa* que é definida pelo nome da mesma e por um *Map* que contém todos os jogadores que estão definidos nessa equipa.

```

public class Equipa implements Serializable {
2     private String nome;
        private Map<Integer, Jogador> jogadores;
        ...
4 }

```

### 3.2.8 Jogo

Sendo o *Football Manager* destinado a jogos de futebol, é importante guardar a informação relativa a cada *Jogo*.

A informação vai ser relativa a jogadores, através dos seus ID's e ao nome das equipas. Além disso, vamos ter informação relativas às substituições que se decidam fazer.

```

1 public class Jogo {
        private String equipaCasa;
3     private String equipaFora;
        private int golosCasa;
5     private int golosFora;
        private LocalDate date;
7     private List<Integer> jogadoresCasa;
        private List<Integer> jogadoresFora;
9     Map<Integer, Integer> substitucoesCasa = new HashMap<>();
        Map<Integer, Integer> substitucoesFora = new HashMap<>();
11 }

```



### 3.2.9 Log

Tal como em todas as aplicações, tem de existir alguma estrutura responsável pelo armazenamento de informação, de forma a respeitar o encapsulamento, apenas a classe que contém a informação pode aceder a ela.

Nesta classe também vão ser métodos responsáveis por qualquer alteração.

A classe *Log* contém um *Map* relativo aos jogadores, um *Map* relativo aos jogadores e uma lista relativa aos *Jogos*.

```

1 public class Log {
    private Map<String, Equipa> equipas = new HashMap<>();
3    private Map<Integer, Jogador> jogadores = new HashMap<>();
    private List<Jogo> jogos = new ArrayList<>();
5    ...
}

```

Como apenas esta classe pode aceder à estrutura, é nela que estão definidos os métodos responsáveis por gravar e carregar a informação do ficheiro *logsV2.txt*, permitindo salvar o estado da aplicação. Estes métodos serão utilizados na classe *FM*.

Além disto, vão ser definidos os seguintes métodos que vão ser utilizados para os requisitos definidos anteriormente:

- adicionaJogadorEquipa : método que vai adicionar um jogador ao sistema;
- adicionaEquipa : método que vai adicionar uma nova equipa ao sistema;
- eliminaJogadorEquipa : método que vai eliminar um determinado jogador de uma equipa;
- mudaJ : método que vai inserir um determinado jogador a uma equipa;
- adicionaJogo : método que vai adicionar um novo jogo ao sistema.

### 3.2.10 Menu

Esta é a classe que representa um menu em modo texto. Este é um fator necessário para permitir a comunicação entre a aplicação e o utilizador.

```

public class Menu {
2    private List<String> opcoes;
    private int op;
4    ...
}

```

### 3.2.11 FM

Tal como o nome indica, FM é a base de todo este projeto e é nela que estão definidos os principais métodos. Esta é a responsável por permitir o funcionamento de todo o sistema. É aqui que os menus são carregados, os pedidos (sobre os jogadores, equipas e jogos) são submetidos e os resultados vão ser apresentados.

```
1 public class FM {  
    private Menu menuPrincipal;  
3     private Log log;  
    ...  
5 }
```

Como é o FM que lida diretamente com o input submetido pelo utilizador, é nesta classe que são geridas os métodos definidos em Log.

## 4 Considerações

Ao desenvolver este projeto deparámo-nos com um conceito ambíguo a *habilidade* e o *números de golos* de cada uma das equipas num jogo.

Para calcular a *habilidade* tivemos que determinar qual seria a expressão utilizada para cada um dos jogadores.

- **Defesa:**

$$\text{habilidade} = \text{posseBola} * 0.3 + \text{velocidade} * 0.15 + \text{resistência} * 0.15 + \text{destreza} * 0.1 + \text{impulsão} * 0.1 + \text{cabeça} * 0.05 + \text{remate} * 0.05 + \text{passe} * 0.1;$$

- **Lateral**

$$\text{habilidade} = \text{cruzamentos} * 0.2 + \text{velocidade} * 0.15 + \text{resistência} * 0.15 + \text{destreza} * 0.1 + \text{impulsão} * 0.125 + \text{cabeça} * 0.1 + \text{remate} * 0.05 + \text{passe} * 0.125;$$

- **Avançado**

$$\text{habilidade} = \text{força} * 0.1 + \text{golos} * 0.1 + \text{agilidade} * 0.1 + \text{velocidade} * 0.1 + \text{resistência} * 0.1 + \text{destreza} * 0.1 + \text{impulsão} * 0.1 + \text{cabeça} * 0.1 + \text{remate} * 0.1 + \text{passe} * 0.1;$$

- **Médio**

$$\text{hailidade} = \text{recuperacao bolas} * 0.2 + \text{velocidade} * 0.1 + \text{resistência} * 0.1 + \text{destreza} * 0.125 + \text{impulsão} * 0.1 + \text{cabeça} * 0.125 + \text{remate} * 0.05 + \text{passe} * 0.20;$$

- **Guarda-Redes**

$$\text{habilidade} = \text{elasticidade} * 0.3 + \text{velocidade} * 0.075 + \text{resistência} * 0.075 + \text{destreza} * 0.15 + \text{impulsão} * 0.15 + \text{cabeça} * 0.075 + \text{remate} * 0.075 + \text{passe} * 0.1;$$

Para além da habilidade, para calcular o número de golos marcado por uma determinada equipa, utilizamos a habilidade de todos os jogadores de todos os titulares.

- $\text{golos} = \text{habilidade de todos os titulares} \% 5.$

## 5 Manual de utilização

Nesta secção, iremos dar uma breve explicação de como utilizar esta aplicação, de modo a facilitar o uso da mesma.

```
*** MENU ***
1 - Pesquisa jogador
2 - Pesquisa equipa
3 - Ver equipas
4 - Ver Jogadores
5 - Ver jogo
6 - Mudar jogador de equipa
7 - Novo Jogador
8 - Nova Equipa
9 - Criar jogo entre duas equipas
10 - Guarda
0 - Sair
Opção:
```

Figura 2: Menu principal.

- **Pesquisa Jogador**

Para este passamos o ID de um jogador para ver as informações sobre ele.

- **Pesquisa Equipa**

Para este passamos o Nome da equipa para ver todos os jogadores que fazem parte dessa equipa.

- **Ver Equipas**

Este vai imprimir todas as equipas e os jogadores de cada uma que está guardado na estrutura de dados *log*.

- **Ver Jogadores**

Este vai imprimir todos os jogadores que estão guardados na estrutura de dados *log*.

- **Ver Jogos**

Este vai imprimir todos os jogos que estão guardados na estrutura de dados *log*.

- **Mudar jogador de equipa**

Para este temos que passar o ID do jogador que queremos mudar de equipa, a equipa onde estava e qual a equipa para qual queremos que vá. Vai ser adicionado no histórico a equipa que estava, podemos ver isso utilizando a pesquisa do jogador.

- **Novo Jogador**

Para este temos que passar no nome do jogador, o número de camisola e a equipa a qual o queremos adicionar. Para vermos isto, podemos utilizar a pesquisa do jogador e de seguida a pesquisa de uma equipa.

- **Nova Equipa**

Para este temos que passar o nome da equipa. Para vermos, podemos pesquisar pelas equipas ou pela equipa.

- **Criar jogo entre duas equipas**

Para criar um jogo entre duas equipas temos que passar o nome das duas equipas, escolher 11 jogadores titulares que fazem parte da equipa, escolher as substituições que queremos efetuar e a data no qual vai ocorrer o jogo. Com a escolha dos titulares, vai haver um cálculo dos golos como explicamos na secção anterior.

- **Guarda**

Este serve para guardar todas as alterações efetuadas, criação do jogador, da equipa e do jogo e a mudança de um jogador de uma determinada equipa.

## 6 Conclusão

Para a realização deste projeto, foi utilizada a linguagem de programação Java, onde a grande maioria dos conhecimentos aplicados no projeto foram adquiridos nesta unidade curricular.

No que se trata de funcionalidades, a nossa aplicação cumpre com os requisitos básicos estipulados que foram referidos no início deste documento.

Em termos de código, o nosso projeto deveria estar melhor estruturado e organizado, também poderíamos ter implementado mais controlos de erros que não implementamos devido à falta de tempo.

Para trabalho futuro, poderíamos tentar simular um jogo de acordo com os requisitos estipulados.

De forma geral, consideramos ter tido um desempenho positivo na realização deste projeto, o que não implica que não existam aspetos que gostaríamos de ter feito de forma melhorada. Mas todos estes pormenores devem agora ser tomados em conta em trabalhos futuros.