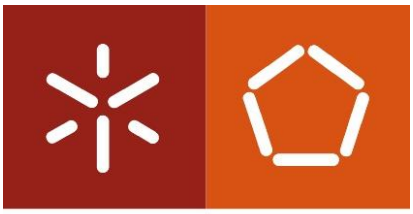


Universidade do Minho

Mestrado Integrado em Engenharia Informática

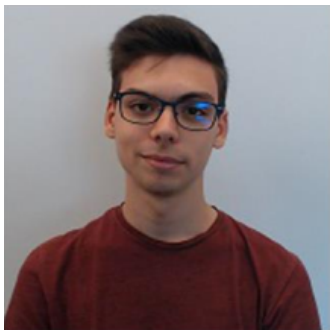
POO - Trabalho Prático

Grupo 05

João Barbosa (a93270) Simão Cunha (a93262)
Tiago Silva (a93277)

Ano Letivo 2020/2021





Conteúdo

Capítulo 1

Introdução e principais desafios

Este projeto consistiu em criar um sistema de gestão e simulação de equipas de um determinado desporto (neste caso futebol) muito semelhante ao jogo *Football Manager*, de forma a aplicarmos os vários conhecimentos adquiridos nas aulas ao longo do semestre.

Tivemos alguns obstáculos para a concessão deste projeto, como a falta de tempo para conseguir uma aplicação que respeitasse melhor os princípios de programação orientada a objetos.

Capítulo 2

Classes

2.1 Main

Classe responsável pelo arranque do programa. Para isto, o método `main` invoca o método `run` da classe `Controlo`.

2.2 Controlo

```
private ControloDados cd;
private Scanner scan;
private final String[] menuPrincipal;
private final String[] menuCarregarDados;
private final String[] menuSimulacao;
private final String[] menuEditarEquipa;
private final String[] menuEditarJogador;
private final String[] menuCriarDados;
private final String[] menuCriarJogador;
private final String[] menuPosicao;
private final String[] menuEscolheTatica;
```

`Controlo` é a classe responsável pelo controlo da aplicação. Associa um menu a uma funcionalidade do programa. Por exemplo, mostra no ecrã a simulação de um jogo de futebol, recorrendo a métodos da classe `ExecutaPartida` e com mensagens pré-concebidas para vários momentos do jogo.

Neste menu temos as seguintes opções:

1. *Simular uma partida de futebol*: é possível escolher equipas, definir o seu onze inicial, esquema tático, efetuar um jogo, etc

2. *Manipular dados*: é possível carregar, criar, ver/editar, gravar e fazer reset a dados

2.3 ControloDados

```
private Map<String,EquipaFutebol>equipas;//nome + equipa
private Map<String,List<PartidaFutebol>>partidas;//Nome + Lista dos jogos
```

`Controlo de dados` é a classe de controlo para a execução de partidas de futebol, que inclui uma `HashMap` de equipas de futebol que são identificadas pelo seu nome e um `HashMap` de partidas de futebol onde é possivel associar o nome de uma equipa à lista de todos os seus jogos efetuados.

2.4 ViewJogo

```
private static Scanner is;
private boolean continuacao;
private List<String> opcoes;
private List<PreCondition> disponivel;
private List<Handler> handlers;
```

A classe `ViewJogo` gere o fluxo do programa através da gestão de menus. Esta é a classe responsável por gerir o menu inicial, disponível para todas as entidades como *JogoFutebol* ou *EquipaFutebol*, que interage com outras classes controladoras que gerem os menus de cada entidade em particular.

2.5 Jogador

```
private int numero;
private String nome;
private String posicao;
private List <String> historial;
private Atributos atributos;
```

Esta é uma das classes essenciais da aplicação. A classe `jogador` representa um jogador de futebol, onde são armazenados o número da camisola, o seu nome, a sua posição em campo, o historial dos clubes por onde passou e os seus atributos (como a capacidade de remate, velocidade, passe,...). Existem **5** tipos de jogador: guarda-redes, defesa, lateral, médio e avançado.

2.6 CompPosicao

A classe `CompPosicao` implementa um *Comparator* para a classe `Jogador`. Este `Comparator` compara dois jogadores através dos números da camisola.

2.7 Atributos

```
private int velocidade;
private int resistencia;
private int destreza;
private int impulsao;
```

```
private int jogoDeCabeca;
private int remate;
private int controloDePasse;
```

A classe `Atributos` é abstrata, pois os valores atribuídos para cada parâmetro variam conforme a posição do jogador. Como será visível mais à frente, decidimos adicionar mais atributos a cada tipo de jogador, mais própriamente em classe hereditárias.

2.7.1 AtributosGR

```
private int elasticidade;
private int reflexos;
```

Esta classe herda atributos da classe `Atributos` e decidimos acrescentar mais 2 atributos: a elasticidade e os reflexos.

2.7.2 AtributosLateral

```
private int precisaoCruzamentos;
private int drible;
```

Esta classe herda atributos da classe `Atributos` e decidimos acrescentar mais 2 atributos: a precisão em efetuar cruzamentos e a capacidade de driblar um adversário.

2.7.3 AtributosDefesa

```
private int posicionamentoDefensivo;
private int cortes;
```

Esta classe herda atributos da classe `Atributos` e decidimos acrescentar mais 2 atributos: o posicionamento defensivo e a capacidade de cortar a bola.

2.7.4 AtributosMedio

```
private int recuperacaoDeBolas;
private int visaoDeJogo;
```

Esta classe herda atributos da classe `Atributos` e decidimos acrescentar mais 2 atributos: a capacidade de recuperação da bola e a visão de jogo.

2.7.5 AtributosAvancado

```
private int penaltis;
private int desmarcacao;
```

Esta classe herda atributos da classe `Atributos` e decidimos acrescentar mais 2 atributos: a capacidade de marcação da marca dos onze metros e a desmarcação para se isolar dos adversários.

2.8 Plantel

```
private Map<Integer,Jogador> titulares;
private Map<Integer,Jogador> suplentes;
private int nJogadoresNoPlantel;
private Tatica tatica;
```

A classe `Plantel` representa um plantel de futebol, com um *map* de jogadores no onze inicial, um *map* de jogadores no banco, o número total de jogadores num plantel e o esquema tático adotado pela equipa.

2.9 Tatica

```
private int nGR;
private int nDF;
private int nLT;
private int nMD;
private int nPL;
```

A classe `Tatica` define o esquema tático de uma equipa. Para tal, será necessário saber o número de guarda-redes (que será sempre 1), o número de defesas, o número de laterais, o número de médios e o número de avançados

2.10 EquipaFutebol

```
private String nome;
private Plantel plantel;
```

O conceito da classe `EquipaFutebol` pode ser confundido com o conceito da classe `Plantel`. Uma `Plantel` é dividido em vários parâmetros para serem mais facilmente acedidos pelo utilizado. Já uma `EquipaFutebol` engloba todos estes parâmetros num objeto `Plantel`, incluindo o nome da equipa.

2.11 PartidaFutebol

```
    private double tempo;
    private int golosVisitante;
    private int golosVisitado;
    private int substituicoesVisitante, substituicoesVisitados;
    private int[][] subsVisitante, subsVisitada;
    private EquipaFutebol equipaVisitante, equipaVisitada;
    private LocalDate data;
};
```

A classe `PartidaFutebol` representa uma partida de futebol. São incluídos o tempo de jogo, o número de golos da equipa visitante, o número de golos da equipa da casa, o número de substituições que a equipa visitante pode efetuar, o número de substituições da equipa da casa que ainda

pode fazer, um *array* de substituições de uma equipa (visitante ou visitada) onde cada jogador é representado pelo seu número de camisola, os equipas que vão jogar e a data da realização do jogo.

2.12 ExecutaPartida

```
private PartidaFutebol partida;
private Jogador jogadorAtual;
private Boolean casa;
private Boolean comecou;
private Random random;
private static final double acaoRapida = 0.5;
private static final double acaoMedia = 1.0;
```

A classe `ExecutaPartida` é responsável por, tal como diz o nome, executar uma partida de futebol. É constituído por uma partida de futebol, pelo jogador que possui atualmente a bola, um booleano que verifica se o jogador é da equipa adversária, um booleano que verifica se um jogo já começou, um objeto `Random` para gerar números aleatórios (por exemplo) e duas constantes que simbolizam uma *ação rápida* (por exemplo, tentar rematar a bola) e uma *ação média* (por exemplo, marcar golo).

Capítulo 3

Estrutura do projeto

O nosso projeto segue a estrutura *Model View Controller* (MVC), estando por isso organizado em três camadas:

- A camada de dados (o **"Model"**) é composta pelas classes Jogador, Atributos, AtributosGR, AtributosDefesa, AributosLateral, AtributosMedio, AtributosAvancado, EquipaFutebol, Plantel, Tatica e PartidaFutebol e pela interface CompPosicao.
- A camada de interação com o utilizador (a **"View"**, ou apresentação) é composta unicamente pela classe ViewJogo, embora existam algumas mensagens impressas para o *stdout* noutras classes de outras camadas.
- A camada de controlo do fluxo do programa (o **"Controller**) é composta pela classe ExecutaPartida, ControloDados. Controlo e pelo Main.

Todo este projeto cumpriu um princípio principal da programação orientada aos objetos: o *encapsulamento*. São exemplos notórios deste cumprimento os *getters* e *setters* de todas as classes tanto para objetos imutáveis (por exemplo, *string* ou *integer*) como para objetos mutáveis (por exemplo *Jogador* ou *Equipa*).

Capítulo 4

Conclusão

A nível geral, e tendo em conta o que foi explicado nos capítulos anteriores, podemos afirmar que temos um projeto bem conseguido, apesar de não estar exatamente igual ao que idealizávamos. Acreditamos que respondemos de forma correta à simulação de uma partida de futebol, que foi um dos pontos mais trabalhosos do nosso projeto. Além disso, é possível um crescimento controlado da nossa aplicação, uma vez que é possível adicionar mais desportos no diretório *Desporto* do nosso projeto.

Apêndice A

Diagrama de Classes

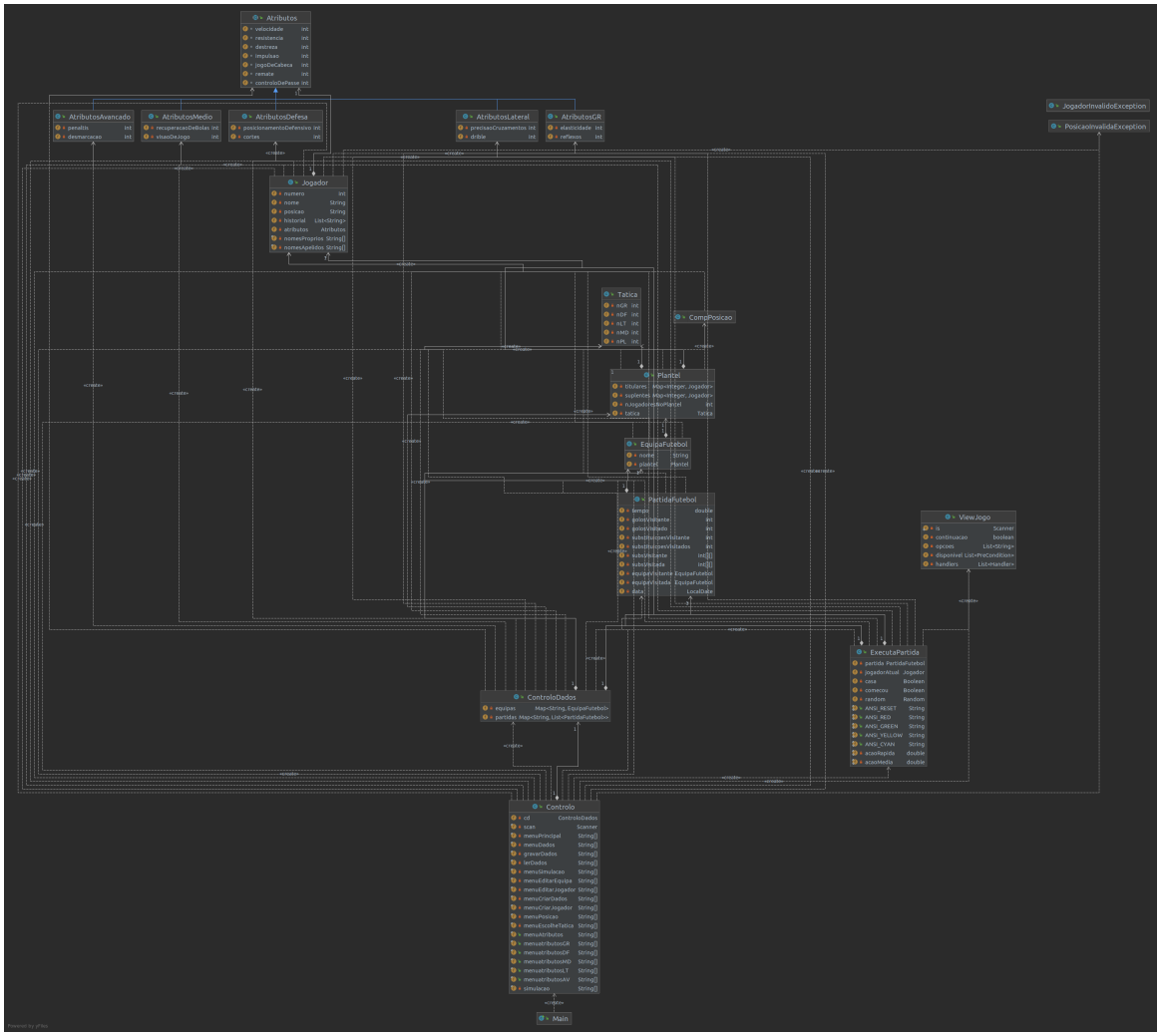


Figura A.1: Diagrama de classes do programa, gerado pelo *IntelliJ*