UNIVERSIDADE DO MINHO

Mestrado Integrado em Engenharia Informática

Programação Orientada Aos Objetos

Grupo 3

Ana Filipa Ribeiro Murta (a93284)

Ana Paula Oliveira Henriques (a93268)

Carlos Daniel da Silva Santos (a85617)

Ano Letivo 2020/2021

Índice

1. Introdução e principais desafios

Este projeto consistiu no desenvolvimento de um jogo de futebol, semelhante ao conhecido jogo *Football Manager*, na linguagem de programação Java de forma a pôr em prática os conhecimentos aprendidos ao longo do semestre.

O programa desenvolvido consiste, portanto, numa partida entre duas equipas, uma identificada pela equipa que joga em casa e a outra pela equipa que joga fora, que foram escolhidas a partir de um ficheiro fornecido pelos docentes ou, então, criada pelo usuário.

Um jogo tem duração de 90 minutos em que, aos 45 minutos, é feito um intervalo para se efetuar substituições. Ao longo do jogo, serão feitas jogadas que serão determinadas consoante os números aleatórios gerados pela classe `Random` do pacote `java.util`.

O maior desafio enfrentado é capaz de ter sido as restrições dos jogadores consoante as suas posições, uma vez que era solicitado haver jogadores que jogam numa posição central e jogadores que jogam na lateral. Deste modo, consoante o seu tipo, é permitido ao jogador jogar só em certas posições no jogo. Por exemplo, um jogador lateral não pode jogar como defesa central e, conforme o modelo tático, um médio só pode jogar como extremo ou médio centro.

IMPORTANTE: todas as classes estão associadas por composição.

2. Arquitetura de classes e aplicação desenvolvida

2.1. Main

Esta classe é responsável por arrancar o programa, começando por imprimir no terminal o menu principal e, a partir daí, o usuário escolherá o que pretende fazer a seguir tendo em conta as opções que lhe são apresentadas.

2.2. Jogador

A cada tipo de jogador é associado um inteiro de forma a podermos pegar diretamente num objeto de tipo `Jogador` e saber se é um avançado, um médio, um lateral, um defesa ou um guarda-redes:

```
private static final int AVANCADO = 1;
private static final int MEDIO = 2;
private static final int LATERAL = 3;
private static final int DEFESA = 4;
private static final int GR = 5;
```

Um jogador tem um conjunto de características-base (cujos valores estão entre 0 a 100):

```
private String nome;
private int nr_camisola;
private int velocidade;
private int resistencia;
private int destreza;
private int impulsao;
private int jogoCabeca;
private int remate;
private int capPasse;
private List<String> historico;
private int tipoJogador;
```

2.3. Subclasses de Jogador

A superclasse `Jogador` possui as seguintes subclasses:

`Avancado`:

```
private int drible;                -- característica extra
```

`Medio`:

```
private int capRecuperacao;
private int dominioBola;           -- característica extra
```

`Lateral`:

```
private int capCruzamento;
```

`Defesa`:

```
private int desarme;                -- característica extra
```

`GuardaRedes`:

```
private int elasticidade;
private int lancamento;             -- característica extra
```

Para além das características-base que nos foram fornecidas, foi-nos, também, proposto acrescentar características extra consoante o tipo do jogador. As únicas características extra que adicionámos foram: o lançamento de bola do guarda redes (com o pé ou com a mão); o domínio da bola do médio, referente à sua capacidade de controlar a bola; o desarme do defesa, que consiste em desarmar uma equipa; e o drible do avançado. A estas características extras foram gerados números aleatórios entre 0 a 100 pela classe `Random` do pacote `java.util`.

Cada jogador possui um valor de habilidade. Ora, os métodos que calculam a habilidade dum jogador encontram-se em cada uma das superclasses visto que este cálculo vai depender das características específicas de cada tipo de jogador. Quanto mais elevado for este valor, melhor é o desempenho esperado do jogador.

2.4. CriarJogador

Esta classe destina-se a criar um jogador com a intervenção do usuário, ou seja, é o usuário que escolhe que tipo de jogador é que pretende criar e lhe atribui um nome e valores às suas características. Depois disto, é perguntado ao utilizador a que equipa é que quer associar este novo jogador, inserindo-o nessa equipa caso esta exista.

2.5. Equipa

Esta classe está encarregue de trabalhar com tudo relacionado com as equipas.

```
private int nr_equipa;
private int nr_tatica;
private String nome;
private Map<Integer, Jogador> jogadores;
private Map<Integer, Integer> titulares;
private Map<Integer, Integer> suplentes;
```

No map jogadores, cada objeto jogador (value) está associado ao número da sua camisola de cada jogador (key). Já no map titulares, cada posição no jogo do titular (value) está associado ao número da sua camisola (key). O mesmo pensamento foi aplicado noo map suplentes, em que cada posição no jogo do suplente (value) está associado ao número da sua camisola (key).

Estes foram os inteiros associados a cada posição no jogo:

1 guarda redes
2 defesa central
3 defesa lateral
4 médio / médio centro
5 extremos
6 avançado / avançado centro
7 suplente

Nesta classe, temos dois métodos importantes, que foram os mais desafiantes. Um deles é o que cria o map dos titulares e o map dos suplentes de acordo com o modelo de tática da equipa em que estão inseridos. O outro é o que trata das substituições entre titulares e suplentes de uma equipa.

A `criaTitularesSuplentes()` é responsável por dividir os jogadores da equipa em titulares e suplentes. Para isto, a estratégia utilizada foi, primeiramente, percorrer o map de jogadores e, segundo o seu tipo, cada jogador era colocado numa lista. No final, existem, portanto, 5 listas porque existem 5 tipos de jogador. Em seguida, percorria-se cada uma das listas e distribuía-se os seus jogadores por titulares e suplentes, colocando cada um deles no map correspondente (ou no map de titulares ou no map de suplentes).

Esta distribuição era efetuada consoante o modelo de tática da equipa. Se o modelo de tática fosse 1-4-4-2, então teríamos:

1 guarda redes para a posição de guarda redes
2 defesas para a posição de defesas centrais
2 laterais para a posição de defesas laterias
4 médios — **2** para a posição de médio centro e **2** para a posição de extremos
2 avançados para a posição de avançados centro

No caso do modelo de tática 1-4-3-3 temos:

1 guarda redes para a posição de guarda redes
2 defesas para a posição de defesas centrais
2 laterais para a posição de defesas laterias
3 médios para a posição de médios centro
3 avançados — 1 para avançado centro e 2 para extremos

Assim, depois de preenchidos estes lugares, os jogadores que sobram são colocados como suplentes.

A `substituirDentroEquipa()` está encarregue de permitir uma substituição entre um titular e um suplente, sendo que só são autorizadas, no máximo, 3 substituições por equipa. As substituições admitidas são:

Um **avançado** pode jogar na posição de avançado centro e extremo
Um **médio** pode jogar na posição de médio centro e extremo
Um **lateral** pode jogar na posição defesa lateral e extremo
Um **defesa** pode jogar como defesa central e avançado centro
Um **guarda redes** pode jogar como guarda redes

2.6. CriarEquipa

Esta classe destina-se a criar uma equipa com a intervenção do usuário: é lhe pergunta o nome da nova equipa e depois é criado um objeto de tipo `Equipa` com todos os seus campos vazios, exceto o campo referente ao nome. É-lhe, ainda, dada a opção de criar os 20 jogadores para essa equipa.

2.7. EstadoJogo

```
private LocalDate data;
private Equipa equipaCasa;
private Equipa equipaFora;
private int scoreCasa;
private int scoreFora;
private List<Integer> jogadoresCasa;
private List<Integer> jogadoresFora;
private Map<Integer,Integer> substituicoesCasa;
private Map<Integer,Integer> substituicoesFora;
private int nrSubstituicoesCasa;
private int nrSubstituicoesFora;
```

Nesta classe, tal como o seu nome sugere, trabalha-se o estado do jogo, desde quem joga em casa a quem joga fora, o resultado de quem joga em casa ao resultado de quem joga fora, por aí em diante. Por outras palavras, trabalha-se tudo relacionado com as duas equipas que estão a jogar em campo numa determinada partida.

2.8. Jogo

```
private String equipaAtual;
private int gameProgress;
private EstadoJogo estado;
```

Nesta classe, cria-se o dito jogo. Temos, então, um método que verifica se um jogo ainda não começou, iniciando-o se sim e outro que verifica se a partida já terminou, apresentando, se sim, os dados referentes à pontuação final e a que equipa ganhou.

Para as jogadas, distingue-se três métodos em que a estratégia utilizada em todos foi gerar números aleatórios através da classe `Random` do pacote `java.util`:

O `iniciaJogada()` é responsável por começar uma jogada a partir de um estado. Se o número aleatório gerado estiver entre 6 a 9 (inclusive), começa-se uma jogada; caso contrário, não acontece nada. No entanto, dentro daquele intervalo, são feitas duas divisões consoante o valor da habilidade de cada equipa. Para fazer estas divisões, tem-se de calcular as hipóteses de cada equipa de ganhar o jogo. Aquela que possuir maior hipótese é a que tomará posse da bola jogada iniciada.

O `constroiJogada()` está encarregue de construir uma jogada a partir de um estado. Se o número aleatório gerado estiver entre 0 a 2 (inclusive), a equipa com posse de bola perde a bola para a equipa adversária; se estiver entre 3 a 6 (inclusive), um jogador da equipa com posse de bola passa a bola para outro jogador; se estiver entre 7 a 8 (inclusive), a equipa adversária pressiona os adversários; e, por último, se for 9, a equipa com a bola remata.

O `remate()` efetua um remate a partir de um estado. Se o número aleatório gerado estiver entre 0 a 3 (inclusive), a equipa com a bola remata e marca golo; se estiver entre 4 e 6 (inclusive), esta remata, mas o guarda-redes adversário defende; e se estiver entre 7 a 9 (inclusive), esta remata, mas falha a baliza.

2.9. JogoRegisto

Esta classe do código delimita-se a moldar um objeto capaz de guardar todas as informações necessárias de um jogo já efetuado anteriormente, daí o nome registo.

```
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;
```

Primeiramente, esta classe será usada logo no inicio do jogo para guardar o registo de Jogos existente no ficheiro logs.txt fornecido. Estes jogos, depois de passarem pelo `parse(String input)`, vão poder ser consultados no menu principal através do método `apresentarJogo()` que apresenta, por essa data em que foi efetuado o jogo, as equipas e os seus respetivos resultados.

Apesar das semelhanças com a classe EstadoJogo (no que tocas às variáveis), esta apenas remete-se a guardar o nome de cada equipa. Como as táticas e as características já foram usadas, podem ser descartadas em prol do Jogo ser guardado de forma menos pesada no objeto Data e eventualmente guardado num ficheiro através do método `saver(PrintWriter print)`.

2.10. MainMenu

A função interativa do jogo inicia-se nesta classe já que esta chama os objetos e os métodos de outras classes de modo a decidir o rumo do jogo de acordo com as escolhas tomadas pelo usuário.

```
private int option;
private Data dados;
```

No construtor, são apresentadas as opções possíveis ao utilizador, sendo estas opções as seguintes: criar uma partida, verificando sempre a validez das equipas que foram escolhidas e, se tal, criar um JogMenu; gerir o jogo e os seus dados, como, por exemplo, criar equipas ou transferir jogadores; e, também, visualizar o registo de jogos já ocorridos num formato de páginas, com a possibilidade de verificar os jogos de acordo com a sua data. Para além destas opções, temos, ainda, métodos que carregam um ficheiro ou simplesmente guardam os dados num ficheiro chamado dados.txt.

2.11. GerirMenu

Tal como o MainMenu, esta classe ofecere um conjunto de opções para o usuário escolher, mas, neste caso, com o propósito de manipular os dados do jogo.

```
private int option;
private Data dados;
```

O construtor começa por apresentar as diferentes escolhas e, de acordo com a decisão do usuário, o menu tem a habilidade de criar uma equipa (isto com a opção de criar 20 jogadores para a respetiva equipa), tal como a de criar um jogador (e as suas respetivas características), adicionando-os na base de dados. É, também, possível inspecionar as equipas, os seus planteis e os jogadores representados na base de dados.

Por fim, o GerirMenu dá também a hipótese de transferir um jogador de equipa para equipa com o método `transferirJog()`. Este método, após perguntar qual a equipa origem, a equipa destino e o jogador em causa, move o jogador, adicionado ao seu histórico a equipa origem (onde previamente estava).

2.12. JogMenu

O terceiro Menu do jogo vai ser apresentado apenas na preparação de uma partida que é planeada pelo jogador no estado.

```
private int option;
private Jogo jogo;
private EstadoJogo estado;
private Data dados;
```

A partir do estado de jogo, depois de receber a opção do utilizador, o JogMenu consegue escolher as substituições feitas no jogo antecipadamente, verificando a validez de todos os parâmetros como a existência desses jogadores e a possibilidade do seu tipo (avançado, médio, etc.) poder jogar nas posições mais específicas (defesa centro, defesa lateral, etc.).

É possível, ainda, fazer a escolha da tática usada por uma equipa através do método `escTatica()`, que faz com que os titulares sejam automaticamente atribuídos com base nas suas novas posições.

2.13. ParserMod

Nesta classe, é feita o parse dos dados sobre os jogadores, as equipas e os jogos a partir do conteúdo do ficheiro input. Os métodos desta classe foram fornecidos pelos docentes, nós apenas o adaptámos e melhorámos para aquilo que era preciso.

2.14. Saver

Nesta classe, encontra-se o método estático `save(Data dados)`, que ao contrário do `ParserMod`, vai guardar toda a informação do jogo carregada e modificada pelo utilizador no ficheiro dados.txt de modo a ser carregada no futuro.

2.15. Data

Nesta classe, será criado um objeto que guarda toda a informação do Jogo, ou seja, todas as equipas e o registo de Jogos.

```
private Map<String, Equipa> equipas;
private List<JogoRegisto> jogos;
```

Não só o objeto é utilizado para ter sempre o jogo atualizado, como também usa métodos próprios para apresentar a informação guardada em detalhe, juntamente com a habilidade de chamar métodos para dividir o plantel da equipa em titulares e suplentes, consoante a tática.

2.16. Exceptions

A exceção `EquipaJaExisteException()` é usada quando o usuário tenta criar uma equipa, mas esta já existe, ou seja, já há uma equipa com este nome.

A exceção `EquipaNaoExisteException()` é aplicada quando o utilizador escolhe uma equipa que não existe.

A exceção `EquipaNaoValidaException()` é usada quando uma equipa não tem no mínimo 16 jogadores.

A exceção `JogadorNaoExisteException()` é aplicada quando o utilizador pretende fazer uma substituição entre um titular e um suplente numa equipa. Por outras palavras, caso o titular escolhido pelo o usuário não exista na equipa é lançado este erro. O mesmo acontece com o suplente.

A exceção `JogoNaoValidoException()` é usada quando um jogo não é válido, ou seja, quando ele põe uma equipa a jogar com si própria.

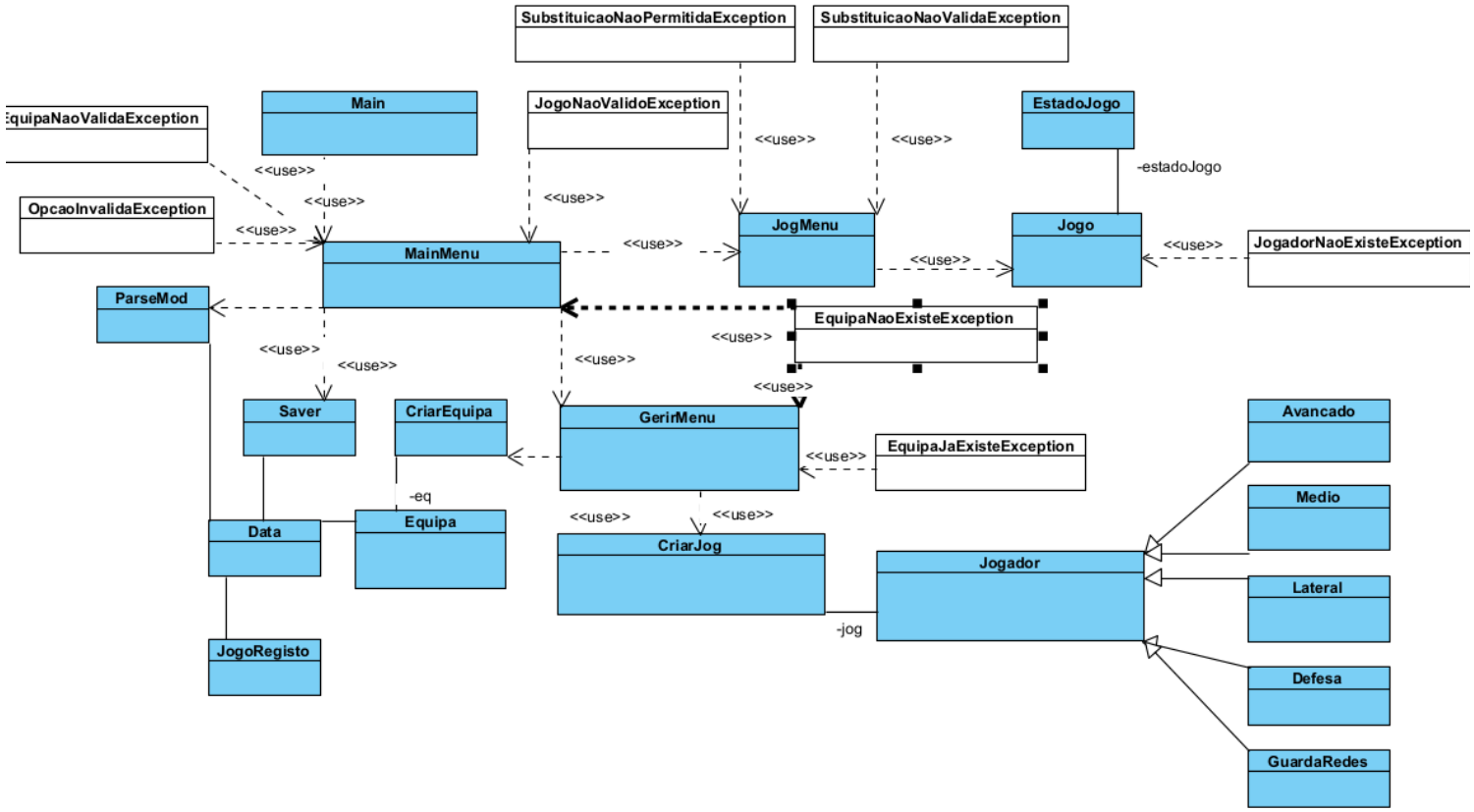
A exceção `LinhaIncorretaMod()` é lançada quando uma linha de registo no ficheiro logs.txt não está de acordo com os parâmetros.

A exceção `OpcaoInvalidaException()` é lançada quando o usuário escolhe a opção errada, por exemplo, quando se fornece três opções devidamente identificadas, mas este não seleciona nenhuma delas.

A exceção `SubstituicaoNaoPermitidaException()` é aplicada quando o usuário deseja fazer uma substituição entre um titular e um suplente numa equipa, mas o número máximo de substituições admitidas numa equipa (3) já foi atingido. Assim, não lhe é permitido fazer mais substituições, lançando um erro.

A exceção `SubstituicaoNaoValidaException()` é aplicada quando o utilizador quer, numa equipa, executar uma substituição entre um titular e um suplente, mas este realiza uma troca não válida. Isto é, o suplente escolhido não pode jogar na posição do titular selecionado; por exemplo, se o utilizador decidir substituir um guarda redes por um avançado, o erro é lançado.

3. Diagrama de classes



4. Aplicação desenvolvida

```
Seja bem-vindo!
Presse "ENTER" para começar.

A carregar dados..
Jogo carregado.


Introduza a sua escolha:
1: Criar Partida
2: Consultar e Criar Jogadores/Equipas
3: Verificar Registo de Jogos
4: Guardar Jogo
5: Carregar Jogo
6: Sair
```

Escolhendo a opção 1 inicialmente, é mostrado ao utilizador todas as equipas que existem. Em seguida, é perguntado qual a equipa que quer que jogue em casa e a que quer que jogue fora, sendo-lhe apresentado, depois desta escolha, este menu:

```
Introduza a sua escolha:
1: Começar Jogo
2: Escolher tática
3: Escolher substituições
4: Sair
```

A opção 1 permite começar o jogo sendo que é imprimido no ecrã todos os acontecimentos ocorridos durante o jogo, isto é, todas as jogadas efetuadas (remates, perdas de bolas, etc). A opção 2 permite ao usuário escolher o modelo de tática para a equipa, mostrando as duas únicas táticas possíveis. A opção 3 permite ao utilizador escolher as substituições entre titulares e suplentes, sendo-lhe mostrado todos os jogadores titulares juntamente com as suas posições e os suplentes juntamente com os seus tipos.

Escolhendo a opção 2 inicialmente, é-lhe apresentado este menu:

```
Introduza a sua escolha:
1: Criar Equipa
2: Criar Jogador
3: Visualizar Jogadores/Equipas
4: Transferir Jogadores
5: Voltar
```

A opção 1 permite criar uma equipa em que o usuário é questionado se quer criar os 20 jogadores ou não. A opção 2 limita-se a criar um jogador, questionando o usuário que tipo de jogador deseja para atribuir valores consoante as suas características. A opção 3 imprime no ecrã o plantel das equipas e, após escolher uma delas, é também imprimido todos os jogadores associados a essa equipa. A opção 4 permite o usuário transferir um jogador entre equipas, adicionando ao histórico do jogador a equipa da qual acabou de sair.

Escolhendo a opção 3 inicialmente, é mostrado o registo de todos os jogos efetuados num formato de páginas:

```
 Página 0 de 6
2021-04-09: Sporting Club Schubert 3 - 4 Wagner Athletic
2021-04-07: Sporting Club Chopin 1 - 1 Handel Athletic
2021-04-06: Beethoven F. C. 1 - 1 Sporting Club Dvorak
2021-04-05: Vivaldi F. C. 3 - 0 Beethoven F. C.
2021-04-05: Vivaldi F. C. 5 - 1 Handel Athletic
2021-04-04: Sporting Club Schubert 3 - 1 Sporting Club Prokofiev
2021-04-03: Sporting Club Chopin 2 - 2 Wagner Athletic
2021-04-02: Sporting Club Chopin 4 - 3 Sporting Club Prokofiev
2021-04-02: Bartok F. C. 3 - 2 Handel Athletic
2021-04-02: Mahler Athletic 0 - 2 Sporting Club Schubert
(0: Voltar) (2: Pag. Seguinte)
```

Escolhendo a opção 4 inicialmente, são carregados os dados dum ficheiro.

Escolhendo a opção 5 inicialmente, são guardados os dados num ficheiro chamado dados.txt.

5. Conclusão

Este projeto foi bastante enriquecedor para todos os elementos do grupo. Primeiramente porque nenhum tinha alguma vez realizado um projeto em java e, como tal, cada momento foi um momento de aprendizagem desde correção de erros à escolha do modelo de associação que melhor encaixava com o grupo à hierarquia de classes e herança, entre outros. Assim, durante este trabalho prático, foi preciso ter um especial cuidado com a organização da apresentação do diagrama de classes já que todos os elementos do grupo trabalharam com o mesmo IDE (BlueJ) de modo a conseguir perceber, à primeira vista, todas as ligações feitas entre as classes bem como a junção das classes do mesmo "conjunto" (por exemplo, tudo o que tinha a ver com os jogadores junto, tudo o que tinha a ver com as exceções junto, por aí em diante).

Para além disso, o esforço constante para respeitar sempre o modelo de associação por composição e o encapsulamento ajudou-nos a compreender melhor o quão importante é o nosso código ser percetível (fácil de entender) para, deste modo, também poder ser reutilizado mais tarde quando assim for preciso.