Universidade do Minho

Departamento de Informática

Trabalho Prático

GRUPO 49

---

# Programação Orientada a Objetos

---

João Manuel Silva de Amorim A89466

João Sá A89550

Rodrigo Caldas Meira A89511

Junho de 2021

# Conteúdo

# Capítulo 1

# Introdução

No âmbito da Unidade Curricular de Programação Orientada a Objetos, foi-nos proposta a realização de um projeto que prevê a implementação de um sistema de gestão e simulação de equipas de um determinado desporto inspirada no conhecido jogo Football Manager.

# Capítulo 2

# Descrições da arquitetura e decisões tomadas

A arquitetura escolhida para o desenvolvimento deste projeto foi a MVC, separando desta forma a aplicação em 3 camadas:

- Model (camada de manipulação dos dados);
- View (camada de interação com o utilizador);
- Controller (camada de controlo, faz a ponte entre a view e o modelo);

# Capítulo 3

# Descrição da aplicação desenvolvida

O package model contém dez classes com informação dos dados:

- Avançado
- Defesa
- Equipa
- Guarda Redes
- Jogador
- Jogo
- Lateral
- Médio
- Sistema
- User

A classe **Jogador** é uma classe abstrata das classes Avançado, Defesa, Guarda Redes, Lateral, Médio. Estas classes que estendem a classe Jogador, possuem métodos que calculam a habilidade de um jogador consoante a posição que ocupam. A classe abstrata tem como variáveis de instância o nome, o número do jogador e as diversas habilidades que um jogador pode ter tais como: velocidade, resistência, destreza, impulsão, entre outros.

A classe **Jogo**, por sua vez tem como variáveis de instância o nome e o número de golos da equipa da casa e da equipa visitante, a data do encontro, uma lista com os jogadores da equipa de fora e outra lista com os jogadores da equipa caseira.

A classe **User** tem como variáveis de instância o email, a palavra passe, o id de um utilizador, tal como o nome da equipa que o mesmo pretende escolher e, também uma varíavel do tipo booleano que diz se o jogador está ou não *"logado"*.

A classe **Equipa** tem como variáveis de instância o nome, os jogadores, a tática e a habilidade da equipa.

Para finalizar, implementamos a classe **Sistema** que tem como variáveis de instância um mapa de utilizadores em que a chave corresponde ao email e o valor devolve um objeto da classe Utilizadores.
Temos também um mapa das equipas em que a chave é relativa ao nome da equipa e o valor devolve um objeto da classe equipa. Adicionalmente, temos duas listas uma com a informação dos jogos que já foram realizados e outra com os jogos que ainda irão acontecer, os jogos futuros.
Por último, temos outra variável de instância, denominada por, equipaUtil do tipo Equipa que tem como funcionalidade armazenar a equipa do utilizador, fizemos desta forma por acharmos mais prático.

O projeto desenvolvido respeita os princípios da programação orientada a objetos, um desses princípios implementados no trabalho prático foi carregar e gravar o estado de um programa num ficheiro de objetos, mais especificamente, num ficheiro binário.

```java
public static Sistema carrega(String nomeficheiro) throws IOException, ClassNotFoundException {
    FileInputStream r = new FileInputStream(nomeficheiro);
    ObjectInputStream o = new ObjectInputStream(r);
    Sistema g = (Sistema) o.readObject();
    o.close();
    return g;
}
```

Figura 3.1: Inicia a aplicação com o carregamento do estado num determinado ficheiro

```java
public void grava(String nomeficheiro) throws IOException {
    FileOutputStream o = new FileOutputStream(nomeficheiro);
    ObjectOutputStream r = new ObjectOutputStream(o);
    this.equipas.replace(this.equipaUtil.getNome(),this.equipaUtil);
    r.writeObject(this);
    r.flush();
    r.close();
}
```

Figura 3.2: Grava o estado da aplicação num determinado ficheiro.

### 3.0.1 Requisitos

Os requisitos que foram implementados neste trabalho prático foram:

- Ver histórico de um jogador;
- Gerar mais jogos;
- Ver calendário dos jogos;
- Ver plantel;
- Tranferir jogador;
- Ver histórico de jogos;
- Simular jogos;
- Mudar a tática da equipa;

### Ver histórico de um jogador

Neste requisito, mostramos todas as equipas que se encontram no sistema. Depois o utilizador escolhe a equipa que pretende e, de seguida, apresentamos os jogadores da equipa selecionada. O utilizador escolhe o jogador que deseja e, posteriormente aparecerá o histórico do mesmo com a informação de todas as equipas onde já jogou.

```
public List<String> getHistoricoJog(int num,String equipa){
    return new ArrayList<>(this.equipas.get(equipa).getHistoricoJogador(num));
}
```

Figura 3.3: Histórico de um Jogador.

### Gerar mais jogos

Este método é responsável por criar jogos entre a equipa do utilizador e as restantes equipas do sistema com resultados por definir, estes jogos são marcados em intervalos de uma semana. Este requisito acaba por ser auxiliar para permitir que haja jogos para o utilizador simular.

```
public void geraJogos(){
    String equipaUt = this.equipaUtil.getNome();
    int par = 0;
    for(Equipa e : this.equipas.values()){
        if(e.getNome().compareTo(equipaUt)!=0){
            if(par % 2 == 0){
                Jogo jogo = new Jogo(equipaUt,e.getNome(), LocalDate.now().plusWeeks(1*par));
                jogo.onzeCasa(this.equipaUtil.onzeInicial());
                jogo.onzeFora(e.onzeInicial());
                this.jogosFuturos.add(jogo);
            }
            else{
                Jogo jogo = new Jogo(e.getNome(),equipaUt, LocalDate.now().plusWeeks(1*par));
                jogo.onzeCasa(e.onzeInicial());
                jogo.onzeFora(this.equipaUtil.onzeInicial());
                this.jogosFuturos.add(jogo);
            }
            par++;
        }
    }

}
```

Figura 3.4: Gera Jogos.

### Ver calendários dos Jogos

Este método apresenta a lista de jogos por realizar no sistema.

```
public List<String> listaJogosFuturos(){
    List<String> ret = new ArrayList<>();
    if(this.jogosFuturos.size() > 0){
        for (Jogo j : this.jogosFuturos) {
            ret.add(j.toStringFuturo());
        }
    }
    return ret;
}
```

Figura 3.5: Calendário dos Jogos.

### Ver plantel

Este método apresenta uma lista de todos os jogadores que constituem a equipa do utilizador.

```
public List<String> listarJogadoresUtilizador() {
    return listarJogadores(this.equipaUtil.getNome());
}
```

Figura 3.6: Ver plantel.

### Tranferir Jogador

Este requisito divide-se em duas opções:

- comprar um jogador;
- vender um jogador;

Para a opção comprar um jogador apresenta-se a lista de todas as equipas do sistema. Posteriormente, o utilizador escolhe uma equipa e é apresentada a lista dos jogadores da mesma. Este indica agora o jogador que quer comprar e o sistema trata de o colocar na equipa do utilizador.

```
public void compra(String equipaOrigem, int numeroJogador) {
    Jogador trans = this.equipas.get(equipaOrigem).buscaJpeloN(numeroJogador);
    while(this.equipaUtil.existeNumero(numeroJogador)) numeroJogador++;
    trans.setNumeroJogador(numeroJogador);
    this.equipas.get(equipaOrigem).removeJogador(trans);
    this.equipaUtil.insereJogador(trans);
}
```

Figura 3.7: Compra de um jogador.

Para a opção de vender um jogador apresenta-se a lista dos jogadores da equipa do utilizador e este indica qual o jogador que quer vender. Após esta ação é apresentada a lista das equipas do sistema, o utilizador indica agora a equipa a que se destina o seu jogador e o sistema trata de colocar o jogador na mesma.

```
public void venda(String equipaDestino, int numeroJogador) {
    Jogador trans = this.equipaUtil.buscaJpeloN(numeroJogador);
    while(this.equipas.get(equipaDestino).existeNumero(numeroJogador)) numeroJogador++;
    trans.setNumeroJogador(numeroJogador);
    this.equipaUtil.removeJogador(trans);
    this.equipas.get(equipaDestino).insereJogador(trans);
}
```

Figura 3.8: Venda de um jogador.

### Ver histórico dos jogos

A lista de jogos realizados contém todos os jogos que estão no ficheiro logs, este método apresenta uma lista de todos os jogos realizados pela equipa do utilizador. Mais detalhadamente, este método acede ao nome da equipa do utilizador e compara com o nome da equipa dos jogos, utilizando o compareTo e caso essas equipas forem iguais, então o nome da equipa será guardada na lista que irá ser retornada.

```
public List<String> listaJogosFeitos(){
    List<String> ret = new ArrayList<>();
    for(Jogo j : this.jogosRealizados) {
        if(j.getEquipaCasa().compareTo(this.equipaUtil.getNome()) == 0 || j.getEquipaFora().compareTo(this.equipaUtil.getNome()) == 0)
        ret.add(j.toString());
    }
    return ret;
}
```

Figura 3.9: Histórico dos Jogos.

### Simular Jogo

O requisito simular Jogo verifica a habilidade de cada equipa e à equipa que joga em casa acrescenta mais 10% de habilidade, ou seja, a equipa que tiver mais habilidade é a vencedora do jogo.

A habilidade de cada equipa é calculada através da média das habilidades de todos os jogadores. Por sua vez, é no parser que após a construção do sistema chamamos o método atribiu habilidades que tem a função de percorrer todo o mapa das equipas chamando para cada equipa, o método calcula habilidade que se encontra na classe equipa.

O resultado do encontro é gerado por dois números aleatórios entre 0 e 5, salientando que, a equipa que ganhou terá de ter um número maior que a equipa derrotada e, para isso, criamos uma serie de condições.

```
public String simularJogo() {
    Jogo jogo = this.getJogosFuturos().get(0);
    Equipa caseira = this.getEquipas().get(jogo.getEquipaCasa()).clone();
    Equipa visitante = this.getEquipas().get(jogo.getEquipaFora()).clone();
    String vencedor = jogo.vencedor(caseira.getHabilidadeE(),visitante.getHabilidadeE());
    this.jogosRealizados.add(jogo);
    this.jogosFuturos.remove( index: 0);
    return vencedor;
}
```

Figura 3.10: Método que simula o jogo.

### Mudar Tática da Equipa

Existem duas táticas no nosso projeto, que estão armazenadas na variável de instância tática da classe equipa, ou seja, se o valor dessa variável for 0 a tática corresponde ao 4-3-3, caso o valor seja de 1 a tática será 4-4-2. O que este requisito adicional altera a formação tática de uma dada equipa.

Decidimos implementar este requisito, de forma a mostrar o que poderá acontecer num jogo de futebol. Se uma equipa estiver a perder, mudar a tática poderá ser uma boa opção.

O método que faz uso deste requisito é o mudarTatica que se encontrar no sistema e chama o método mudaTatica que se encontra na classe equipa que faz o tal cálculo que altera a formação.

```
public void mudaTatica(){
    if(this.tatica == 0) this.tatica = 1;
    else this.tatica = 0;
}
```

Figura 3.11: Método que mudaTatica da classe Equipa.

```
public void mudarTatica() {
    this.equipaUtil.mudaTatica();
}
```

Figura 3.12: Método mudar tática da classe Sistema.

# Capítulo 4

# Conclusão

Concluindo, de acordo com o objetivo do projeto, concluímos que este foi atingido. No entanto, no decorrer do trabalho deparamo-nos com diversos desafios que, na sua grande maioria, foram ultrapassados. À medida que o trabalho ia sendo concebido, fomos aprofundando os nossos conceitos no que diz respeito à programação orientada a objetos, tanto relativamente ao tratamento de erros, como na leitura e escrita para um ficheiro de objetos, entre outros.