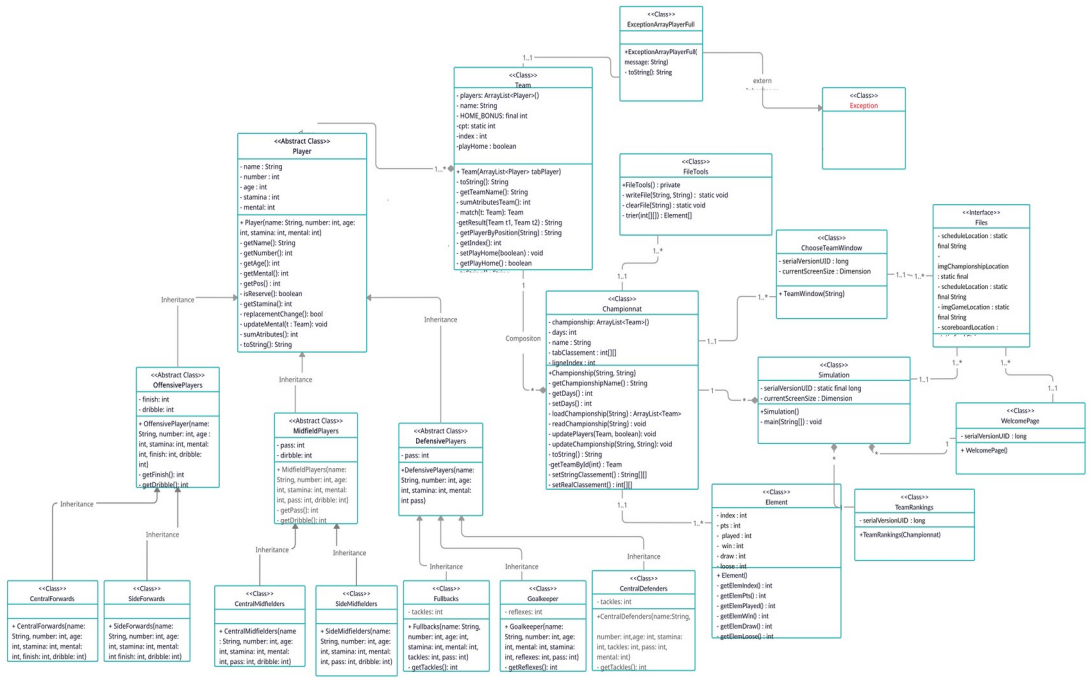
Projet LU2IN002 - 2020-2021

Numéro du groupe de TD/TME :

Nom : Sima	*Nom :* Walter	*Nom :* Sadykova
Prénom : Daniel	*Prénom :* Walter	*Prénom :* Karima
N° étudiant : 3800992	*N° étudiant :*	*N° étudiant :*

Thème choisi (en 2 lignes max.)
Simulation de championnat de football (Première ligue anglaise) avec génération de scores et du classement journée par journée.

Description des classes et de leur rôle dans le programme (2 lignes max par classe)
Classe **Player** avec classes filles: **DefensivePlayer**, **MidfieldPlayer** et **OffensivePlayer** → définit sommairement les rôles des Players. Classe **CentralDefender**, **Fullback** et **Goalkeeper** → classes filles de la classe DefensivePlayer qui définit précisément le rôle de chaque Player en défense Classe **CentralMidfielder**, **SideMidfielder** → classes filles de la classe MidfieldPlayer qui définit précisément le rôle de chaque Player au milieu du terrain Classe **CentralForward**, **SideForward** → classes filles de la classe OffensivePlayer qui définit précieusement le rôle de chaque Player en attaque Classe **Team** → classe avec composition de Players qui sert a créer une équipe composée de joueurs et a récupérer les statistiques au cours du championnat. Classe **Championnat** → avec composition de Team Classe **Element** → sert à récupérer les données d'un élément d'un tableau pour effectuer des opération style le tri du tableau (utile pour l'actualisation du classement) Classe **TeamsWindow** → avec heritage de JFrame et implémentation de Files. Elle sert à afficher une fenêtre contenant le score du match Classe **Welcome** → page avec héritage de JLabel et implémentation de Files. Elle sert de page d'accueil de la simulation Classe **TeamRankings** → avec heritage de JPanel : sert à charger une interface graphique contenant le tableau des équipes Classe **Simulation** → hérite de JFrame et implémente Files : sert à instancier un championnat et à charger une fenêtre graphique qui simule le déroulement du championnat et met à jour le classement Classe **FileTools** → classe static contenant des méthodes statiques utiles dans certaines classes du projet Interface **Files** → interface contenant essentiellement des chemins de fichiers / images utilisés comme argument dans plusieurs classes → Classe **ExceptionArrayPlayerFull** → hérite de Exception et sert en cas de problème à l'instanciation d'une equipe

Schéma UML des classes vision fournisseur (dessin “à la main” scanné ou photo acceptés)
 Le schéma est également disponible sous forme .png dans le dossier

Checklist des contraintes prises en compte:	*Nom(s) des classe(s) correspondante(s)*
Classe contenant un tableau ou une ArrayList	Team, Championnat, FileTools
Classe avec membres et méthodes statiques	FilsTools
Classe abstraite et méthode abstraite	Player, DefensivePlayer, MidfieldPlayer, OffensivePlayer
Interface	Files
Classe avec un constructeur par copie ou clone()	
Définition de classe étendant Exception	ExceptionArrayPlayerFull
Gestion des exceptions	Championnat, Simulation, FileTools
Utilisation du pattern singleton	

Présentation de votre projet (max. 2 pages) : texte libre expliquant en quoi consiste votre projet.

Notre projet consiste a créer une simulation d'un championnat de football existant. Nous avons choisi le championnat anglais car c'est le championnat le plus imprévisible, spectaculaire et connu. (Pour aller plus loin on pourrai également ajouter d'autres championnats/compétitions.)

Pour parvenir a créer notre projet, nous avons eu besoin de plusieurs bases de données, notamment pour tous les joueurs (dans le cadre de notre projet chaque équipe est composé de 16 joueurs), équipes et le calendrier des matchs. En ce qui concerne la base de données des joueurs nous avons attribué quelques caractéristiques selon le rôle de chaque joueur et selon le niveau de chaque joueur. Certaines de ces caractéristiques sont communes a tous les joueurs, comme stamina et mental. De plus, l'attribut mental d'un joueur tend a être modifié en fonction des résultats de son équipe, ce qui peut diminuer les chances de victoire de son équipe.

Les résultats sont générés en fonction des attributs de toute l'équipe. Il y a également une part d'aléatoire et le fait de jouer a domicile est pris en compte.
Ces résultats sont ensuite charges dans un tableau qui sert de classement et qui est actualisé au bout de chaque journée/étape de championnat. Pour illustrer cela, on utilise la Java swing qui est une bibliothèque graphique qui nous a permis de mettre en place le classement, différentes illustrations et les scores lors des journées.

Copier / coller vos classes et interfaces à partir d'ici :

```
public abstract class Player {
	private String name;
	protected int number, age, stamina;
	protected int mental;
/*-------------------------------------------------------------------------------*/
	public Player(String name, int number, int age, int stamina, int mental){
		this.name = name;
		this.number = number;
		this.age = age;
		this.stamina = stamina;
		this.mental = mental;
	}
/*-------------------------------------------------------------------------------*/
	public String getName(){return name;}
	public int getNumber(){return number;}
	public int getAge(){return age;}
	public int getStamina(){return stamina;}
	public int getMental(){return mental;}
/*-------------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return name+", "+age+", "+number+"||Stamina: "+stamina+"|| Mental: 
"+mental+"||Pass: ";
	}
/*-------------------------------------------------------------------------------*/
	public abstract String getPos();
	public abstract int sumAtributes();
/*-------------------------------------------------------------------------------*/
	// Non utilisé car pas le temps de developper le mechanisme mais importance theorique 
majeure
```

```
	public abstract boolean replacementChance();
/*--------------------------------------------------------------------------*/
	public void updateMental(boolean result){
		if(result){
			if(mental <= 90) mental += 10;
		}
		else{
			if(mental >= 20) mental -= 10;
		}
	}
/*--------------------------------------------------------------------------*/
}



public abstract class DefensivePlayer extends Player {
	// Player technical caracteristics
	protected int pass;
/*--------------------------------------------------------------------------*/
	public DefensivePlayer(String name, int number, int age, int stamina, int mental, int pass) {
		super(name, number, age, stamina, mental);
		this.pass = pass;
	}
/*--------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return super.toString()+"||Pass :"+pass;
	}
/*--------------------------------------------------------------------------*/
	public int getPass(){
		return pass;
	}
/*--------------------------------------------------------------------------*/
	public abstract String getPos();
	public abstract boolean replacementChance();
	public abstract int sumAtributes();
/*--------------------------------------------------------------------------*/
}



public abstract class MidfieldPlayer extends Player {
	// Player technical caracteristics
	protected int pass;
	protected int dribble;
/*--------------------------------------------------------------------------*/
	public MidfieldPlayer(String name, int number, int age, int stamina, int mental, int pass, int
dribble){
		super(name, number, age, stamina, mental);
		this.pass = pass;
		this.dribble = dribble;
	}
```

```
/*-------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return super.toString()+"||Pass :"+pass+"||Dribble :"+dribble;
	}
/*-------------------------------------------------------------------------*/
	public int getPass(){
		return pass;
	}
/*-------------------------------------------------------------------------*/
	public int getDribble(){
		return dribble;
	}
/*-------------------------------------------------------------------------*/
	@Override
	public int sumAtributes(){
		return this.getStamina() + this.getMental() + this.getPass() + this.getDribble();
	}
/*-------------------------------------------------------------------------*/
	public abstract String getPos();
	public abstract boolean replacementChance();
/*-------------------------------------------------------------------------*/
}



public abstract class OffensivePlayer extends Player {
	// Player technical caracteristics
	protected int finish;
	protected int dribble;
/*-------------------------------------------------------------------------*/
	public OffensivePlayer(String name, int number, int age, int stamina, int mental, int finish,
int dribble) {
		super(name, number, age, stamina, mental);
		this.finish = finish;
		this.dribble = dribble;
	}
/*-------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return super.toString()+"||Finish :"+finish+"||Dribble :"+dribble;
	}
/*-------------------------------------------------------------------------*/
	public int getFinish(){
		return finish;
	}
/*-------------------------------------------------------------------------*/
	public int getDribble(){
		return dribble;
	}
/*-------------------------------------------------------------------------*/
	@Override
```

```
	public int sumAtributes(){
		return this.getStamina() + this.getMental() + this.getFinish() + this.getDribble();
	}
/*-------------------------------------------------------------------------------*/
	public abstract String getPos();
	public abstract boolean replacementChance();
/*-------------------------------------------------------------------------------*/
}



public class CentralDefender extends DefensivePlayer {
	private int tackles;
/*-------------------------------------------------------------------------------*/
	public CentralDefender(String name, int number, int age, int stamina, int mental, int pass,
int tackles){
		super(name, number, age, stamina, mental, pass);
		this.tackles = tackles;
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public boolean replacementChance(){
		if(this.getStamina() < 50){  // ~
			return this.getMental() < 90;
		}
		return true;
	}
/*-------------------------------------------------------------------------------*/
	public int getTackles(){
		return tackles;
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public String getPos(){return "Central Defender";}
/*-------------------------------------------------------------------------------*/
	@Override
	public int sumAtributes(){
		return this.getStamina() + this.getMental() + this.getPass() + this.getTackles();
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return "Central Defender: "+super.toString()+"||Tackles :"+tackles;
	}
/*-------------------------------------------------------------------------------*/
}



public class Fullback extends DefensivePlayer {
	private int tackles;
/*-------------------------------------------------------------------------------*/
```

```
	public Fullback(String name, int number, int age, int stamina, int mental, int pass, int
tackles){
		super(name, number, age, stamina, mental, pass);
		this.tackles = tackles;
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public String getPos(){return "Fullback";}
/*-------------------------------------------------------------------------------*/
	@Override
	public boolean replacementChance(){
		if(this.getStamina() < 50){
			return this.getMental() < 90;
		}
		return true;
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return "Fullback: "+super.toString()+"||Tackles :"+tackles;
	}
/*-------------------------------------------------------------------------------*/
	public int getTackles(){
		return tackles;
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public int sumAtributes(){
		return this.getStamina() + this.getMental() + this.getPass() + this.getTackles();
	}
/*-------------------------------------------------------------------------------*/
}



public class Goalkeeper extends DefensivePlayer {
	private int reflexes;
/*-------------------------------------------------------------------------------*/
	public Goalkeeper(String name, int number, int age, int stamina, int reflexes, int pass, int
mental){
		super(name, number, age, stamina, mental, pass);
		this.reflexes = reflexes;
	}
/*-------------------------------------------------------------------------------*/
	@Override
	public String getPos(){return "Goalkeeper";}
/*-------------------------------------------------------------------------------*/
	public int getReflexes(){return reflexes;}
/*-------------------------------------------------------------------------------*/
	@Override
	public boolean replacementChance(){
		if(this.getStamina() < 50){
```

```
                return getMental() < 90;
            }
            return true;
        }
/*-------------------------------------------------------------------------*/
        @Override
        public String toString(){
            return "Goalkeeper: "+super.toString()+"||Reflexes :"+reflexes;
        }
/*-------------------------------------------------------------------------*/
        @Override
        public int sumAtributes(){
            return this.getReflexes() + this.getPass() + this.getMental() + this.getStamina();
        }
/*-------------------------------------------------------------------------*/
}



public class CentralMidfielder extends MidfieldPlayer {
        public CentralMidfielder(String name, int number, int age, int stamina, int mental, int pass,
int dribble){
            super(name,number,age,stamina,pass,dribble,mental);
        }
/*-------------------------------------------------------------------------*/
        @Override
        public String getPos(){ return "Central Midfielder";}
/*-------------------------------------------------------------------------*/
        @Override
        public boolean replacementChance(){
            if(this.getStamina() < 50){
                return this.getMental() < 90;
            }
            return true;
        }
/*-------------------------------------------------------------------------*/
        @Override
        public String toString(){
            return "Central Midfielder: "+super.toString();
        }
/*-------------------------------------------------------------------------*/
}



public class SideMidfielder extends MidfieldPlayer {
        public SideMidfielder(String name, int number, int age, int stamina, int mental, int pass, int
dribble){
            super(name,number,age,stamina,pass,dribble,mental);
        }
/*-------------------------------------------------------------------------*/
        @Override
```

```
	public String getPos(){ return "Side Midfielder";}
/*-------------------------------------------------------------------------*/
	@Override
	public boolean replacementChance(){
		if(this.getStamina() < 50){
			return this.getMental() < 90;
		}
		return true;
	}
/*-------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return "Side Midfielder: "+super.toString();
	}
/*-------------------------------------------------------------------------*/
}



public class CentralForward extends OffensivePlayer {
	public CentralForward(String name, int number, int age, int stamina, int mental, int finish,
int dribble){
		super(name, number, age, stamina, mental, finish, dribble);;
	}
/*-------------------------------------------------------------------------*/
	@Override
	public String getPos(){ return "Central Forward";}
/*-------------------------------------------------------------------------*/
	@Override
	public boolean replacementChance(){
		if(this.getStamina() < 50){
			return getMental() < 90;  // ou this.getMental() ???
		}
		return true;
	}
/*-------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return "Central Forward: "+super.toString();
	}
/*-------------------------------------------------------------------------*/
}



public class SideForward extends OffensivePlayer {
	public SideForward(String name, int number, int age, int stamina, int mental, int finish, int
dribble) {
		super(name, number, age, stamina, mental, finish, dribble);
	}
/*-------------------------------------------------------------------------*/
	@Override
```

```
	public String getPos(){return "Side Forward";}
/*-------------------------------------------------------------------------*/
	@Override
	public boolean replacementChance(){
		if(this.getStamina() < 50){
			return this.getMental() < 90;
		}
		return true;
	}
/*-------------------------------------------------------------------------*/
	@Override
	public String toString(){
		return "Side Forward: "+super.toString();
	}
/*-------------------------------------------------------------------------*/
}



import java.io.*;
import java.util.*;
import java.util.concurrent.TimeUnit;


public class Championnat{
	private String name;
	private ArrayList<Team> championship = new ArrayList<Team>();
	private int[][] tabClassement;
	private int days = 1;
	private int ligneIndex;
/*-------------------------------------------------------------------------*/
	public Championnat(String fileName, String name) {
		this.name = name;
		championship = loadChampionship(fileName);
		tabClassement = new int[championship.size()][6];
		for (int i = 0; i < championship.size(); i++) {
			tabClassement[i][0] = championship.get(i).getIndex();
			for (int j = 1; j < 6; j++) {
				tabClassement[i][j] = 0;
			}
		}
		ligneIndex = 0;
	}
/*-------------------------------------------------------------------------*/
	public String getChampionshipName() {
		return name;
	}
/*-------------------------------------------------------------------------*/
	public int getDays() {
		return days;
	}
/*-------------------------------------------------------------------------*/
```

```
	public void setDays(String schedule) {
		BufferedReader br = null;
		try {
			br = new BufferedReader(new FileReader(schedule));

			String ligne;
			ligne = br.readLine();
			while (ligne != null) {
				if (ligne.length() == 0)
					days++;
				ligne = br.readLine();
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			if (br != null)
				try {
					br.close();
				} catch (IOException e) {
					e.printStackTrace();
				}
		}

	}
/*-------------------------------------------------------------------------*/
	public ArrayList<Team> loadChampionship(String teamsFile) {
		ArrayList<Player> tabPlayer = new ArrayList<Player>();
		ArrayList<Team> tabTeam = new ArrayList<Team>();
		String teamName = "";

		try {
			FileReader fr = new FileReader(new File(teamsFile));
			BufferedReader br = new BufferedReader(fr);

			String ligne = br.readLine();
			while (ligne != null) { // lecture ligne par ligne jusqu'a la fin du fichier
				int i = 0;
				while (i < 17) { // 17 lignes par equipe
					String[] liste = ligne.split(" ");
					if (liste.length == 1) {
						teamName = liste[0];
						tabPlayer = new ArrayList<Player>();
					} else { // Exception NumberFormatException
						try {
							Player p;
							switch (liste[0]) {
								case "Goalkeeper":
									p = new Goalkeeper(liste[1],
Integer.parseInt(liste[2]), Integer.parseInt(liste[3]),
```

```
        Integer.parseInt(liste[4]), Integer.parseInt(liste[5]),

        Integer.parseInt(liste[6]), Integer.parseInt(liste[7]));
                                                        break;
                                                case "Fullback":
                                                        p = new Fullback(liste[1],
Integer.parseInt(liste[2]), Integer.parseInt(liste[3]),

        Integer.parseInt(liste[4]), Integer.parseInt(liste[5]),

        Integer.parseInt(liste[6]), Integer.parseInt(liste[7]));
                                                        break;
                                                case "CentralDefender":
                                                        p = new
CentralDefender(liste[1], Integer.parseInt(liste[2]),

        Integer.parseInt(liste[3]), Integer.parseInt(liste[4]),

        Integer.parseInt(liste[5]), Integer.parseInt(liste[6]),

        Integer.parseInt(liste[7]));
                                                        break;
                                                case "CentralMidfielder":
                                                        p = new
CentralMidfielder(liste[1], Integer.parseInt(liste[2]),

        Integer.parseInt(liste[3]), Integer.parseInt(liste[4]),

        Integer.parseInt(liste[5]), Integer.parseInt(liste[6]),

        Integer.parseInt(liste[7]));
                                                        break;
                                                case "SideMidfielder":
                                                        p = new SideMidfielder(liste[1],
Integer.parseInt(liste[2]),

        Integer.parseInt(liste[3]), Integer.parseInt(liste[4]),

        Integer.parseInt(liste[5]), Integer.parseInt(liste[6]),

        Integer.parseInt(liste[7]));
                                                        break;
                                                case "SideForward":
                                                        p = new SideForward(liste[1],
Integer.parseInt(liste[2]),

        Integer.parseInt(liste[3]), Integer.parseInt(liste[4]),

        Integer.parseInt(liste[5]), Integer.parseInt(liste[6]),

        Integer.parseInt(liste[7]));
                                                        break;
```

```
                                                default:
                                                        p = new
CentralForward(liste[1], Integer.parseInt(liste[2]),

        Integer.parseInt(liste[3]), Integer.parseInt(liste[4]),

        Integer.parseInt(liste[5]), Integer.parseInt(liste[6]),

        Integer.parseInt(liste[7]));
                                                                break;
                                                        }
                                                        tabPlayer.add(p);
                                                } catch (NumberFormatException e) {
                                                        System.out.println(e.getMessage());
                                                }
                                        }
                                        i++;
                                        ligne = br.readLine();
                                }
                                tabTeam.add(new Team(teamName, tabPlayer));
                        }
                        br.close();
                        fr.close();
                } catch (NullPointerException e) {
                        System.out.println(e);
                } catch (FileNotFoundException e) {
                        System.out.println(e);
                } catch (IOException e) {
                        System.out.println(e);
                }
                return tabTeam;
        }
/*---------------------------------------------------------------------------*/
        public void readChampionship(String scoreboardFile) { // afficher le championnat
                BufferedReader br = null;
                String ligne;

                try {
                        br = new BufferedReader(new FileReader(scoreboardFile));
                        ligne = br.readLine();
                        while (ligne != null) {
                                // on recupere l'id des 2 equipes ainsi que l'id de l'equipe gagnante
dans le
                                // ficher scoreboard
                                String[] tabI = ligne.split(" ");
                                int indexT1 = Integer.parseInt(tabI[0]);
                                int indexT2 = Integer.parseInt(tabI[1]);
                                int winner = Integer.parseInt(tabI[2]);
                                // int score1 = Integer.parseInt(tabI[3]);
                                // int score2 = Integer.parseInt(tabI[4]);

                                // on parcourt le tabClassement pour identifier les 2 equipes et
```

```
mettre a jour
                                // leurs points
                                for (int i = 0; i < championship.size(); i++) {
                                        if (tabClassement[i][0] == indexT1) {
                                                tabClassement[i][1]++;
                                                if (winner == indexT1) { // T1 win
                                                        tabClassement[i][2]++;
                                                        tabClassement[i][5] += 3;
                                                } else if (winner == -1) { // draw
                                                        tabClassement[i][3]++;
                                                        tabClassement[i][5] += 1;
                                                } else
                                                        tabClassement[i][4]++;
                                        }
                                        if (tabClassement[i][0] == indexT2) {
                                                tabClassement[i][1]++;
                                                if (winner == indexT2) { // T2 win
                                                        tabClassement[i][2]++;
                                                        tabClassement[i][5] += 3;
                                                } else if (winner == -1) {
                                                        tabClassement[i][3]++;
                                                        tabClassement[i][5] += 1;
                                                } else
                                                        tabClassement[i][4]++;
                                        }
                                }

                                ligne = br.readLine();
                        }
                } catch (FileNotFoundException e) {
                        System.out.println(e.getMessage());
                } catch (IOException e) {
                        System.out.println(e.getMessage());
                } finally {
                        try {
                                if (br != null)
                                        br.close();
                        } catch (IOException ex) {
                                System.out.println(ex.getMessage());
                        }
                }
        }
/*-------------------------------------------------------------------------*/
        public void updateChampionship(String scheduleFile, String scoreboardFile) {
                BufferedReader br = null;
                BufferedWriter bw = null;
                String ligne;
                String result;
                Team t1 = null;
                Team t2 = null;
                boolean b = true;
```

```
		try {
			// on utilise un BufferedReader pour lire le scheduleFile ligne par ligne
			br = new BufferedReader(new FileReader(scheduleFile));
			bw = new BufferedWriter(new FileWriter(scoreboardFile));

			for (int i = 0; i < ligneIndex; i++) br.readLine();
			ligne = br.readLine();
			while (ligne.length() != 0) {
				String[] tabI = ligne.split(" ");
				int indexT1 = Integer.parseInt(tabI[0]);
				int indexT2 = Integer.parseInt(tabI[1]);
				t1 = null;
				t2 = null;
				for (int i = 0; i < championship.size(); i++) {
					if (championship.get(i).getIndex() == indexT1)
						t1 = championship.get(i);
					if (championship.get(i).getIndex() == indexT2)
						t2 = championship.get(i);
				}
				Team winner = t1.match(t2);
				// on utilise l'egalite pour identifier le vainqueur
				if (winner != null) {
					updatePlayers(t1, (t1 == winner));
					updatePlayers(t2, (t2 == winner));
					String data = t1.getIndex() + " " + t2.getIndex() + " " +
winner.getIndex() + "\n";
					FileTools.writeFile(scoreboardFile, data);
				} else {
					String data = t1.getIndex() + " " + t2.getIndex() + " " + (-1)
+ "\n";
					FileTools.writeFile(scoreboardFile, data);
				}

				result = t1.getResult(t2, winner);
				if(b)
					new TeamsWindow(result);
				b = !b;

				try {
					TimeUnit.MILLISECONDS.sleep(500);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}

				ligne = br.readLine();
			}
			ligneIndex += 11;
		} catch (FileNotFoundException e) {
			System.out.println(e.getMessage());
		} catch (IOException e) {
			System.out.println(e.getMessage());
		} finally {
```

```
                        try {
                                if (br != null)
                                        br.close();
                                if (bw != null)
                                        bw.close();
                        } catch (IOException ex) {
                                System.out.println(ex.getMessage());
                        }
                }
        }
/*-------------------------------------------------------------------------------*/
        public void updatePlayers(Team t, boolean result) {
                // on utilise un boolean pour : win / loose
                // (un draw etant percu comme une defaite pour tout footballer qui se respecte
                // !)
                for (int i = 0; i < t.players.size(); i++) {
                        t.players.get(i).updateMental(result);
                }
        }
/*-------------------------------------------------------------------------------*/
        @Override
        public String toString() {
                String chaine = "";
                for (int i = 0; i < championship.size(); i++)
                        chaine += championship.get(i) + "\n";

                return chaine;
        }
/*-------------------------------------------------------------------------------*/
        public Team getTeamById(int id) {
                for (int i = 0; i < championship.size(); i++) {
                        if (id == championship.get(i).getIndex())
                                return championship.get(i);
                }
                return null;
        }
/*-------------------------------------------------------------------------------*/
        public String[][] setStringClassement() {
                int[][] realClassement = setRealClassement();
                String[][] tab = new String[realClassement.length][6];
                for (int i = 0; i < realClassement.length; i++) {
                        tab[i][0] = getTeamById(realClassement[i][0]).getTeamName();
                        for (int j = 1; j < 6; j++) {
                                tab[i][j] = "" + realClassement[i][j];
                        }
                }
                return tab;
        }
/*-------------------------------------------------------------------------------*/
        public int[][] setRealClassement(){
                Element[] tab = FileTools.trier(tabClassement);
                int[][] tabRes = new int[tabClassement.length][6];
```

```
                for(int i=0; i<tabClassement.length;i++){
                        tabRes[i][0] = tab[i].getElemIndex();
                        tabRes[i][1] = tab[i].getElemPlayed();
                        tabRes[i][2] = tab[i].getElemIWin();
                        tabRes[i][3] = tab[i].getElemIDraw();
                        tabRes[i][4] = tab[i].getElemLoose();
                        tabRes[i][5] = tab[i].getElemPts();
                }

                return tabRes;
        }
/*-------------------------------------------------------------------------*/
}



public class Element {
   private int index; //
   private int pts; //
   private int played;
   private int win;
   private int draw;
   private int loose;
/*-------------------------------------------------------------------------*/
   public Element(int index, int played, int win, int draw, int loose, int pts){
      this.index = index;
      this.pts = pts;
      this.played = played;
      this.win = win;
      this.draw = draw;
      this.loose = loose;
   }
/*-------------------------------------------------------------------------*/
   public int getElemIndex(){
      return index;
   }
/*-------------------------------------------------------------------------*/
   public int getElemPts(){
      return pts;
   }
/*-------------------------------------------------------------------------*/
        public int getElemLoose() {
                return loose;
        }
/*-------------------------------------------------------------------------*/
        public int getElemIDraw() {
                return draw;
        }
/*-------------------------------------------------------------------------*/
        public int getElemIWin() {
                return win;
```

```
        }
/*-------------------------------------------------------------------------------*/
        public int getElemPlayed() {
                return played;
        }
/*-------------------------------------------------------------------------------*/
}



public class ExceptionArrayPlayerFull extends Exception {
   public ExceptionArrayPlayerFull(String message){
      super(message);
   }
/*-------------------------------------------------------------------------------*/
   public String toString(){
      return "Tableau de joueurs plein";
   }
/*-------------------------------------------------------------------------------*/
}



public interface Files {
   String imgChampionshipLocation = "Championship.jpg";
   String teamsLocation = "database/Teams.txt";   // ..
   String scheduleLocation = "database/schedule.txt";
   String imgGameLocation = "Game.jpg";
   String scoreboardLocation = "database/scoreboard.txt";
}




import java.io.*;

public class FileTools {
        // classe static contenant methodes utiles pour la simulation
   private  FileTools(){}
/*-------------------------------------------------------------------------------*/
        // methode pour ecrire dans un fichier en argument
   public static void writeFile(String file, String data) {
                BufferedWriter writer = null;
                try {
                        writer = new BufferedWriter(new FileWriter(file, true));
                        writer.write(data);
                } catch (IOException e) {
                        System.out.println(e.getMessage());
                } finally {
                        try {
                                if (writer != null)
                                        writer.close();
```

```
                        } catch (IOException ex) {
                                System.out.println(ex.getMessage());
                        }
                }
        }
/*---------------------------------------------------------------------------*/
        // methode pour reinitialiser le contenu d'un fichier
        public static void clearFile(String file) {
      try {
        FileWriter f = new FileWriter(file, false);
        PrintWriter pw = new PrintWriter(f, false);
        pw.flush();
        pw.close();
        f.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
        }
/*---------------------------------------------------------------------------*/
        // trie un tableau d'elements en ordre decroissant
        public static Element[] trier(int[][] tab){
                Element[] tableau = new Element[tab.length];
                for(int i=0;i<tab.length;i++){
                        for(int j=0;j<6;j++){
                                tableau[i] = new Element(tab[i][0],tab[i][1],tab[i][2],tab[i]
[3],tab[i][4], tab[i][5]);
                        }
                }

                Element temp;
                int max;
                for (int index = 0; index < tableau.length - 1; index++){
                        max = index;
                        for (int i = index + 1; i < tableau.length; i++)
                                if (tableau[i].getElemPts() > tableau[max].getElemPts())
                                        max = i;

                        temp = tableau[max];
                        tableau[max] = tableau[index];
                        tableau[index] = temp;
                }
                return tableau;
        }
/*---------------------------------------------------------------------------*/
}



import java.awt.*;
import java.util.concurrent.TimeUnit;

import javax.swing.*;
```

```
public class Simulation extends JFrame implements Files{
	private static final long serialVersionUID = 1L;
	private Dimension currentScreenSize;
/*---------------------------------------------------------------------------*/
	public Simulation() {
		Championnat championnat = new Championnat(teamsLocation,
"PremierLeague");
		championnat.setDays(scheduleLocation);
		currentScreenSize = Toolkit.getDefaultToolkit().getScreenSize();

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setTitle("Classement");
		setLocation((int)currentScreenSize.getWidth()/2,
(int)currentScreenSize.getWidth()/21);
		setSize(680, 750);

		JPanel startPanel = new JPanel();
		startPanel.add(new WelcomePage());

		add(startPanel);
		setVisible(true);
		try {
			TimeUnit.SECONDS.sleep(3);
		} catch (InterruptedException e1) {
			e1.printStackTrace();
		}
		FileTools.clearFile(scoreboardLocation);
		championnat.readChampionship(scoreboardLocation);
		add(new TeamRankings(championnat));
		revalidate();
		for (int i = 0; i < championnat.getDays(); i++) {
			try {
				championnat.updateChampionship(scheduleLocation,
scoreboardLocation);
				championnat.readChampionship(scoreboardLocation);
				add(new TeamRankings(championnat));
				revalidate();
				TimeUnit.SECONDS.sleep(1);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}

		}
	}
/*---------------------------------------------------------------------------*/
	public static void main(String [] args) {
		new Simulation();
	}
/*---------------------------------------------------------------------------*/
}
```

```
import java.util.ArrayList;

public class Team {
	public String name;
	protected ArrayList<Player> players = new ArrayList<Player>();
	private static final int HOME_BONUS = 55; // bonus constant du match a domicile
	private static int cpt = 0; //compteur d'index d'equipe pour gestion du schedule
	private int index;
	private boolean playHome;
/*-------------------------------------------------------------------------*/
	public Team(String name, ArrayList<Player> tabPlayer){
		this.name = name;
		this.index = cpt++;
		playHome = false;
		for(int i=0; i < 16; i++)    //16 Players in a team
			players.add(tabPlayer.get(i));
	}
/*-------------------------------------------------------------------------*/
	public int sumAtributesTeam(){
		int total = 0;
		for(int i=0; i < 16; i++)
			total += players.get(i).sumAtributes();

		return total;
	}
/*-------------------------------------------------------------------------*/
	public Team match(Team t){  // retourne l'equipe gagnate
		int totalTeam1 = this.sumAtributesTeam();
		int totalTeam2 = t.sumAtributesTeam();
		this.setPlayHome(true);
		t.setPlayHome(false);

		totalTeam1 += (int)(Math.random()*21);
		totalTeam2 += (int)(Math.random()*21);

		totalTeam1 += HOME_BONUS;  // t1 plays at home

		if(totalTeam1 > (totalTeam2+15)) return this; // +10 marge du match nul
		else if(totalTeam2 > (totalTeam1+15)) return t;
		else return null;  // match nul
	}
/*-------------------------------------------------------------------------*/
	public String getResult(Team t, Team winner){  // retourne le score des deux teams
		int scoreTeam1 = 0, scoreTeam2 = 0;  // pour eviter les warnings

		if(winner == null){ // draw
			scoreTeam1 = (int)(Math.random()*6); // 5 score max  a decider
			scoreTeam2 = scoreTeam1;
			if(this.getPlayHome())
				return this.getTeamName()+" "+scoreTeam1+" - "+scoreTeam2+"
"+t.getTeamName()+"\n";
```

```
                        else
                                return t.getTeamName()+" "+scoreTeam1+" - "+scoreTeam2+"
"+t.getTeamName()+"\n";
                }
                if(winner.getPlayHome()){
                        while(scoreTeam1 <= scoreTeam2){
                                scoreTeam1 = (int)(Math.random()*6);
                                scoreTeam2 = (int)(Math.random()*6);
                        }
                        if(winner == t)
                                return winner.getTeamName()+" "+scoreTeam1+" -
"+scoreTeam2+" "+this.getTeamName()+"\n";
                        else
                                return winner.getTeamName()+" "+scoreTeam1+" -
"+scoreTeam2+" "+t.getTeamName()+"\n";
                }
                else{ // winner ne joue pas a domicile mais this. joue a domicile
                        while(scoreTeam2 <= scoreTeam1){
                                scoreTeam1 = (int)(Math.random()*6);
                                scoreTeam2 = (int)(Math.random()*6);
                        }
                        if(winner == t)
                                return this.getTeamName()+" "+scoreTeam1+" - "+scoreTeam2+"
"+winner.getTeamName()+"\n";
                        else
                                return t.getTeamName()+" "+scoreTeam1+" - "+scoreTeam2+"
"+winner.getTeamName()+"\n";
                }
        }
/*-------------------------------------------------------------------------*/
        public String getScorersAssists(Team teamOutside, String result){  // retourne les buteurs et
assists
                String resultBis = result;
                String[] tab = result.split(" ");
                int scoreTeam1 = Integer.parseInt(tab[1]), scoreTeam2 = Integer.parseInt(tab[2]);

                for(int i=0; i < scoreTeam1; i++){
                        int opportunity = (int)(Math.random()*101);
                        if(opportunity >= 50) // the attackers score
                                resultBis += this.getPlayerByPosition("Offensive")+"("+getAssist()
+")";
                        else if((opportunity < 50) && (opportunity >= 14)) // midfielder scores
                                resultBis += getPlayerByPosition("Midfield")+"("+getAssist()+")";
                        else if((opportunity < 14) && (opportunity >= 0)) // defenders scores
                                resultBis += getPlayerByPosition("Defensive")+"("+getAssist()
+")";
                        }
                        resultBis += " || "; // a redefinir si jamais pour la lecture

                        for(int i=0; i < scoreTeam2; i++){
                        int opportunity = (int)(Math.random()*101);
                        if(opportunity >= 50) // the attackers score
```

```
                        resultBis += teamOutside.getPlayerByPosition("Offensive")
+"("+teamOutside.getAssist()+")";
                    else if((opportunity < 50) && (opportunity >= 14)) // midfielder scores
                        resultBis += teamOutside.getPlayerByPosition("Midfield")
+"("+teamOutside.getAssist()+")";
                    else if((opportunity < 14) && (opportunity >= 0)) // defenders scores
                        resultBis += teamOutside.getPlayerByPosition("Defensive")
+"("+teamOutside.getAssist()+")";
                    }

                    return resultBis +"\n";
                }
/*-------------------------------------------------------------------------*/
        public String getAssist(){  // utilisé dans getScorersAssists
                int opportunity = (int)(Math.random()*101);
                if(opportunity >= 50) // the attackers assist
                        return getPlayerByPosition("Offensive");
                else if((opportunity < 50) && (opportunity >= 14)) // midfielder assist
                        return this.getPlayerByPosition("Midfield");
                else // defenders assist
                        return this.getPlayerByPosition("Defensive");
        }
/*-------------------------------------------------------------------------*/
        public String getPlayerByPosition(String position){  // Offensive or Midfield or Defensive
Player
                do{
                        int i = (int)(Math.random()*16);
                        if(position == "Offensive"){
                                if(players.get(i) instanceof OffensivePlayer)
                                        return players.get(i).getName();
                        }
                        else if(position == "Midfield"){
                                if(players.get(i) instanceof MidfieldPlayer)
                                        return players.get(i).getName();
                        }
                        else if(position == "Defensive"){
                                if((players.get(i) instanceof DefensivePlayer) && !(players.get(i)
instanceof Goalkeeper)) // a revoir pour les assists
                                        return players.get(i).getName();
                        }/*
                        else{
                                System.out.println("Error - position unclassified");
                                return "";
                        }       */
                }while(true); // a revoir
        }
/*-------------------------------------------------------------------------*/
        public String getTeamName(){return name;}
/*-------------------------------------------------------------------------*/
        public int getIndex(){return index;}
/*-------------------------------------------------------------------------*/
        public void setPlayHome(boolean bool){
```

```
		playHome = bool;
	}
/*---------------------------------------------------------------------------*/
	public boolean getPlayHome(){
		return playHome;
	}
/*---------------------------------------------------------------------------*/
	@Override
	public String toString(){
		String chaine = "Equipe "+name+":\n";
		for(int i=0; i < players.size(); i++){
			chaine += players.get(i)+"\n";
		}

		return chaine;
/*---------------------------------------------------------------------------*/
	}
}



import javax.swing.*;
import javax.swing.border.TitledBorder;

public class TeamRankings extends JPanel{
	private static final long serialVersionUID = 1L;
/*---------------------------------------------------------------------------*/
  public TeamRankings(Championnat championnat){
	setBorder(BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
			" ======== Barclay's Premier League Table ========= ",
TitledBorder.CENTER, TitledBorder.TOP));

	String [][] data = championnat.setStringClassement();
	String [] header = {"Team","Played","Win","Draw","Loose","Points"};
	JTable teamTable = new JTable(data, header);

	add(new JScrollPane(teamTable));
  }
/*---------------------------------------------------------------------------*/
}



import java.awt.*;
import java.util.concurrent.TimeUnit;

import javax.swing.*;

public class TeamsWindow extends JFrame implements Files{
	private static final long serialVersionUID = 1L;
	private Dimension currentScreenSize;
/*---------------------------------------------------------------------------*/
```

```
	public TeamsWindow(String result) {
		currentScreenSize = Toolkit.getDefaultToolkit().getScreenSize();

		new JFrame();
		setSize(1000, 750);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		getContentPane().setLayout(null);
		setLocation(0, (int) currentScreenSize.getHeight() / 12);

		JLabel twoTeams = new JLabel("twoTeams");
		twoTeams.setHorizontalAlignment(SwingConstants.CENTER);

		Image img2 = new
ImageIcon(this.getClass().getResource(imgGameLocation)).getImage();

		JTextArea InfoTeams = new JTextArea();

		InfoTeams.setBackground(new Color(202, 235, 252));
		InfoTeams.setForeground(new Color(0, 102, 0));
		InfoTeams.setFont(new Font("Comic Sans MS", Font.PLAIN, 22));

		InfoTeams.setText(result);

		InfoTeams.setBounds(350, 576, 320, 70);
		getContentPane().add(InfoTeams);

		twoTeams.setIcon((Icon) new ImageIcon(img2));

		twoTeams.setBounds(0, 0, 1000, 750);
		getContentPane().add(twoTeams);

		setVisible(true);
		try {
			TimeUnit.SECONDS.sleep(1);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		setVisible(false);
	}
/*-------------------------------------------------------------------------*/
}



import java.awt.Image;

import javax.swing.Icon;
import javax.swing.ImageIcon;

import javax.swing.JLabel;

public class WelcomePage extends JLabel implements Files{
```

```
  private static final long serialVersionUID = 1L;
/*-------------------------------------------------------------------------*/
  public WelcomePage() {

                    Image img1 = new
ImageIcon(this.getClass().getResource(imgChampionshipLocation)).getImage();
    setIcon((Icon) new ImageIcon(img1));
  }
/*-------------------------------------------------------------------------*/
}
```