1866 AUB

Faculty of Arts & Sciences
Department of Computer Science
CMPS 200—Introduction to Programming
Final Exam (2 hours)

UEFA Champions League 2016 (100%)

A soccer tournament consists of a set of games, where in each game two soccer teams compete against each other, and eventually the team with the highest scoring record in all the games is crowned as the champion. In this problem, you have to implement the set of classes that simulates a soccer tournament. Three classes in total should be implemented, `Player`, `Team`, and `Tournament`. A forth class called `UEFA2016` is given to you; you should simply run it to test your code. The classes that you should implement are described hereafter:

- The `Player` class holds details about a single player. A Player is represented by six private fields:
  - An integer called `jerseyNumber` to record the jersey number of the player
  - A string called `position` to indicate the position of the player. It can take 4 values:
    - "GK" if the player is a goalkeeper;
    - "DF" if the player is a defender;
    - "MF" if the player is a midfielder;
    - "FW" if the player is a forward.
  - A string called `name` to hold the name of the player
  - An integer called `scoredGoals` to track the number of goals that the player (will) has scored. It is updated based on the value of `scoringProb` as described next.
  - A double called `scoringProb` which is a decimal number in the range [0, 1). This number is used as a threshold to know whether the player will score a goal or not. For example, if the `scoringProb` is 0.9, you can generate a random number in the range [0, 1), and then assume that the player will score a goal if the random number is less than or equal to 0.9; in which case, you must increment the instance variable `scoredGoals` by 1, otherwise the player will not score (keep `scoredGoals` intact).
  - A boolean called `isPlaying` to indicate whether the player is playing in the game or is a substitute.

The behavior (methods) of class Player is as follows:

`public class Player`	
	// a constructor that sets the fields of the Player object to the passed variables *//* `scoredGoals` *is set by default to 0* *//* `isPlaying` *is set by default to false.* `Player(int jerseyNumber, String position, String name, double scoringProb)`
`int`	*// getter for* `scoredGoals` `getScoredGoals ()`
`void`	*// this method calculates/updates the value of* `scoredGoals` *// based on the value of* `scoringProb`. *//* `scoredGoals` *is incremented by 1 only if a* *// generated random number (in the range* `[0,1.0)`*) is less than or equal to* *//* `scoringProb` `calcScoredGoals()`

`double`	*// getter for* `scoringProb` `getScoringProb()`
`void`	*// setter for* `scoringProb` `setScoringProb(double  scoringProb)`
`boolean`	*// getter for* `isPlaying` `getIsPlaying()`
`void`	*// setter for* `isPlaying` `setIsPlaying(boolean  isPlaying)`
`String`	*// getter for* `position` `getPosition()`
`String`	*// Returns a string representation of the Player, for example:* *//* `[Romario, 11, FW, 2, substitute]` *// represents, the player name, jersey number, position,* *// scoredGoals, and whether the player is "*`playing`*" or is a "*`substitute`*"* `toString()`

- The `Team` class holds details about a team of players. A `Team` is represented by six private fields:
  - A string called `teamName` to hold the name of the team
  - A string called `country` to hold the home country of the team
  - An array of `Player` objects called `players` comprising the players of the team
  - A string called `formation` to indicate the formation of the team. The `formation` string is composed of three numbers separated by dashes (no spaces), where each number is a single digit, and the three numbers add up to 10. The first integer indicates the number of defenders in the team, the second integer indicates the number of midfielders and the third integer indicates the number of forwards. Every team has only one goalkeeper. For example if the value of `formation` is "4-3-3", the team is composed of 4 defenders, 3 midfielders, 3 forwards, and (**always**) one goalkeeper
  - An integer called `scoringAbility` that indicates the scoring ability of the team. To calculate the scoring ability of the team, you will have to add up the `scoredGoals` of every **playing** player in the team
  - A boolean called `won` to indicate whether the team has won the game it previously played or not. By default `won` is set to `true`

The behavior (methods) of class `Team` is as follows:

`public class Team`	
	// Constructor that takes as parameter the Scanner object, and initializes the *// fields:* `teamName`, `country`, `players`, *and* `formation`. *// The input file includes all the teams (back-to-back) such that each team is* *// organized as follows:* *// on the first line, the name of the team is specified* *// on the second line, the country of the team is specified* *// on the third line, the number of players (say N) in the team is specified* *// The following N lines represent the players of the team; every line specifies* *// information about a single player as follows:* *//* `jerseyNumber position name scoringProb` *// the following line contains the team* `formation` *// the file contains even number of teams* `public Team(Scanner input)`

`String`	*// getter for* `country` `getCountry()`
`String`	*// getter for* `teamName` `getTeamName()`
`int`	*// getter for* `scoringAbility` `getScoringAbility()`
`boolean`	*// getter for* `won` `getWon()`
`void`	*// setter for* `won` `setWon(boolean won)`
`void`	*// this method parses the* `formation` *string and decides which players are* *// playing; it also finds the number of goals each player has scored (i.e., using the* *// method* `calcScoredGoals`*). For example, if the formation string is* *//* `4-3-3`*, the first 4 defenders in the array* `players` *will be playing, the first 3* *// midfielders, the first 3 forwards, and the first goalkeeper. For every player that* *// is set to "playing", the corresponding* `scoredGoals` *should be updated.* `createFirstEleven ()`
`void`	*// This method finds the scoring ability of the team and assigns the result to the* *// field variable* `scoringAbility` `scoringAbility()`
`Player`	*// This method loops over the* `players` *array, and returns the* ***first*** `Player` *with* *// the max number of goals i.e., the max* `scoredGoals`*.* *// NOTE: the player should be "playing".* `getBestPlayer()`
`String`	*// this method return a String representation of the Team object as follows:* *//* `teamName` *followed by* `country`*, followed by* `scoringAbility`*, followed* *// by the playing players (each player on a line); see Sample Run below as example.* `toString()`

- The `Tournament` class holds details about the tournament. A `Tournament` is represented by one private field, which is an array of type Team (call the array `teams`). The behavior (methods) of class `Tournament` is set as follows:

`public class Tournament`	
	// a constructor that takes as parameter an integer number indicating the number *// of teams that will be playing in the tournament. The constructor should read* *// from the file called* `Teams.txt`*, the information about the teams and* *// consequently initialize the array* `teams` `public  Tournament(int numOfTeams)`
`void`	*// this method mainly does four things: (1) creates the formation of t1 and t2 i.e.,* *// which players will be playing in the game, (2) calculates the scoring ability of t1* *// and t2, (3) depending on the scoring ability of every team, one of the two teams* *// is set as Winner (if there is a tie the first team wins), and (4) prints on the* *// Console who won, for example* `RealMadrid beats PSG!` `play(Team t1, Team t2)`

void	// this method simulates the tournament. Initially all teams in the `teams` array are
	// set to be winners (`won = true`) by default. Starting left to right,
	// every two consecutive teams who won a game will compete against each other.
	// When the first round of games is over, a second round of games is
	// played where the winners of the previous round compete against each other.
	// This process continues until one team wins the tournament (check sample run).
	// Once done, print on the console the winning team and the best player in the
	// team, which is declared as the "Player of the tournament"
	// PS: You may assume that the number of teams is always equal to 8 (per the input
	// file `Teams.txt`)
	run()

- The `UEFA2016` class is used to test your code. The UEFA2016 class and the corresponding sample run are given as follows, respectively:

```
import java.io.*;

public class UEFA2016 {
   public static void main(String[] args) throws FileNotFoundException{
          Tournament championsLeague = new Tournament(8);
          championsLeague.run();
   }
}
```

Sample Run:

```
Barcelona beats Chelsea!
RealMadrid beats PSG!
Juventus beats ManCity!
ACMilan beats BayernMunich!
RealMadrid beats Barcelona!
Juventus beats ACMilan!
RealMadrid beats Juventus!

The Championship goes to:
RealMadrid,Spain,20
[Navas, 1, GK, 0, playing]
[Varane, 2, DF, 0, playing]
[Pepe, 3, DF, 2, playing]
[Ramos, 4, DF, 2, playing]
[Nacho, 6, DF, 2, playing]
[Ronaldo, 7, FW, 3, playing]
[Kroos, 8, MF, 3, playing]
[Benzema, 9, FW, 3, playing]
[James, 10, MF, 2, playing]
[Bale, 11, FW, 2, playing]
[Casemiro, 14, MF, 1, playing]

Player of the tournament: [Ronaldo, 7, FW, 3, playing]
```

Notes and Submission Instructions

- Your exam submission must consist of a single zip archive named **s#_finalExam_*netid*** that contains your properly commented **three** source files (.java files) only (**Player.java**, **Team.java**, and **Tournament.java**). No other files will be accepted. We will compile and run your programs.
- Give meaningful names to methods and variables in your code.