Unit 1. Final exercise

League Manager

Service and Process Programming

Arturo Bernal
Nacho Iborra

IES San Vicente

Index of contents

1. Introduction

In this final exercise we are going to manage the statistics of some sports league. To be more precise, we are going to deal with the Spanish National Football and Basketball Leagues through a Java console application. We will load the team data and results from text files, and then we will be able to add new results and show the league tables with a list of filters that we can choose.

Here you can see a couple of screenshots of our application:

```
1. Show football league table
2. Show basketball league table
3. Add new football result
4. Add new basketball result
5. Show teams with more goals for than against for football league
6. Show scored points average for basketball league
7. Show points difference in football league
0. Exit

Choose an option:
```

```
                              Total    W    D    L   GF   GA  Pts
FCB  F.C. Barcelona               2    2    0    0    8    2    6
ATM  At. Madrid                   2    1    1    0    3    2    4
VIL  Villarreal                   2    1    1    0    3    1    4
CEL  Celta                        1    0    1    0    1    1    1
EIB  Eibar                        1    0    1    0    1    1    1
RSO  R. Sociedad                  1    0    1    0    1    1    1
VAL  Valencia                     1    0    1    0    1    1    1
ALA  Alavés                       0    0    0    0    0    0    0
ATH  Ath. Bilbao                  1    0    0    1    1    2    0
BET  Betis                        0    0    0    0    0    0    0
DEP  Deportivo                    0    0    0    0    0    0    0
ESP  Espanyol                     0    0    0    0    0    0    0
GRA  Granada                      0    0    0    0    0    0    0
LPA  Las Palmas                   0    0    0    0    0    0    0
LEG  Leganés                      1    0    0    1    0    2    0
MGA  Málaga                       0    0    0    0    0    0    0
OSA  Osasuna                      1    0    0    1    1    5    0
RMA  R. Madrid                    0    0    0    0    0    0    0
SEV  Sevilla                      1    0    0    1    1    3    0
SPO  Sporting                     0    0    0    0    0    0    0
```

1.1. Creating the project and source packages

Create a Java project called **LeagueManager**. Create a package called *unit1.leaguemanager* inside the source folder, with a main class in it (call it *LeagueManager*). Then, create two subpackages: one called *unit1.leaguemanager.model* and another one called *unit1.leaguemanager.utils*. We will populate these packages later.

2. The model

The model of an application is composed by all the classes that store the information of this application. In our case, we are going to work with teams and their data, so we will define a *Team* class, and two subclasses to manage the particular information of the two leagues we are going to store: football teams and basketball teams. Besides, we will define a class to store the information of every match between two given teams. All these classes must be created inside the ***unit1.leaguemanager.model*** package. Let's go!

2.1. The Team class

First of all, define an abstract class called *Team.* It will have the following elements:

- **Attributes** (protected):
  - Team code (a String of three characters that will identify a team among all the teams of the same league)
  - Team name (String)
  - Total number of matches won (int)
  - Total number of matches lost (int)
- **Constructors**:
  - A constructor with only the team name and code (all the *int* attributes will be set to 0)
- **Methods**
  - Getter and setter for each attribute
  - A method called *getTotalPlayed* that will return the total number of matches played (*matches won + matches lost*)
  - An abstract method called *getTotalPoints* that will return the total number of points won by the team. It will depend of the team type, since points are calculated differently in football and basketball.
  - An abstract, void method called *print* that will print a line with the team information, as we will see later.
  - You can add two additional methods that will be useful: one called *incMatchesWon* that will increase (++) the attribute of total number of matches won, and another one called *incMatchesLost* that will increase the total number of matches lost.

2.2. Team subclasses

We will define two subclasses of *Team* class: one to store specific information of football teams, and another one for basketball teams. **Try to re-use as much code as possible from the superclass** (i.e. do not duplicate your code if not necessary).

2.2.1. FootballTeam

Define a class called *FootballTeam* that extends *Team* class. We will add some new (private) **attributes**:

- Total number of matches drawn (int)
- Total number of goals for (int)
- Total number of goals against (int)

Define a **constructor** with two parameters (team code and name).

Regarding **methods:**

- Add getters and setters for the three new attributes
- Override the method *getTotalPlayed* and add to the result given by the superclass the total number of matches drawn (the final result should be the total number of matches won + lost + drawn).
- Override the abstract method *getTotalPoints*. In football, the total number of points for a team is calculated as *3 * matches won + matches drawn*
- Override the abstract method *print*. We are going to print a line with the following format:

  ```
  CODE NAME PLAYED WON DRAWN LOST FOR AGAINST POINTS
  ```

  Each attribute must be spaced a given number of spaces. For instance, for this team:

  ```
  FCB  F.C. Barcelona                  2    2    0    0    8    2    6
  ```

```
Spaces> 5               30                5    5    5    5    5    5    5
```

2.2.2. BasketballTeam

Define a class called *BasketballTeam* that extends *Team* class. We will add some new (private) **attributes**:

- Total number of points for (int)
- Total number of points against (int)

Define a **constructor** with two parameters (team code and name).

Regarding **methods:**

- Add getters and setters for the two new attributes
- Override the abstract method *getTotalPoints*. In basketball, the total number of points for a team is calculated as *2 * matches won*
- Override the abstract method *print*. We are going to print a line with the following format:

  ```
  CODE NAME PLAYED WON LOST FOR AGAINST POINTS
  ```

  Each attribute must be spaced a given number of spaces. For instance:

  ```
  VAL  Valencia Basket                 2    2    0  178  152    4
  ```

```
Spaces> 5               30                5    5    5    5    5    5
```

NOTE: See the official documentation of the *System.out.printf* or *format* methods to see how to format and tabulate text to display it properly in the console.

2.3. Storing match information

To store the information of every league match, we are going to define a *Match* class. It will use *generics* (see Section 4 of the contents for the 1st week) to define a match depending of the team type (either *FootballTeam* or *BasketballTeam*).

```
public class Match<T extends Team>
{
    ...
```

The class will have four private **attributes**:

- The local team (of type T)
- The visitor team (of type T)
- The local score (either goals or points, it doesn't matter. It will be an integer)
- The visitor score (also an integer)

There will be one **constructor** with all the attribtues

Regarding **methods**:

- Define a getter and setter for every attribute
- It might be useful a method that, given the match and its score, updates the local team and visitor team information (increases the total number of matches won/lost/drawn, and increases the total number of goals/points for and against). You can call this method *updateTeamsData*.

3. Reading and writing files

Let's see which information are we going to store in the files and how to read and write it.

3.1. File information

For each league, we will manage two text files: one containing the team names and codes, and another one containing the match results.

3.1.1. Team names and codes

For instance, for the football league, we can store the team names and codes in a file called *football.txt*, with the following format:

```
RMA;R. Madrid
FCB;F.C. Barcelona
SEV;Sevilla
VIL;Villarreal
...
```

Note that we separate the team code and name with a semicolon (;).

3.1.2. Match results

For the results file, we can store them in a file called *football_stats.txt* or *football_results.txt* (as you prefer), with the following format:

```
FCB 3 SEV 1
VIL 2 LEG 0
EIB 1 RSO 1
FCB 5 OSA 1
...
```

Note that we place the local team code, its score, and then the visitor team code and its result.

3.2. Static constants

It may be useful to define some static constants storing the file names from which we will read and write information. For instance:

```
private static final String FOOTBALL_TEAMS_LIST_FILE = "football.txt";
private static final String FOOTBALL_TEAMS_STATS_FILE = "football_stats.txt";
private static final String BASKETBALL_TEAMS_LIST_FILE = "basketball.txt";
private static final String BASKETBALL_TEAMS_STATS_FILE = "basketball_stats.txt";
```

As you can see, there will be a couple of constants for each league (football and basketball) pointing at the corresponding file name.

3.3. FileUtils class

Create a class called *FileUtils* in the *unit1.leaguemanager.utils* package. We are going to define some static methods inside to read and store information from the files.

3.3.1. Reading team codes and names

We are going to store the teams in a generic list. Define a method called *loadFootballTeams* that returns a list of *FootballTeam* objects, and another one called *loadBasketballTeams* that returns a list of *BasketballTeam* objects:

```
public static List<FootballTeam> loadFootballTeams() { … }
public static List<BasketballTeam> loadBasketballTeams() { … }
```

These methods will read the corresponding text file containing the team codes and names, and create *FootballTeam*s or *BasketballTeams* with these data, returning a list with all of them.

3.3.2. Reading match results

In the same way, define a method called *loadFootballMatches* and another one called *loadBasketballMatches*. They return a list of matches (*Match* class) of the corresponding type (football or basketball):

```
public static List<Match<FootballTeam>>
   loadFootballMatches(List<FootballTeam> teamList)
public static List<Match<BasketballTeam>>
   loadBasketballMatches(List<BasketballTeam> teamList)
```

These methods will read the corresponding text file containing the match results. They have the original team list as a parameter, so that they can check if the teams exist in the team list, and update the teams data.

3.3.3. Storing match results

Every time we close the application (choosing the appropriate option menu, as we will see later), we must save the lists of matches in their corresponding files. So we should define two methods: *saveFootballMatches* and *saveBasketballMatches* that will receive the list of matches as a parameter, and store them in the corresponding file with the appropriate format (see section 3.1.2 above)

4. The main program

When we launch the application, all the files must be loaded in their corresponding collections at the beginning:

- A list of football teams
- A list of basketball teams
- A list of football matches
- A list of basketball matches

The teams lists must be ordered by the total number of points in descending order. To do this, you must implement a *Comparator* by using a lambda expression, and order the teams first by their total number of points. To sort the list whenever you want, you can use the static method from the *Collections* class:

```
Collections.sort(List, Comparator)
```

4.1. The main menu

After loading the data, a menu will be displayed with the following options:

```
1. Show football league table
2. Show basketball league table
3. Add new football result
4. Add new basketball result
5. Show teams with more goals for than against for football league
6. Show scored points average for basketball league
7. Show points difference in football league
0. Exit
```

When the user chooses any option from 1 to 7, an action will be performed and then the menu will be displayed again. If user chooses 0, the program will exit. Any other option will show an appropriate error message in the console, and show the menu again.

4.2. Show full table league

Options 1 and 2 will show the whole list of *Team* objects (depending on the option itself, we will show either football teams or basketball teams), ordered by total number of points in descendant order, in the following format:

		Total	W	D	L	GF	GA	Pts
FCB	F.C. Barcelona	2	2	0	0	8	2	6
ATM	At. Madrid	2	1	1	0	3	2	4
VIL	Villarreal	2	1	1	0	3	1	4
CEL	Celta	1	0	1	0	1	1	1
EIB	Eibar	1	0	1	0	1	1	1
RSO	R. Sociedad	1	0	1	0	1	1	1
VAL	Valencia	1	0	1	0	1	1	1
ALA	Alavés	0	0	0	0	0	0	0
ATH	Ath. Bilbao	1	0	0	1	1	2	0
BET	Betis	0	0	0	0	0	0	0
DEP	Deportivo	0	0	0	0	0	0	0
ESP	Espanyol	0	0	0	0	0	0	0
GRA	Granada	0	0	0	0	0	0	0
LPA	Las Palmas	0	0	0	0	0	0	0
LEG	Leganés	1	0	0	1	0	2	0
MGA	Málaga	0	0	0	0	0	0	0
OSA	Osasuna	1	0	0	1	1	5	0
RMA	R. Madrid	0	0	0	0	0	0	0
SEV	Sevilla	1	0	0	1	1	3	0
SPO	Sporting	0	0	0	0	0	0	0

NOTE: use the *print* method of every team to print its own line (see section 2.2) and an appropriate *printf* instruction to print the table header.

4.3. Add new result

Options 3 and 4 lets us add a new match result to either the football or basketball league, respectively. When choosing any of these options, the following information message must appear:

```
Enter the result in the following format:
LOCAL_CODE LOCAL_SCORE VISITOR_CODE VISITOR_SCORE
Example:
FCB 3 SEV 1
```

Then, the user will type a new result with the same format that the example given. For instance:

```
VIL 3 CEL 1
```

This line must be properly parsed to add a new match to the corresponding list. The team data (matches won/lost/drawn and total goals or points) must be automatically updated, and the corresponding team list must be reordered according to this new result (a new call to *Collections.sort* must be executed after every new result).

If there is any error choosing any team or entering the result, no new result will be added, and an error message will be displayed. In any case, the main menu will be shown again.

4.4. Filters

Options 5 to 7 of the main menu must be implemented using Java 8 streams or lambda expressions (depending on the filter itself).

4.4.1. Show teams with more goals for than against for football league

This filter will list the teams of the football league where the total number of goals for is greater than the total number of goals against, ordered by total number of points in descendant order. Use a simple stream to implement this filter.

4.4.2. Show scored points average for basketball league

This filter will calculate the "points for" average for the basketball league, and show this data (an integer or float, as you prefer) in the screen. To do this, map the stream to the attribute "goals for", and calculate the average (see section 10.3 of the unit contents (part III)).

4.4.3. Show points difference in football league

This filter will calculate and show the difference between the total number of points of the first and the last team in the list. Although this operation can be calculated easily, we ask you to implement it with a *BiFunction.* Define a BiFunction that takes two teams as parameters, and returns the points difference between them.

> See Annex IV to learn more about the *BiFunction* interface and how to use lambda expressions to implement it.

5. Optional improvements

You can also implement this suggested improvements. Each one can increase up to 1 point your final mark for this exercise.

5.1. Controlling new results

In the basic implementation of this feature (menu options #3 and #4), you are not asked to control if there is already a result entered for the match selected. In other words, we could enter multiple results for the same match.

Try to control that only one result is entered for each match, taking into account that the match *F.C. Barcelona – Villarreal*, for instance, is different than the match *Villarreal – F.C. Barcelona*, since each team acts as local in one match, and as visitor in the other one.

> You may need to use a different collection type (instead of a list) to store the matches and check for duplicates.

5.2. New filter: teams going down

Add a new option to the main menu (it would be option number 8) with the text "*Show teams that will go down to 2nd division*". It will be implemented with a stream that filters the last 3 teams, maps their names and returns a *String* with the last 3 team names, separated by commas. You can also use a *Function* to filter the teams and return the *String* with their names. Use a *StringJoiner* (see Annex IV) to create the resulting *String*

5.3. Showing team information

Add a new option to the main menu (at the end) called "*Show team results*". If we choose this option, we will be asked to select the league (football or basketball) and the team (entering its code), and then we will use a stream to filter all the matches played by this team (either as local or visitor), and print the results, indicating in each match if the team won or lost this match. For instance:

```
Select league (1 - Football, 2 – Basketball):
1
Enter team code:
RSO
Showing matches for Real Sociedad:
Real Sociedad 1 Eibar 1 (Drawn)
Villarreal 2 Real Sociedad 1 (Lost)
Real Sociedad 3 Granada 1 (Won)
...
```

6. Evaluation rules

6.1. Compulsory part

To get your final mark, the following rules will be applied:

- Class structure (model package), with the *Team* class and its subclasses, and the *Match* class implemented with generics: 1 point. Every unnecessary duplication of code will make you lose part of your mark (try to use the *super* methods as much as possible).
- Class *FileUtils* with appropriate methods to read and store information from files: 1 point
- *Main menu* with the corresponding options, controlling that the user chooses a correct option every time it is displayed: 0,5 points
- Displaying the table league(s) ordered and formatted properly (menu option #1 and #2): 2 points
- Adding new results to the list, and controlling every possible error while adding the result (menu options #3 and #4): 1,5 points
- Showing football teams with more goals for than against (menu option #5): 1 point
- Showing the average of scored points (menu option #6): 1 point
- Showing the points difference (menu option #7): 1 point
- Code documentation (Javadoc comments for every class and public method or constructor), cleanliness and efficiency: 1 points

6.2. About the optional improvements

If you decide to implement any optional improvement for this exercise, keep in mind that:

- If your compulsory part is lower than 10, you can get up to 10 points by implementing one (or many) of the optional improvements (up to 1 point per improvement)
- If your compulsory part is perfect (10 points), you can get up to 11 points by implementing ALL the optional improvements.