A picture containing text

Description automatically generatedA picture containing shape

Description automatically generated

**INFORMATICS INSTITUE OF TECHNOLOGY  
DEPARTMENT OF COMPUTING**

**Module:** 5COSC007C.1  
**Object Oriented Programming  
Module Leader:** Mr. Guhanathan Poravi

**Name**: Radhika Ranasinghe

**Uow**: w1761764

**IIT ID**: 2018199

**Acknowledgement**

First and foremost, I would like to express my deep and sincere gratitude to my lecturers, Mr. Guhanathan Poravi, Mr. Deshan Sumanathilaka and Mr. Iresh Bandara, for providing invaluable guidance throughout this coursework. Their dynamism, vision, sincerity, and motivation have deeply inspired me during this project.

I am extremely grateful to my parents for their love, care and sacrifices made throughout educating and preparing me for my future.

Last but not the least, I would like to thank and acknowledge all my colleagues and friends who have helped me immensely.

**Table of Contents**

[Chapter 1 Introduction 1](#_Toc60640230)

[Problem Specification 1](#_Toc60640231)

[Chapter 2 Design 3](#_Toc60640232)

[Class Diagrams 3](#_Toc60640233)

[Use Case Diagrams 4](#_Toc60640234)

[Chapter 3 Implementation 6](#_Toc60640235)

[Java Source Code 6](#_Toc60640236)

[Entities package 6](#_Toc60640237)

[Utils Package 53](#_Toc60640238)

[Controllers Package 54](#_Toc60640239)

[Angular Source Code 59](#_Toc60640240)

[Dashboard-home 59](#_Toc60640241)

[League-table.component 62](#_Toc60640242)

[Nav-side-bar.component 67](#_Toc60640243)

[Played-matches.component 69](#_Toc60640244)

[Random-match.component 77](#_Toc60640245)

[App.component 82](#_Toc60640246)

[Root 87](#_Toc60640247)

[Chapter 4 Testing 89](#_Toc60640248)

[JUnit Source Code 89](#_Toc60640249)

[Entities 89](#_Toc60640250)

[Controllers 95](#_Toc60640251)

[Chapter 5 Assumptions 101](#_Toc60640252)

[Chapter 6 Conclusion 102](#_Toc60640253)

[Chapter 7 References 103](#_Toc60640254)

**Table of Figures**

[Figure 1: Class Diagram 3](file:///C:\Users\ASUS\Documents\w1761764\OOP.docx#_Toc60650945)

[Figure 2: Use Case diagram for CLI 4](file:///C:\Users\ASUS\Documents\w1761764\OOP.docx#_Toc60650946)

[Figure 3: Use Case Diagram GUI 5](#_Toc60650947)

[Figure 4: Test results of the PremierLeagueManagerTest.java 94](file:///C:\Users\ASUS\Documents\w1761764\OOP.docx#_Toc60650948)

[Figure 5: Results of the PremierLeagueControllerTest.java 100](#_Toc60650949)

**List of Tables**

**No table of figures entris found.**

# Introduction

## Problem Specification

The problem description states to create an application using java which simulates the manipulation of the English Premier League Championship. The student is expected to design a solution for this system considering Object Oriented Principles. The design should comprise of class diagrams, two use case diagrams for the system.

The user should be able to do the following from the command line interface

* Create a new football club and add it in the premier league.
* Delete or relegate and existing club from the premier league
* Display statistic of a selected club
* Display the Premier League Table, i.e.: display all the clubs playing in the premier league and their statistics in the descending order etc.
* Add a played match with its scores and its date so that the statistics of the two clubs involved and the premier league table are updated automatically.
* Saving all the data/information entered by the user. [Use of database is prohibited]
* The next time the application run it should retrieve all the information saved in the file and continue operations based on that with user being able to enter new information or changer the exiting information.

Designing and implement a Graphical user interface which will do the following

* Display a table of clubs and their statistics in the descending order of points.
  + Give the user possibility to sort the table according to the goals scored (descending order).
  + Give the user possibility to sort the table according to largest number of the wins scored (descending order).
  + A button which generates one random match played match between two randomly chosen clubs and it automatically updates the premier league table by add the match (points, score, statistics). The user should be able to see the match generated, in order to be able to verify the correctness of the code for the updated information of the table.
* Add a button to display all the played matches sorted in ascending order of date played, (both randomly generated or manually entered using text menu).
* Add a button and a textbox which can be used to search for all search matches played in a given date. The full details of the matches should be displayed.

# Design

## Class Diagrams

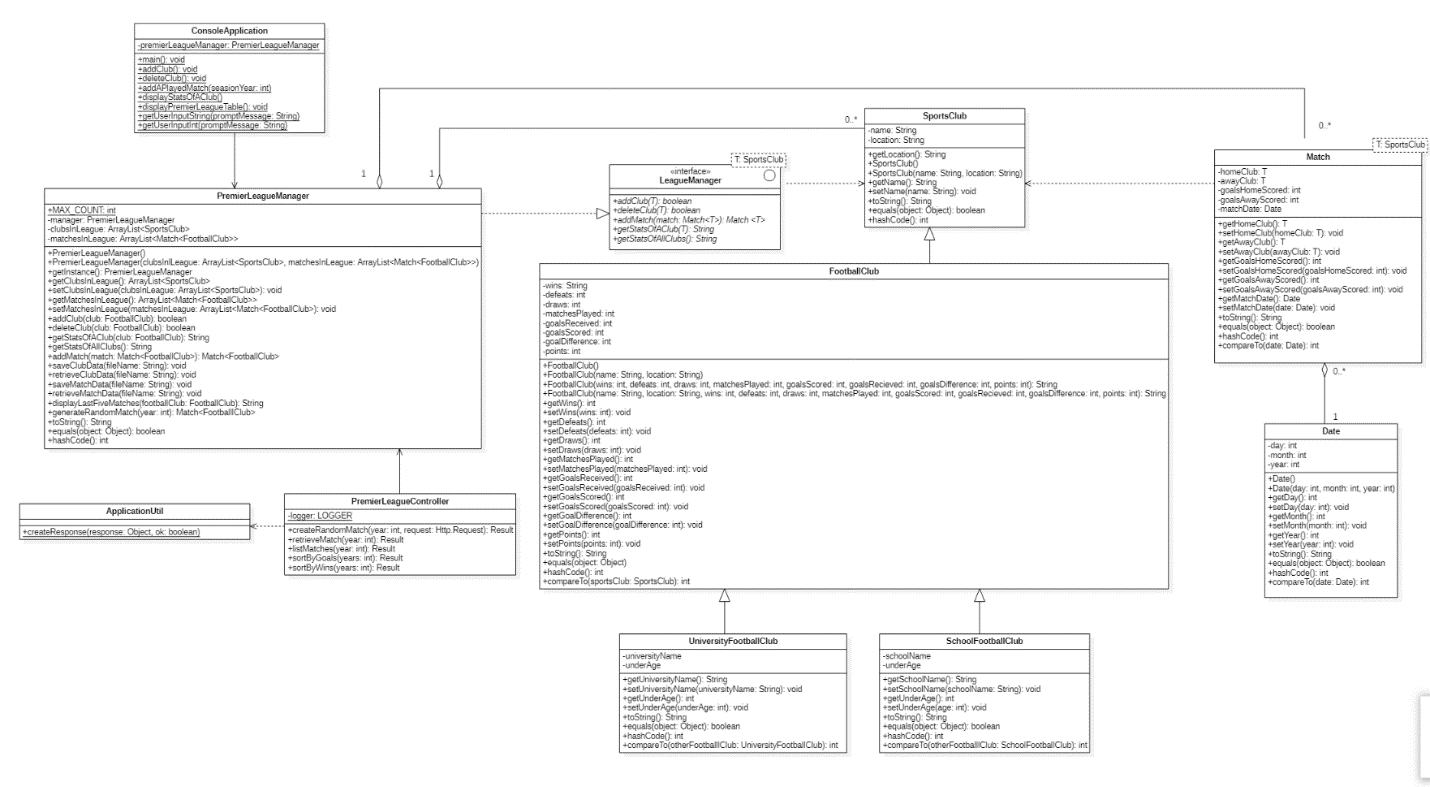


Figure 1: Class Diagram

## Use Case Diagrams

Figure 2: Use Case diagram for CLI

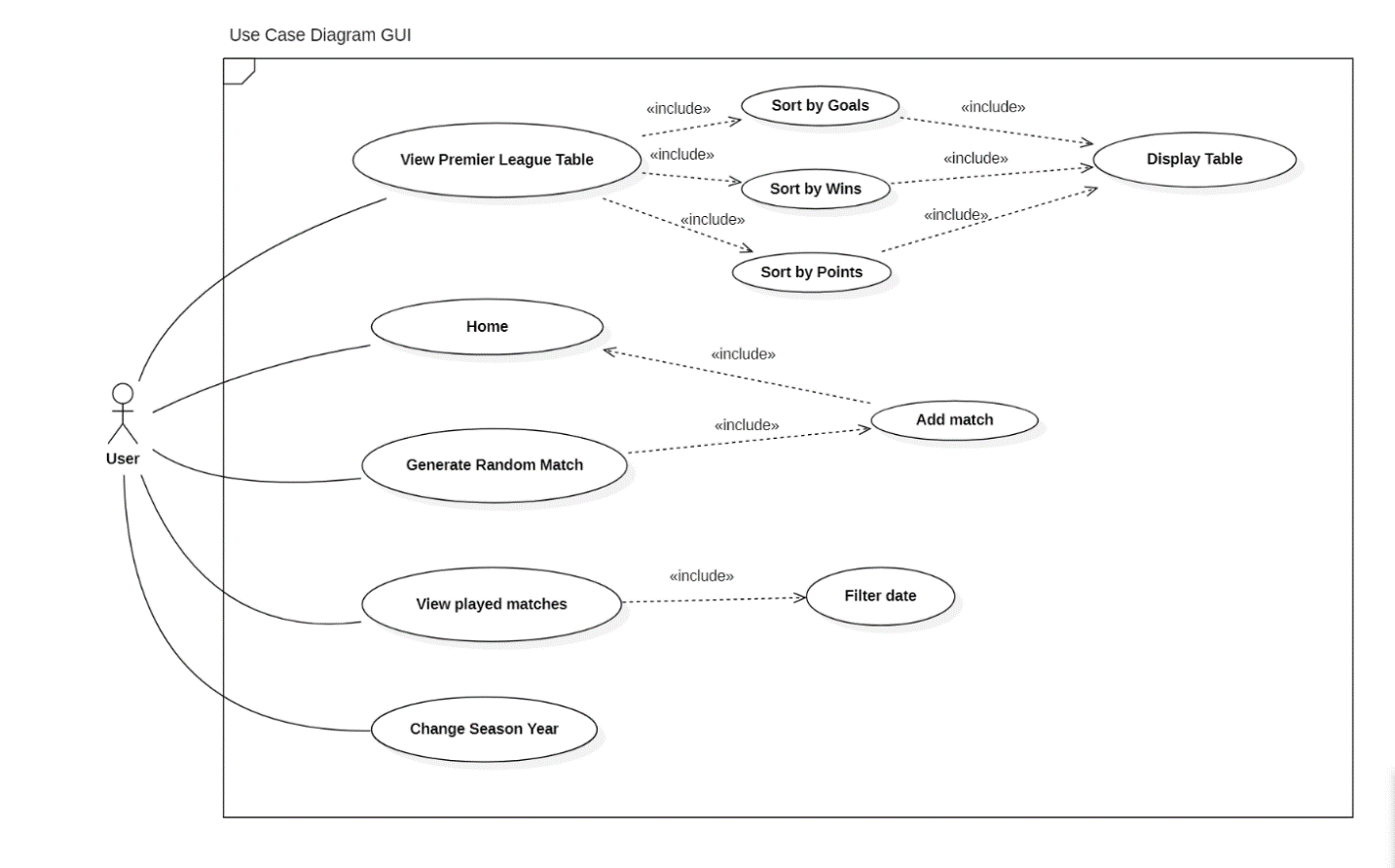


Figure 3: Use Case Diagram GUI

# Implementation

## Java Source Code

### Entities package

#### **LeagueManager.java**

App/entities/LeagueManager.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
9. **package** entities;
11. /\*\*
12. \* This is the Interface LeagueManager which contains all the abstract methods of the functionalities done by a league.
13. \*
14. \* @author Radhika Ranasinghe
15. \* @version 1.0
16. \* @since 2020-11-15
17. \*/
18. **public** **interface** LeagueManager<T **extends** SportsClub> {
20. /\*\*
21. \* Method that adds a sports club to the league
22. \*/
23. **boolean** addClub(T club);
25. /\*\*
26. \* Method that deletes a sports club from the league
27. \*/
28. **boolean** deleteClub(String clubName);
30. /\*\*
31. \* Method that adds a played match to the league
32. \*
33. \* @param match the match that is to be added to league
34. \* @return match object containing the match added  to the league
35. \*/
36. Match<T> addMatch(Match<T> match);
38. /\*\*
39. \* Method that returns the stats of the given saved club.
40. \*
41. \* @param club the club which the stats belong to
42. \*/
43. String getStatsOfAClub(T club);
45. /\*\*
46. \* Method that returns the stats of all the clubs saved
47. \*
48. \* @return ArrayList containing Strings of stats of all the clubs
49. \*/
50. String getStatsOfAllClubs();
52. }

#### **SportsClub.java**

App/entities/SportsClub.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
9. **package** entities;
11. **import** com.fasterxml.jackson.databind.annotation.JsonDeserialize;
13. **import** java.io.Serializable;
14. **import** java.util.Objects;
16. /\*\*
17. \* This is the abstract class SportsClub which includes Accessors and Mutators of the instance variables and
18. \* the toString method of the class.
19. \*
20. \* @author Radhika Ranasinghe
21. \* @version 1.0
22. \* @since 2020-11-15
23. \*/
24. @JsonDeserialize(as = FootballClub.**class**)
25. **public** **abstract** **class** SportsClub **implements** Serializable, Comparable<SportsClub> {
26. **private** String name;
27. **private** String location;

30. /\*\*
31. \* Default constructor of the abstract class SportClub
32. \*/
33. **public** SportsClub() {
34. }
36. /\*\*
37. \* Constructor of the abstract class SportsClub.
38. \*
39. \* @param name     name of the club
40. \* @param location location of the club
41. \*/
42. **public** SportsClub(String name, String location) {
43. **this**.name = name;
44. **this**.location = location;
46. }
48. /\*\*
49. \* Getter/Accessor of the name of the SportsClub
50. \*
51. \* @return the name of the SportsClub
52. \*/
53. **public** String getName() {
54. **return** name;
55. }
57. /\*\*
58. \* Setter/Mutator of the name of the SportsClub
59. \*
60. \* @param name the name to be set as the name of the SportsClub
61. \*/
62. **public** **void** setName(String name) {
63. **this**.name = name;
65. }
67. /\*\*
68. \* Getter/Accessor of the location of the SportClub
69. \*
70. \* @return the location of the SportClub
71. \*/
72. **public** String getLocation() {
73. **return** location;
74. }
76. /\*\*
77. \* Setter/Mutator of the location of the SportsClub
78. \*
79. \* @param location the location of the SportClub
80. \*/
81. **public** **void** setLocation(String location) {
82. **this**.location = location;
83. }
85. /\*\*
86. \* toString method of the class SportsClub
87. \*
88. \* @return a string containing all the instance variable with the respective instantiation
89. \*/
90. @Override
91. **public** String toString() {
92. **return** "SportsClub[" +
93. "name='" + name + '\'' +
94. ", location='" + location + '\'' +
95. ']';
96. }
98. /\*\*
99. \* Equals method of the SportsClub class
100. \*
101. \* @param o object containing any type
102. \* @return a boolean if the object is an instance of SportsClub
103. \*/
104. @Override
105. **public** **boolean** equals(Object o) {
106. **if** (**this** == o) **return** **true**;
107. **if** (!(o **instanceof** SportsClub)) **return** **false**;
108. SportsClub club = (SportsClub) o;
109. **return** getName().equals(club.getName()) && getLocation().equals(club.getLocation());
110. }
112. /\*\*
113. \* Hashcode method of the class SportsClub
114. \*
115. \* @return int containing the hashcode
116. \*/
117. @Override
118. **public** **int** hashCode() {
119. **return** Objects.hash(getName(), getLocation());
120. }
121. }

#### FootballClub.java

App/entities/FootballClub.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. **import** java.io.Serializable;
11. **import** java.util.Objects;
13. /\*\*
14. \* This is the concrete class FootballClub which extends SportsClub class and inherits those
15. \* instance variables and methods.
16. \* Also it includes Accessors and Mutators of the instance variables and
17. \* the toString method of the class.
18. \*
19. \* @author Radhika Ranasinghe
20. \* @version 2.0
21. \* @since 2020-11-15
22. \*/
23. **public** **class** FootballClub **extends** SportsClub **implements** Serializable {
24. **private** **int** wins;
25. **private** **int** defeats;
26. **private** **int** draws;
27. **private** **int** matchesPlayed;
28. **private** **int** goalsReceived;
29. **private** **int** goalsScored;
30. **private** **int** goalDifference;
31. **private** **int** points;
33. /\*\*
34. \* Default constructor of the Football class
35. \*/
36. **public** FootballClub() {
37. }
39. **public** FootballClub(String name, String location) {
40. **super**(name, location);
41. }
43. /\*\*
44. \* Constructor of the Football class
45. \*
46. \* @param wins           number of win achieved by the club
47. \* @param defeats        number of defeats accounted by the club
48. \* @param draws          number of draws achieved by the club
49. \* @param goalsScored    number of goals scored by the club
50. \* @param goalsReceived  number of goals received by the club
51. \* @param goalDifference number of goals difference
52. \* @param points         number of points club currently has
53. \* @param matchesPlayed  number of matches played by the club
54. \*/
55. **public** FootballClub(**int** wins, **int** defeats, **int** draws, **int** goalsScored, **int** goalsReceived, **int** goalDifference, **int** points, **int** matchesPlayed) {
56. **this**.wins = wins;
57. **this**.defeats = defeats;
58. **this**.draws = draws;
59. **this**.goalsScored = goalsScored;
60. **this**.goalsReceived = goalsReceived;
61. **this**.goalDifference = goalDifference;
62. **this**.points = points;
63. **this**.matchesPlayed = matchesPlayed;
64. }
66. /\*\*
67. \* Constructor of the Football class
68. \*
69. \* @param name           name of the club
70. \* @param location       location of the club
71. \* @param wins           number of win achieved by the club
72. \* @param defeats        number of defeats accounted by the club
73. \* @param draws          number of draws achieved by the club
74. \* @param goalsScored    number of goals scored by the club
75. \* @param goalsReceived  number of goals received by the club
76. \* @param goalDifference number of goals difference
77. \* @param points         number of points club currently has
78. \* @param matchesPlayed  number of matches played by the club
79. \*/
80. **public** FootballClub(String name, String location, **int** wins, **int** defeats, **int** draws, **int** goalsScored, **int** goalsReceived, **int** goalDifference, **int** points, **int** matchesPlayed) {
81. **super**(name, location);
82. **this**.wins = wins;
83. **this**.defeats = defeats;
84. **this**.draws = draws;
85. **this**.goalsScored = goalsScored;
86. **this**.goalsReceived = goalsReceived;
87. **this**.goalDifference = goalDifference;
88. **this**.points = points;
89. **this**.matchesPlayed = matchesPlayed;
90. }
92. /\*\*
93. \* Getter/Accessor of the number of goals received instance variable of the FootballClub
94. \*
95. \* @return int the number of goals received by the FootballClub
96. \*/
97. **public** **int** getGoalsReceived() {
98. **return** goalsReceived;
99. }
101. /\*\*
102. \* Setter/Mutator of the number of goals received instance variable of the FootballClub
103. \*
104. \* @param goalsReceived the number of goals received by the FootballClub
105. \*/
106. **public** **void** setGoalsReceived(**int** goalsReceived) {
107. **this**.goalsReceived = goalsReceived;
108. }
110. /\*\*
111. \* Getter/Accessor of the number of goals scored instance variable of the FootballClub
112. \*
113. \* @return int the number of goals scored by the FootballClub
114. \*/
115. **public** **int** getGoalsScored() {
116. **return** goalsScored;
117. }
119. /\*\*
120. \* Setter/Mutator of the number of goals scored instance variable of the FootballClub
121. \*
122. \* @param goalsScored the number of goals scored by the FootballClub
123. \*/
124. **public** **void** setGoalsScored(**int** goalsScored) {
125. **this**.goalsScored = goalsScored;
126. }
128. /\*\*
129. \* Getter/Accessor of the number of current points instance variable of the FootballClub
130. \*
131. \* @return int the number of current point the FootballClub has
132. \*/
133. **public** **int** getPoints() {
134. **return** points;
135. }
137. /\*\*
138. \* Setter/Mutator of the number of current points instance variable of the FootballClub
139. \*
140. \* @param points the number of current point the FootballClub has
141. \*/
142. **public** **void** setPoints(**int** points) {
143. **this**.points = points;
144. }
146. /\*\*
147. \* Getter/Accessor of the number of matches played by the FootballClub
148. \*
149. \* @return the number of matches played by the SportsClub
150. \*/
151. **public** **int** getMatchesPlayed() {
152. **return** matchesPlayed;
153. }
155. /\*\*
156. \* Setter/Mutator of the number of matches played by the FootballClub
157. \*
158. \* @param matchesPlayed the number of matches played by the SportsClub
159. \*/
160. **public** **void** setMatchesPlayed(**int** matchesPlayed) {
161. **this**.matchesPlayed = matchesPlayed;
162. }
164. /\*\*
165. \* Getter/Accessor of the number of wins of the SportClub
166. \*
167. \* @return the number of wins the SportsClub has achieved
168. \*/
169. **public** **int** getWins() {
170. **return** wins;
171. }
173. /\*\*
174. \* Setter/Mutator of the number of wins of the SportsClub
175. \*
176. \* @param wins the number of wins the SportsClub has achieved
177. \*/
178. **public** **void** setWins(**int** wins) {
179. **this**.wins = wins;
180. }
182. /\*\*
183. \* Getter/Accessor of the number of defeats of the SportClub
184. \*
185. \* @return the number of defeats the SportsClub has accounted
186. \*/
187. **public** **int** getDefeats() {
188. **return** defeats;
189. }
191. /\*\*
192. \* Setter/Mutator of the number of defeats of the SportsClub
193. \*
194. \* @param defeats the number of defeats the SportsClub has accounted
195. \*/
196. **public** **void** setDefeats(**int** defeats) {
197. **this**.defeats = defeats;
198. }
200. /\*\*
201. \* Getter/Accessor of the number of draws of the SportClub
202. \*
203. \* @return the number of draws the SportsClub has achieved
204. \*/
205. **public** **int** getDraws() {
206. **return** draws;
207. }
209. /\*\*
210. \* Setter/Mutator of the number of draws of the SportsClub
211. \*
212. \* @param draws the number of draws the SportsClub has achieved
213. \*/
214. **public** **void** setDraws(**int** draws) {
215. **this**.draws = draws;
216. }
218. /\*\*
219. \* toString method of the class concrete class FootballClub
220. \*
221. \* @return a string containing all the instance variable with the respective instantiation
222. \*/
223. **public** **int** getGoalDifference() {
224. **return** goalsScored - goalsReceived;
225. }
227. **public** **void** setGoalDifference(**int** goalDifference) {
228. **this**.goalDifference = goalDifference;
229. }
231. /\*\*
232. \* toString method of the class Football CLub
233. \*
234. \* @return a string containing all the instance variables with the respective instantiation
235. \*/
236. @Override
237. **public** String toString() {
238. **return** "FootballClub[" +
239. "name=" + **super**.getName() +
240. ", location=" + **super**.getLocation() +
241. ", wins=" + wins +
242. ", defeats=" + defeats +
243. ", draws=" + draws +
244. ", matchesPlayed=" + matchesPlayed +
245. ", goalsReceived=" + goalsReceived +
246. ", goalsScored=" + goalsScored +
247. ", goalDifference=" + goalDifference +
248. ", points=" + points +
249. ']';
250. }
252. /\*\*
253. \* Equals method of the FootballClub
254. \*
255. \* @param o object containing any type
256. \* @return a boolean if the object is an instance of FootballClub
257. \*/
258. @Override
259. **public** **boolean** equals(Object o) {
260. **if** (**this** == o) **return** **true**;
261. **if** (!(o **instanceof** FootballClub)) **return** **false**;
262. **if** (!**super**.equals(o)) **return** **false**;
263. FootballClub that = (FootballClub) o;
264. **return** getWins() == that.getWins() && getDefeats() == that.getDefeats() && getDraws() == that.getDraws() && getMatchesPlayed() == that.getMatchesPlayed() && getGoalsReceived() == that.getGoalsReceived() && getGoalsScored() == that.getGoalsScored() && getGoalDifference() == that.getGoalDifference() && getPoints() == that.getPoints();
265. }
267. /\*\*
268. \* Hashcode method of the class FootballClub
269. \*
270. \* @return int containing the hashcode
271. \*/
272. @Override
273. **public** **int** hashCode() {
274. **return** Objects.hash(**super**.hashCode(), getWins(), getDefeats(), getDraws(), getMatchesPlayed(), getGoalsReceived(), getGoalsScored(), getGoalDifference(), getPoints());
275. }
277. /\*\*
278. \* compareTo method of the class FootballClub
279. \*
280. \* @return int containing the integer comparison of points of the instances of Football class
281. \*/
282. @Override
283. **public** **int** compareTo(SportsClub otherFootballClub) {
284. **if** (**this**.points > ((FootballClub) otherFootballClub).getPoints()) {
285. **return** 1;
286. } **else** **if** (**this**.points < ((FootballClub) otherFootballClub).getPoints()) {
287. **return** -1;
288. } **else** {
289. **return** Integer.compare(**this**.getGoalDifference(), ((FootballClub) otherFootballClub).getGoalDifference());
290. }
291. }
293. }

#### UniversityFootballClub.java

App/entities/UniversityFootballClub.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. /\*\*
11. \* This is the concrete class UniversityFootballClub which extends FootballClub class and inherits those
12. \* instance variables and methods.
13. \* Also it includes Accessors and Mutators of the instance variables and
14. \* the toString method of the class.
15. \*
16. \* @author Radhika Ranasinghe
17. \* @version 1.0
18. \* @since 2020-11-15
19. \*/
20. **public** **class** UniversityFootballClub **extends** FootballClub {
22. **private** String universityName;
23. **private** **int** underAge;
25. /\*\*
26. \* Default constructor of the concrete class FootballClub
27. \*/
28. **public** UniversityFootballClub() {
29. }
31. /\*\*
32. \* Constructor of the concrete class UniversityFootballClub.
33. \*
34. \* @param name           name of the club
35. \* @param location       location of the club
36. \* @param wins           number of win achieved by the club
37. \* @param defeats        number of defeats accounted by the club
38. \* @param draws          number of draws achieved by the club
39. \* @param goalsScored    number of goals scored by the club
40. \* @param goalsReceived  number of goals received by the club
41. \* @param goalDifference number of goals difference
42. \* @param points         number of points club currently has
43. \* @param matchesPlayed  number of matches played by the club
44. \* @param universityName name of the university which owns the football club
45. \* @param underAge       under which age category the football club exists
46. \*/
47. **public** UniversityFootballClub(String name, String location, **int** wins, **int** defeats, **int** draws, **int** goalsScored, **int** goalsReceived, **int** goalDifference, **int** points, **int** matchesPlayed, String universityName, **int** underAge) {
48. **super**(name, location, wins, defeats, draws, goalsScored, goalsReceived, goalDifference, points, matchesPlayed);
49. **this**.universityName = universityName;
50. **this**.underAge = underAge;
51. }
53. /\*\*
54. \* Constructor of the concrete class UniversityFootballClub.
55. \*
56. \* @param universityName name of the university which owns the football club
57. \* @param underAge       under which age category the football club exists
58. \*/
59. **public** UniversityFootballClub(String universityName, **int** underAge) {
60. **this**.universityName = universityName;
61. **this**.underAge = underAge;
62. }
64. /\*\*
65. \* Getter/Accessor of the name of the university instance variable of the UniversityFootballClub
66. \*
67. \* @return String a String containing the name of the University of the UniversityFootballClub
68. \*/
69. **public** String getUniversityName() {
70. **return** universityName;
71. }
73. /\*\*
74. \* Setter/Mutator of the name of the university instance variable of the UniversityFootballClub
75. \*
76. \* @param universityName the name of the University of the UniversityFootballClub
77. \*/
78. **public** **void** setUniversityName(String universityName) {
79. **this**.universityName = universityName;
80. }
82. /\*\*
83. \* Getter/Accessor of the under age category the football club exists instance variable of the UniversityFootballClub
84. \*
85. \* @return int age of the under age category of the UniversityFootballClub
86. \*/
87. **public** **int** getUnderAge() {
88. **return** underAge;
89. }
91. /\*\*
92. \* Setter/Mutator of the under age category the football club exists instance variable of the UniversityFootballClub
93. \*
94. \* @param underAge age of the under age category of the UniversityFootballClub
95. \*/
96. **public** **void** setUnderAge(**int** underAge) {
97. **this**.underAge = underAge;
98. }
100. /\*\*
101. \* toString method of the class concrete class UniversityFootballClub
102. \*
103. \* @return a string containing all the instance variable with the respective instantiation
104. \*/
105. **public** String toString() {
106. **return** "UniversityFootballClub[" +
107. "clubName= '" + **super**.getName() + "'" +
108. ", clubLocation= '" + **super**.getLocation() + "'" +
109. ", numOfWins= " + **super**.getWins() +
110. ", numOfDefeats= " + **super**.getDefeats() +
111. ", numOfDraws= " + **super**.getDraws() +
112. ", numOfGoalsReceived= " + **super**.getGoalsReceived() +
113. ", numOfGoalsScored= " + **super**.getGoalsScored() +
114. ", currentNumOfPoints= " + **super**.getPoints() +
115. ", numOfMatchesPlayed= " + **super**.getMatchesPlayed() +
116. ", universityName= '" + universityName + "'" +
117. ", underAge= " + underAge +
118. "]";
119. }
121. /\*\*
122. \* Equals method of the UniversityFootballClub class
123. \*
124. \* @param o object containing any type
125. \* @return a boolean if the object is an instance of UniversityFootballClub
126. \*/
127. **public** **boolean** equals(Object o) {
128. **if** (**this** == o) {
129. **return** **true**;
130. }
131. **if** (!(o **instanceof** UniversityFootballClub)) {
132. **return** **false**;
133. }
134. **if** (!**super**.equals(o)) {
135. **return** **false**;
136. }
137. UniversityFootballClub universityFootballClub = (UniversityFootballClub) o;
138. **return** getUnderAge() == universityFootballClub.getUnderAge() &&
139. getUniversityName().equals(universityFootballClub.getUniversityName());
140. }
142. /\*\*
143. \* Hashcode method of the class UniversityFootballClub
144. \*
145. \* @return int containing the hashcode
146. \*/
147. **public** **int** hashCode() {
148. **final** **int** prime = 31;
149. **int** result = **super**.hashCode();
150. result = prime \* result + **this**.underAge;
151. result = prime \* result + ((**this**.universityName == **null**) ? 0 : **this**.universityName.hashCode());
152. **return** result;
153. }
155. /\*\*
156. \* compareTo method of the class UniversityFootballClub
157. \*
158. \* @return int containing the integer comparison of the UniversityFootballClub class
159. \*/
160. **public** **int** compareTo(UniversityFootballClub otherFootballClub) {
161. **return** Integer.compare(**super**.getPoints(), otherFootballClub.getPoints());
162. }
163. }

#### SchoolFootballClub.java

app/entities/SchoolFootballClub.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. **import** java.util.Objects;
12. /\*\*
13. \* This is the concrete class SchoolFootballClub which extends FootballClub class and inherits those
14. \* instance variables and methods.
15. \* Also it includes Accessors and Mutators of the instance variables and
16. \* the toString method of the class.
17. \*
18. \* @author Radhika Ranasinghe
19. \* @version 1.0
20. \* @since 2020-11-15
21. \*/
22. **public** **class** SchoolFootballClub **extends** FootballClub {
24. **private** String schoolName;
25. **private** **int** underAge;
27. /\*\*
28. \* Default constructor of the SchoolFootballClub concrete class
29. \*/
30. **public** SchoolFootballClub() {
31. }
33. /\*\*
34. \* Constructor of the SchoolFootballClub concrete class
35. \*
36. \* @param name           name of the club
37. \* @param location       location of the club
38. \* @param wins           number of win achieved by the club
39. \* @param defeats        number of defeats accounted by the club
40. \* @param draws          number of draws achieved by the club
41. \* @param goalsScored    number of goals scored by the club
42. \* @param goalsReceived  number of goals received by the club
43. \* @param goalDifference number of goals difference
44. \* @param points         number of points club currently has
45. \* @param matchesPlayed  number of matches played by the club
46. \* @param schoolName     name of the school which owns the football club
47. \* @param underAge       under which age category the football club exists
48. \*/
49. **public** SchoolFootballClub(String name, String location, **int** wins, **int** defeats, **int** draws, **int** goalsScored, **int** goalsReceived, **int** goalDifference, **int** points, **int** matchesPlayed, String schoolName, **int** underAge) {
50. **super**(name, location, wins, defeats, draws, goalsScored, goalsReceived, goalDifference, points, matchesPlayed);
51. **this**.schoolName = schoolName;
52. **this**.underAge = underAge;
53. }
55. /\*\*
56. \* Constructor of the SchoolFootballClub concrete class
57. \*
58. \* @param schoolName name of the school which owns the football club
59. \* @param underAge   under which age category the football club exists
60. \*/
61. **public** SchoolFootballClub(String schoolName, **int** underAge) {
62. **this**.schoolName = schoolName;
63. **this**.underAge = underAge;
64. }
66. /\*\*
67. \* Getter/Accessor of the name of the school instance variable of the SchoolFootballClub
68. \*
69. \* @return String a String containing the name of the School of the SchoolFootballClub
70. \*/
71. **public** String getSchoolName() {
72. **return** schoolName;
73. }
75. /\*\*
76. \* Setter/Mutator of the name of the school instance variable of the SchoolFootballClub
77. \*
78. \* @param schoolName the name of the school of the SchoolFootballClub
79. \*/
80. **public** **void** setSchoolName(String schoolName) {
81. **this**.schoolName = schoolName;
82. }
84. /\*\*
85. \* Getter/Accessor of the under age category the football club exists instance variable of the SchoolFootballClub
86. \*
87. \* @return int age of the under age category of the SchoolFootballClub
88. \*/
89. **public** **int** getUnderAge() {
90. **return** underAge;
91. }
93. /\*\*
94. \* Setter/Mutator of the under age category the football club exists instance variable of the SchoolFootballClub
95. \*
96. \* @param underAge age of the under age category of the SchoolFootballClub
97. \*/
98. **public** **void** setUnderAge(**int** underAge) {
99. **this**.underAge = underAge;
100. }
102. /\*\*
103. \* toString method of the class concrete class SchoolFootballClub
104. \*
105. \* @return a string containing all the instance variable with the respective instantiation
106. \*/
107. **public** String toString() {
108. **return** "SchoolFootballClub[ " +
109. "clubName= '" + **super**.getName() + "'" +
110. ", clubLocation= '" + **super**.getLocation() + "'" +
111. ", numOfWins= " + **super**.getWins() +
112. ", numOfDefeats= " + **super**.getDefeats() +
113. ", numOfDraws= " + **super**.getDraws() +
114. ", numOfGoalsReceived= " + **super**.getGoalsReceived() +
115. ", numOfGoalsScored= " + **super**.getGoalsScored() +
116. ", currentNumOfPoints= " + **super**.getPoints() +
117. ", numOfMatchesPlayed= " + **super**.getMatchesPlayed() +
118. ", schoolName= '" + schoolName + "'" +
119. ", underAge= " + underAge +
120. "]";
121. }
123. /\*\*
124. \* Equals method of the SchoolFootballClub
125. \*
126. \* @param o object containing any type
127. \* @return a boolean if the object is an instance of SportsClub
128. \*/
129. @Override
130. **public** **boolean** equals(Object o) {
131. **if** (**this** == o) **return** **true**;
132. **if** (!(o **instanceof** SchoolFootballClub)) **return** **false**;
133. **if** (!**super**.equals(o)) **return** **false**;
134. SchoolFootballClub that = (SchoolFootballClub) o;
135. **return** getUnderAge() == that.getUnderAge() && getSchoolName().equals(that.getSchoolName());
136. }
138. /\*\*
139. \* Hashcode method of the class SchoolFootballClub
140. \*
141. \* @return int containing the hashcode
142. \*/
143. @Override
144. **public** **int** hashCode() {
145. **return** Objects.hash(**super**.hashCode(), getSchoolName(), getUnderAge());
146. }
148. /\*\*
149. \* compareTo method of the class SchoolFootballClub
150. \*
151. \* @return int containing the integer comparison of the SchoolFootballClub class
152. \*/
153. **public** **int** compareTo(SchoolFootballClub otherFootballClub) {
154. **return** Integer.compare(**super**.getPoints(), otherFootballClub.getPoints());
155. }
156. }

#### Date.java

App/entities/Date.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. **import** java.io.Serializable;
11. **import** java.util.Objects;
13. /\*\*
14. \* This is the concrete class Date which includes Accessors and Mutators of the instance variables and
15. \* the toString method of the class.
16. \*
17. \* @author Radhika Ranasinghe
18. \* @version 1.0
19. \* @since 2020-11-15
20. \*/
21. **public** **class** Date **implements** Serializable {
22. **private** **int** day;
23. **private** **int** month;
24. **private** **int** year;
26. /\*\*
27. \* Default constructor of the Date concrete class
28. \*/
29. **public** Date() {
30. }
32. /\*\*
33. \* Constructor of the Date concrete class
34. \*
35. \* @param day   the given day
36. \* @param month the given month
37. \* @param year  the given year
38. \*/
39. **public** Date(**int** day, **int** month, **int** year) {
40. **this**.day = day;
41. **this**.month = month;
42. **this**.year = year;
43. }
45. /\*\*
46. \* Getter/Accessor of the day instance variable
47. \*
48. \* @return int the day of the instance variable
49. \*/
50. **public** **int** getDay() {
51. **return** day;
52. }
54. /\*\*
55. \* Setter/Mutator of the day instance variable
56. \*
57. \* @param day the day to be set as the instance variable
58. \*/
59. **public** **void** setDay(**int** day) {
60. **this**.day = day;
61. }
63. /\*\*
64. \* Getter/Accessor of the month instance variable
65. \*
66. \* @return int the month of the instance variable
67. \*/
68. **public** **int** getMonth() {
69. **return** month;
70. }
72. /\*\*
73. \* Setter/Mutator of the month instance variable
74. \*
75. \* @param month the month to be set as the instance variable
76. \*/
77. **public** **void** setMonth(**int** month) {
78. **this**.month = month;
79. }
81. /\*\*
82. \* Getter/Accessor of the year instance variable
83. \*
84. \* @return int the year of the instance variable
85. \*/
86. **public** **int** getYear() {
87. **return** year;
88. }
90. /\*\*
91. \* Setter/Mutator of the year instance variable
92. \*
93. \* @param year the year to be set as the instance variable
94. \*/
95. **public** **void** setYear(**int** year) {
96. **this**.year = year;
97. }
99. /\*\*
100. \* toString method of the class concrete class Date
101. \*
102. \* @return a string containing all the instance variable with the respective instantiation
103. \*/
104. @Override
105. **public** String toString() {
106. **return** "Date{" +
107. "day=" + day +
108. ", month=" + month +
109. ", year=" + year +
110. '}';
111. }
113. /\*\*
114. \* Equals method of the Date class
115. \*
116. \* @param o object containing any type
117. \* @return a boolean if the object is an instance of Date
118. \*/
119. @Override
120. **public** **boolean** equals(Object o) {
121. **if** (**this** == o) **return** **true**;
122. **if** (!(o **instanceof** Date)) **return** **false**;
123. Date date = (Date) o;
124. **return** getDay() == date.getDay() && getMonth() == date.getMonth() && getYear() == date.getYear();
125. }
127. /\*\*
128. \* Hashcode method of the Date class
129. \*
130. \* @return int containing the hashcode
131. \*/
132. @Override
133. **public** **int** hashCode() {
134. **return** Objects.hash(getDay(), getMonth(), getYear());
135. }
137. /\*\*
138. \* compareTo method of the class Date
139. \*
140. \* @return int containing the integer comparison of the Date class
141. \*/
142. **public** **int** compareTo(Date date) {
143. **if** (**this**.year > date.getYear()) {
144. **return** 1;
145. } **else** **if** (**this**.year < date.getYear()) {
146. **return** -1;
147. } **else** {
148. **if** (**this**.month > date.getMonth()) {
149. **return** 1;
150. } **else** **if** (**this**.month < date.getMonth()) {
151. **return** -1;
152. } **else** {
153. **return** Integer.compare(**this**.day, date.getDay());
154. }
155. }
156. }
157. }

#### Match.java

App/entities/Match.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. **import** java.io.Serializable;
11. **import** java.util.Objects;
13. /\*\*
14. \* This is the concrete class Match which includes Accessors and Mutators of the instance variables and
15. \* the toString method of the class.
16. \*
17. \* @author Radhika Ranasinghe
18. \* @version 1.0
19. \* @since 2020-11-15
20. \*/
21. **public** **class** Match<T **extends** SportsClub> **implements** Serializable, Comparable<Match<FootballClub>> {
22. **private** T homeClub;
23. **private** T awayClub;
24. **private** **int** goalsHomeScored;
25. **private** **int** goalsAwayScored;
26. **private** Date matchDate;
28. /\*\*
29. \* Default constructor of the Match concrete class
30. \*/
31. **public** Match() {
32. }
34. /\*\*
35. \* Constructor of the concrete class Match
36. \*
37. \* @param homeClub        the club that played as the home team
38. \* @param awayClub        the club that played as the away team
39. \* @param goalsHomeScored number of goals scored by the home club
40. \* @param goalsAwayScored number of goals scored by the away club
41. \* @param matchDate       the date the match was played on
42. \*/
43. **public** Match(T homeClub, T awayClub, **int** goalsHomeScored, **int** goalsAwayScored, Date matchDate) {
44. **this**.homeClub = homeClub;
45. **this**.awayClub = awayClub;
46. **this**.goalsHomeScored = goalsHomeScored;
47. **this**.goalsAwayScored = goalsAwayScored;
48. **this**.matchDate = matchDate;
49. }
51. /\*\*
52. \* Getter/Accessor of the home club from the played match
53. \*
54. \* @return SportsClub the club that played as the home team
55. \*/
56. **public** T getHomeClub() {
57. **return** homeClub;
58. }
60. /\*\*
61. \* Setter/Mutator of the home club from the played match
62. \*
63. \* @param homeClub the club that played as the home team
64. \*/
65. **public** **void** setHomeClub(T homeClub) {
66. **this**.homeClub = homeClub;
67. }
69. /\*\*
70. \* Getter/Accessor of the away club from the played match
71. \*
72. \* @return SportsClub the club that played as the away team
73. \*/
74. **public** T getAwayClub() {
75. **return** awayClub;
76. }
78. /\*\*
79. \* Setter/Mutator of the away club from the played match
80. \*
81. \* @param awayClub the club that played as the away team
82. \*/
83. **public** **void** setAwayClub(T awayClub) {
84. **this**.awayClub = awayClub;
85. }
87. /\*\*
88. \* Getter/Accessor of the goals scored by the home club from the played match
89. \*
90. \* @return int the number of goals scored by the club that played as the home team
91. \*/
92. **public** **int** getGoalsHomeScored() {
93. **return** goalsHomeScored;
94. }
96. /\*\*
97. \* Setter/Mutator of the goals scored by the home club from the played match
98. \*
99. \* @param goalsHomeScored the number of goals scored by the club that played as the home team
100. \*/
101. **public** **void** setGoalsHomeScored(**int** goalsHomeScored) {
102. **this**.goalsHomeScored = goalsHomeScored;
103. }
105. /\*\*
106. \* Getter/Accessor of the goals scored by the away club from the played match
107. \*
108. \* @return int the number of goals scored by the club that played as the away team
109. \*/
110. **public** **int** getGoalsAwayScored() {
111. **return** goalsAwayScored;
112. }
114. /\*\*
115. \* Setter/Mutator of the goals scored by the away club from the played match
116. \*
117. \* @param goalsAwayScored the number of goals scored by the club that played as the away team
118. \*/
119. **public** **void** setGoalsAwayScored(**int** goalsAwayScored) {
120. **this**.goalsAwayScored = goalsAwayScored;
121. }
123. /\*\*
124. \* Getter/Accessor of the date of the played match
125. \*
126. \* @return Date the date which the match was played on
127. \*/
128. **public** Date getMatchDate() {
129. **return** matchDate;
130. }
132. /\*\*
133. \* Setter/Mutator of the date of the played match
134. \*
135. \* @param matchDate the date which the match was played on
136. \*/
137. **public** **void** setMatchDate(Date matchDate) {
138. **this**.matchDate = matchDate;
139. }
141. /\*\*
142. \* toString method of the class concrete class Match
143. \*
144. \* @return a string containing all the instance variable with the respective instantiation
145. \*/
146. @Override
147. **public** String toString() {
148. **return** "Match[" +
149. "homeClub= " + homeClub +
150. ", awayClub= " + awayClub +
151. ", goalsHomeScored= " + goalsHomeScored +
152. ", goalsAwayScored= " + goalsAwayScored +
153. ", matchDate= " + matchDate +
154. "]";
155. }
157. /\*\*
158. \* Equals method of the Match class
159. \*
160. \* @param o object containing any type
161. \* @return a boolean if the object is an instance of SportsClub
162. \*/
163. @Override
164. **public** **boolean** equals(Object o) {
165. **if** (**this** == o) **return** **true**;
166. **if** (!(o **instanceof** Match)) **return** **false**;
167. Match<?> match = (Match<?>) o;
168. **return** getGoalsHomeScored() == match.getGoalsHomeScored() && getGoalsAwayScored() == match.getGoalsAwayScored() && Objects.equals(getHomeClub(), match.getHomeClub()) && Objects.equals(getAwayClub(), match.getAwayClub()) && Objects.equals(getMatchDate(), match.getMatchDate());
169. }
171. /\*\*
172. \* Hashcode method of the class Match
173. \*
174. \* @return int containing the hashcode
175. \*/
176. @Override
177. **public** **int** hashCode() {
178. **return** Objects.hash(getHomeClub(), getAwayClub(), getGoalsHomeScored(), getGoalsAwayScored(), getMatchDate());
179. }
181. /\*\*
182. \* compareTo method of the class Match
183. \*
184. \* @return int containing the integer comparison of the match class
185. \*/
186. @Override
187. **public** **int** compareTo(Match<FootballClub> match) {
188. **if** (**this**.matchDate.getYear() > match.getMatchDate().getYear()) {
189. **return** 1;
190. } **else** **if** (**this**.matchDate.getYear() < match.matchDate.getYear()) {
191. **return** -1;
192. } **else** {
193. **if** (**this**.matchDate.getMonth() > match.matchDate.getMonth()) {
194. **return** 1;
195. } **else** **if** (**this**.matchDate.getMonth() < match.matchDate.getMonth()) {
196. **return** -1;
197. } **else** {
198. **return** Integer.compare(**this**.matchDate.getDay(), match.matchDate.getDay());
199. }
200. }
201. }
202. }

#### PremierLeagueManager.java

App/entities/PremierLeagueManager.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. **import** java.io.\*;
11. **import** java.util.\*;
13. /\*\*
14. \* This is the concrete class PremierLeagueManager which implements the LeagueManager Interface and
15. \* inherits all the abstract methods from it.
16. \* Also it includes Accessors and Mutators of the instance variables and the toString method of the class.
17. \*
18. \* @author Radhika Ranasinghe
19. \* @version 1.0
20. \* @since 2020-11-15
21. \*/
22. **public** **class** PremierLeagueManager **implements** LeagueManager<FootballClub> {
23. // Declaring a constant called 'MAX\_COUNT' and assigning to 20
24. **public** **static** **final** **int** MAX\_COUNT = 20;
25. // Declaring a PremierLeagueManager called 'manager' and assigning to null
26. **private** **static** PremierLeagueManager manager = **null**;
27. // Declaring a List of type SpotsClub called 'clubsInLeague' and assigning to ArrayList
28. **private** List<SportsClub> clubsInLeague = **new** ArrayList<>();
29. // Declaring a List of type Match called 'matchesInLeague' and assigning to ArrayList
30. **private** List<Match<FootballClub>> matchesInLeague = **new** ArrayList<>();
32. /\*\*
33. \* Default constructor of the concrete class PremierLeagueManager
34. \*/
35. **private** PremierLeagueManager() {
36. }
38. /\*\*
39. \* Constructor of the concrete class PremierLeagueManager
40. \*
41. \* @param clubsInLeague   an ArrayList of all the clubs that are registered for the Premier League
42. \* @param matchesInLeague an Arraylist of all the matches played in the Premier League
43. \*/
44. **public** PremierLeagueManager(ArrayList<SportsClub> clubsInLeague, ArrayList<Match<FootballClub>> matchesInLeague) {
45. **this**.clubsInLeague = clubsInLeague;
46. **this**.matchesInLeague = matchesInLeague;
47. }
49. /\*\*
50. \* Initializes a newly created object if the object is not created
51. \*
52. \* @return PremierLeagueManager object
53. \*/
54. **public** **static** PremierLeagueManager getInstance() {
55. //Lock is only needed if the object is null
56. **if** (manager == **null**) {
57. //To ensure no two threads entered at the same time
58. **synchronized** (PremierLeagueManager.**class**) {
59. **if** (manager == **null**) {
60. manager = **new** PremierLeagueManager();
61. }
62. }
63. }
64. **return** manager;
65. }
67. /\*\*
68. \* Method that adds a sports club to the league
69. \*
70. \* @param club the club that is to be added to the Premier League
71. \* @return boolean value which states if the club is added or not
72. \*/
73. @Override
74. **public** **boolean** addClub(FootballClub club) {
75. // Declaring a boolean and assigning true
76. **boolean** isAdded = **true**;
77. // Ensuring if the capacity if there to add another club
78. **if** (clubsInLeague.size() < MAX\_COUNT) {
79. // Adding the football club to the premier league
80. clubsInLeague.add(club);
81. } **else** {
82. // Assigning the boolean value to false
83. isAdded = **false**;
84. }
85. **return** isAdded;
87. }
89. /\*\*
90. \* Method that deletes a sports club from the premier league
91. \*
92. \* @param clubName the name of the club that is to be deleted
93. \* @return boolean value which states if the club is deleted or not
94. \*/
95. @Override
96. **public** **boolean** deleteClub(String clubName) {
97. // Declaring a boolean and assigning false
98. **boolean** hasFound = **false**;
99. // Looping within the clubs in the league
100. **for** (SportsClub club : clubsInLeague) {
101. // Checks if the the user input matches with iteration name
102. **if** (club.getName().equalsIgnoreCase(clubName)) {
103. // Removes the club from the premier league
104. hasFound = clubsInLeague.remove(club);
105. // Break the for loop
106. **break**;
107. }
108. }
109. **return** hasFound;
110. }
112. /\*\*
113. \* Method that returns the stats of the given saved club.
114. \*
115. \* @param club the club which the stats belong to
116. \* @return String value having statistics of the given club
117. \*/
118. @Override
119. **public** String getStatsOfAClub(FootballClub club) {
120. **return** "\nStatistics of the club " + club.getName() + " (" + club.getLocation() + ")" +
121. "\n\t> Number of Wins\t: " + club.getWins() + "\n\t> Number of Draws\t: " + club.getDraws() +
122. "\n\t> Number of Defeats\t: " + club.getDefeats() + "\n\t> Goals Scored\t\t: " + club.getGoalsScored() +
123. "\n\t> Goals Against\t\t: " + club.getGoalsReceived() + "\n\t> Goal Difference\t: " + club.getGoalDifference() +
124. "\n\t> Matched Played\t: " + club.getMatchesPlayed() + "\n\t> Current Points\t: " + club.getPoints();
125. }
127. /\*\*
128. \* Method that adds a played match to the premier league
129. \*
130. \* @param match containing the user input data about the match
131. \* @return Match object containing the match added by the user
132. \*/
133. **public** Match<FootballClub> addMatch(Match<FootballClub> match) {
134. // Declaring a FootballClub called homeClub and assigning to a Football
135. FootballClub homeClub = **new** FootballClub();
136. // Declaring a FootballClub called awayClub and assigning to a Football
137. FootballClub awayClub = **new** FootballClub();
139. // For each loop to iterate through clubs in the premier league
140. **for** (SportsClub club : clubsInLeague) {
141. // To check if the name and location of the iteration matches the name and the location of the home club of the match
142. **if** (club.getName().equals(match.getHomeClub().getName()) &&
143. club.getLocation().equals(match.getHomeClub().getLocation())) {
144. homeClub = (FootballClub) club; //Assigning homeClub variable to the correspondent football club
145. }
146. // To check if the name and location of the iteration matches the name and the location of the away club of the match
147. **if** (club.getName().equals(match.getAwayClub().getName()) &&
148. club.getLocation().equals(match.getAwayClub().getLocation())) {
149. awayClub = (FootballClub) club; //Assigning awayClub variable to the correspondent football club
150. }
151. }
152. //Ensuring if the match if already exists in the premier league
153. **if** (!matchesInLeague.contains(match)) {
154. //if homeClub has scored more than the awayClub
155. **if** (match.getGoalsHomeScored() > match.getGoalsAwayScored()) {
156. homeClub.setWins(homeClub.getWins() + 1); // homeClub's wins increases by 1
157. awayClub.setDefeats(awayClub.getDefeats() + 1); // awayClub defeats increases by 1
158. homeClub.setPoints(homeClub.getPoints() + 3); // homeClub's points increases by 3
160. //if awayClub has scored more than the homeClub
161. } **else** **if** (match.getGoalsHomeScored() < match.getGoalsAwayScored()) {
162. awayClub.setWins(match.getAwayClub().getWins() + 1); // awayClub's wins increases by 1
163. homeClub.setDefeats(match.getHomeClub().getDefeats() + 1); // homeClub defeats increases by 1
164. awayClub.setPoints(match.getAwayClub().getPoints() + 3); // awayClub's points increases by 3
166. } **else** {
167. //if awayClub and homeClub scored the same
168. homeClub.setDraws(match.getHomeClub().getDraws() + 1); // homeClub's draws increases by 1
169. awayClub.setDraws(match.getAwayClub().getDraws() + 1);  // awayClub's draws increases by 1
170. homeClub.setPoints(match.getHomeClub().getPoints() + 1); // homeClub's points increases by 1
171. awayClub.setPoints(match.getAwayClub().getPoints() + 1); // awayClub's points increases by 1
172. }
173. // Other user inputs will be increased respectively
174. homeClub.setMatchesPlayed(match.getHomeClub().getMatchesPlayed() + 1);
175. homeClub.setGoalsScored(match.getHomeClub().getGoalsScored() + match.getGoalsHomeScored());
176. homeClub.setGoalsReceived(match.getHomeClub().getGoalsReceived() + match.getGoalsAwayScored());
177. awayClub.setMatchesPlayed(match.getAwayClub().getMatchesPlayed() + 1);
178. awayClub.setGoalsScored(match.getAwayClub().getGoalsScored() + match.getGoalsAwayScored());
179. awayClub.setGoalsReceived(match.getAwayClub().getGoalsReceived() + match.getGoalsHomeScored());
180. matchesInLeague.add(match); // The match is added afterwards
181. **return** match;
182. }
183. **return** **null**;
184. }
186. /\*\*
187. \* Method that returns the stats of all the clubs saved
188. \*
189. \* @return ArrayList containing Strings of stats of all the clubs
190. \*/
191. **public** String getStatsOfAllClubs() {
192. // Declaring a List called tempList and assigning to ArrayList
193. List<FootballClub> tempList = **new** ArrayList<>();
194. //Adding the clubs of the premier league
195. **for** (SportsClub club : clubsInLeague) {
196. tempList.add((FootballClub) club);
197. }
198. //Sorting by points
199. tempList.sort(Collections.reverseOrder());
200. List<String[]> rows = **new** ArrayList<>();  // Declaring a List called rows and assigning to a ArrayList
201. StringBuilder sb = **new** StringBuilder();  // Declaring a StringBuilder called tempList and assigning to a StringBuilder
202. sb.append("\n\t\tT H E   P R E M I E R   L E A G U E   T A B L E\n");
203. sb.append("\n>\tWins are shown with '+' \n>\tLosses are shown with '-' \n>\tDraws are shown with '\*'\n>\tMatch data not found is shown with '/' \n\n");
204. String[] headers = {"Position", "Club Name",
205. "Club Location",
206. "Played Matches",
207. "Won",
208. "Loss",
209. "Drawn",
210. "GF",
211. "GA",
212. "GD",
213. "Points",
214. "Last 5 Matches"};
215. // Maximum width the columns will take up
216. **int**[] maxWidths = Arrays.stream(headers).mapToInt(String::length).toArray();
217. //Adding columns to the table
218. **for** (FootballClub club : tempList) {
219. rows.add(**new** String[]{
220. String.valueOf(tempList.indexOf(club) + 1),
221. club.getName(),
222. club.getLocation(),
223. String.valueOf(club.getMatchesPlayed()),
224. String.valueOf(club.getWins()),
225. String.valueOf(club.getDefeats()),
226. String.valueOf(club.getDraws()),
227. String.valueOf(club.getGoalsScored()),
228. String.valueOf(club.getGoalsReceived()),
229. String.valueOf(club.getGoalDifference()),
230. String.valueOf(club.getPoints()),
231. displayLastFiveMatches(club)});
232. }

235. **for** (String[] cells : rows) {
236. **for** (**int** i = 0; i < cells.length; i++) {
237. maxWidths[i] = Math.max(maxWidths[i], cells[i].length());
238. }
239. }
241. **for** (**int** i = 0; i < maxWidths.length; i++) {
242. String line = String.join("", Collections.nCopies(maxWidths[i] +
243. "|".length() + 1, "-"));
244. sb.append("+").append(line).append(i == maxWidths.length - 1 ? "+" : "");
245. }
246. sb.append("\n");
247. **for** (**int** i = 0; i < headers.length; i++) {
248. String s = headers[i];
249. String verStrTemp = i == headers.length - 1 ? "|" : "";
250. String headingLine = String.format("%s %-" + maxWidths[i] + "s %s", "|", s, verStrTemp);
251. sb.append(headingLine);
252. }
253. sb.append("\n");
254. **for** (**int** i = 0; i < maxWidths.length; i++) {
255. String line = String.join("", Collections.nCopies(maxWidths[i] +
256. "|".length() + 1, "-"));
257. sb.append("+").append(line).append(i == maxWidths.length - 1 ? "+" : "");
258. }
259. sb.append("\n");
260. **for** (String[] cells : rows) {
261. **for** (**int** i = 0; i < cells.length; i++) {
262. String s = cells[i];
263. String verStrTemp = i == cells.length - 1 ? "|" : "";
264. String rowLine = String.format("%s %-" + maxWidths[i] + "s %s", "|", s, verStrTemp);
265. sb.append(rowLine);
266. }
267. sb.append("\n");
268. }
269. **for** (**int** i = 0; i < maxWidths.length; i++) {
270. String line = String.join("", Collections.nCopies(maxWidths[i] +
271. "|".length() + 1, "-"));
272. sb.append("+").append(line).append(i == maxWidths.length - 1 ? "+" : "");
273. }
274. sb.append("\n");
275. **return** sb.toString();
276. }
278. /\*\*
279. \* Method to save the data of the Football clubs to file obtained by the user
280. \*
281. \* @param fileName the string taken as the file name to save data
282. \* @throws IOException handles failures related to reading, writing and searching for the called file
283. \*/
284. **public** **void** saveClubData(String fileName) **throws** IOException {
285. **try** {
286. // Opening the streams
287. FileOutputStream fileOutputStream = **new** FileOutputStream(fileName);
288. ObjectOutputStream objectOutputStream = **new** ObjectOutputStream(fileOutputStream);
290. // Iterating through the arrayList and writing Object by object to the file
291. **for** (SportsClub club : clubsInLeague) {
292. objectOutputStream.writeObject(club);
293. }
294. // Clears the stream
295. objectOutputStream.flush();
297. // Closing the streams
298. fileOutputStream.close();
299. objectOutputStream.close();
300. } **catch** (IOException e) {
301. e.printStackTrace();
302. }
304. }
306. /\*\*
307. \* Method to retrieve the data of the Football clubs from the file saved
308. \*
309. \* @param fileName the string taken as the file name of the data retrieval file
310. \* @throws IOException            handles failures related to reading, writing and searching for the called file
311. \* @throws ClassNotFoundException handles when a particular class tries to load and doesn't find the requested class in classpath
312. \*/
313. **public** **void** retrieveClubData(String fileName) **throws** IOException, ClassNotFoundException {
314. clubsInLeague.clear(); // Clearing the arrayList
315. **try** {
316. // Opening the streams
317. FileInputStream fileInputStream = **new** FileInputStream(fileName);
318. ObjectInputStream objectInputStream = **new** ObjectInputStream(fileInputStream);
320. // Iterating through the file object by object and adding to the arrayList called 'clubsInLeague'
321. **for** (; ; ) {
322. **try** {
323. clubsInLeague.add((FootballClub) objectInputStream.readObject());
324. } **catch** (EOFException e) {
325. //When it reaches the end of the file, the loop breaks
326. **break**;
327. }
328. }
329. // Closing the streams
330. fileInputStream.close();
331. objectInputStream.close();
332. } **catch** (FileNotFoundException ignored) {
334. }
335. }
337. /\*\*
338. \* Method to save the data of the Matches played by the football clubs to file obtained by the user
339. \*
340. \* @param fileName the string taken as the file name to save data
341. \* @throws IOException handles failures related to reading, writing and searching for the called file
342. \*/
343. **public** **void** saveMatchData(String fileName) **throws** IOException {
344. // Opening the streams
345. FileOutputStream fileOutputStream = **new** FileOutputStream(fileName);
346. ObjectOutputStream objectOutputStream = **new** ObjectOutputStream(fileOutputStream);
348. // Iterating through the arrayList and writing Object by object to the file
349. **for** (Match<FootballClub> match : matchesInLeague) {
350. objectOutputStream.writeObject(match);
351. }
352. //Clearing the stream
353. objectOutputStream.flush();
354. // Closing the streams
355. fileOutputStream.close();
356. objectOutputStream.close();
357. }
359. /\*\*
360. \* Method to retrieve the data of the Matches played by the football clubs from the file saved
361. \*
362. \* @param fileName the string taken as the file name to save data
363. \* @throws IOException            handles failures related to reading, writing and searching for the called file
364. \* @throws ClassNotFoundException handles when a particular class tries to load and doesn't find the requested class in classpath
365. \*/
366. **public** **void** retrieveMatchData(String fileName) **throws** IOException, ClassNotFoundException {
367. matchesInLeague.clear(); // Clearing the arrayList
368. **try** {
369. // Opening the streams
370. FileInputStream fileInputStream = **new** FileInputStream(fileName);
371. ObjectInputStream objectInputStream = **new** ObjectInputStream(fileInputStream);
373. // Iterating through the file object by object and adding to the arrayList called 'matchesInLeague'
374. **for** (; ; ) {
375. **try** {
376. Match<FootballClub> match = (Match<FootballClub>) objectInputStream.readObject();
377. matchesInLeague.add(match);
378. } **catch** (EOFException e) {
379. //When it reaches the end of the file, the loop breaks
380. **break**;
381. }
382. }
383. // Closing the streams
384. fileInputStream.close();
385. objectInputStream.close();
386. } **catch** (FileNotFoundException ignored) {
387. }
388. }
390. /\*\*
391. \* Getter/Accessor to get all the registered clubs in the premier league
392. \*
393. \* @return ArrayList containing all the clubs which are registered for the premier league
394. \*/
395. **public** List<SportsClub> getClubsInLeague() {
396. **return** clubsInLeague;
397. }
399. /\*\*
400. \* Setter/Mutator to set all the registered clubs in the premier league
401. \*
402. \* @param clubsInLeague containing all the clubs which are registered for the premier league
403. \*/
404. **public** **void** setClubsInLeague(ArrayList<SportsClub> clubsInLeague) {
405. **this**.clubsInLeague = clubsInLeague;
406. }
408. /\*\*
409. \* Getter/Accessor to get all the matches played in the premier league
410. \*
411. \* @return ArrayList containing all matches played in the premier league
412. \*/
413. **public** List<Match<FootballClub>> getMatchesInLeague() {
414. **return** matchesInLeague;
415. }
417. /\*\*
418. \* Setter/Mutator to set all the matches played in the premier league
419. \*
420. \* @param matchesInLeague containing all matches played in the premier league
421. \*/
422. **public** **void** setMatchesInLeague(ArrayList<Match<FootballClub>> matchesInLeague) {
423. **this**.matchesInLeague = matchesInLeague;
424. }
426. /\*\*
427. \* Method to display the last 5 matches of the selected club
428. \*
429. \* @param footballClub selected club to display
430. \* @return String containing an overview of the last 5 matches
431. \*/
432. **public** String displayLastFiveMatches(FootballClub footballClub) {
433. // Declaring a List called 'matchesPlayedByClub' and assigning to capacity to 5
434. List<Match<FootballClub>> matchesPlayedByClub = **new** ArrayList<>(5);
435. matchesInLeague.sort(Collections.reverseOrder()); // soring by compareTo
437. **for** (Match<FootballClub> match : matchesInLeague) {
438. // Checking if either the home club or the away club of the match is the selected club and the size of the arrayList is below 5
439. **if** ((match.getHomeClub().equals(footballClub) || match.getAwayClub().equals(footballClub)) && matchesPlayedByClub.size() <= 5) {
440. matchesPlayedByClub.add(match);
441. }
442. }
443. StringBuilder sb = **new** StringBuilder();
444. String won = " + ";
445. String lost = " - ";
446. String draw = " \* ";
447. String noMatch = " / ";
449. //Iterating through the selected 5 latest matches to be displayed
450. **for** (Match<FootballClub> match : matchesPlayedByClub) {
451. //If the football club taken in by the parameter is equal to the home club of the match of the iteration
452. **if** (footballClub.equals(match.getHomeClub())) {
453. // if the home club has scored more than the away club
454. **if** (match.getGoalsHomeScored() > match.getGoalsAwayScored()) {
455. // the home club has won
456. sb.append(won);
457. //If the away club has scored more than home club
458. } **else** **if** (match.getGoalsHomeScored() < match.getGoalsAwayScored()) {
459. // the away club has won
460. sb.append(lost);
461. } **else** {
462. // if both clubs has scored the same
463. sb.append(draw);
464. }
465. //If the football club taken in by the parameter is equal to the away club of the match of the iteration
466. } **else** {
467. // if the away club has scored more than the home club
468. **if** (match.getGoalsAwayScored() > match.getGoalsHomeScored()) {
469. // the away club has won
470. sb.append(won);
471. // if the home club has scored more than the away club
472. } **else** {
473. // the home club has won
474. sb.append(lost);
475. }
476. }
477. }
478. // Checking the arrayList has only has 5 or less than 5 items
479. **if** (matchesPlayedByClub.size() <= 5) {
480. **int** noMatchTimes = 5 - matchesPlayedByClub.size();
481. sb.append(noMatch.repeat(noMatchTimes));
482. }
483. **return** sb.toString();
484. }
486. /\*\*
487. \* Generates a random match containing a two random clubs from the premier league
488. \* and generates a random match date from given year
489. \*
490. \* @param year given season year
491. \* @return match random generated
492. \*/
493. **public** Match<FootballClub> generateRandomMatch(**int** year) {
494. // Declaring a 'Random' called rand and assigning to Random
495. Random rand = **new** Random();
496. // Declaring a List called 'randomTeams' and assigning to an ArrayList
497. List<SportsClub> randomTeams = **new** ArrayList<>();
498. // Declaring a List called 'scores' and assigning to an ArrayList
499. List<Integer> scores = **new** ArrayList<>();
501. **int** listOfElements = 2;
503. // Checking if the premier league has enough clubs to generate match out of
504. **if** (PremierLeagueManager.getInstance().getClubsInLeague().size() >= 2) {
505. //Iterating two times
506. **for** (**int** i = 0; i < listOfElements; i++) {
507. // Getting a random club index
508. **int** randClubIndex = rand.nextInt(PremierLeagueManager.getInstance().getClubsInLeague().size());
509. // Getting a random score below 11
510. **int** randScore = rand.nextInt(11);
512. // Selecting and adding them to the arrayList
513. randomTeams.add(PremierLeagueManager.getInstance().getClubsInLeague().get(randClubIndex));
514. scores.add(randScore);
515. **if** (randomTeams.size() > 1) {
516. // If the selected home club and away club is equal, different clubs are selected
517. **while** (randomTeams.get(0) == randomTeams.get(1)) {
518. //Removing the away club
519. randomTeams.remove(1);
520. //Selecting a random index within the size of the arrayList
521. randClubIndex = rand.nextInt(PremierLeagueManager.getInstance().getClubsInLeague().size());
522. randomTeams.add(PremierLeagueManager.getInstance().getClubsInLeague().get(randClubIndex));
523. }
524. }
525. }
526. **boolean** isLeapYear = **false**;
527. //Checking if the year is a leap year
528. **if** (year % 4 == 0) {
529. **if** (year % 100 == 0) {
530. **if** (year % 400 == 0) {
531. isLeapYear = **true**;
532. }
533. } **else** {
534. isLeapYear = **true**;
535. }
536. }
537. // Getting a number between 1 and 12 (includes)
538. **int** randMonth = rand.nextInt(12) + 1;
539. // Getting a number between 1 and 31 (includes)
540. **int** randDay = rand.nextInt(31) + 1;
541. // if it's a leap year and the month is february
542. **if** (isLeapYear && randMonth == 2) {
543. // Getting a number between 1 and 29 (includes)
544. randDay = rand.nextInt(29) + 1;
545. }
546. // For the months having 30 a days, getting a number between 1 and 30
547. **if** (randMonth == 4 || randMonth == 6 || randMonth == 9 || randMonth == 11) {
548. randDay = rand.nextInt(30) + 1;
549. }
550. **return** **new** Match<>((FootballClub) randomTeams.get(0), (FootballClub) randomTeams.get(1), scores.get(0), scores.get(1), **new** Date(randDay, randMonth, year));
551. }
552. **return** **null**;
553. }
555. /\*\*
556. \* toString method of the class concrete class PremierLeagueManager
557. \*/
558. @Override
559. **public** String toString() {
560. **return** "PremierLeagueManager[" +
561. "clubsInLeague= " + clubsInLeague +
562. ", matchesInLeague= " + matchesInLeague +
563. "]";
564. }
566. /\*\*
567. \* Equals method of the PremierLeagueManager
568. \*
569. \* @param o object containing any type
570. \* @return a boolean if the object is an instance of PremierLeagueManager
571. \*/
572. @Override
573. **public** **boolean** equals(Object o) {
574. **if** (**this** == o) **return** **true**;
575. **if** (!(o **instanceof** PremierLeagueManager)) **return** **false**;
576. PremierLeagueManager that = (PremierLeagueManager) o;
577. **return** Objects.equals(getClubsInLeague(), that.getClubsInLeague()) && Objects.equals(getMatchesInLeague(), that.getMatchesInLeague());
578. }
580. /\*\*
581. \* Hashcode method of the class PremierLeagueManager
582. \*
583. \* @return int containing the hashcode
584. \*/
585. @Override
586. **public** **int** hashCode() {
587. **return** Objects.hash(getClubsInLeague(), getMatchesInLeague());
588. }
590. }

#### ConsoleApplication.java

App/cli/ConsoleApplication.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** cli;
10. **import** entities.\*;
12. **import** java.awt.\*;
13. **import** java.io.IOException;
14. **import** java.net.URI;
15. **import** java.net.URISyntaxException;
16. **import** java.util.InputMismatchException;
17. **import** java.util.Scanner;
18. **import** java.util.regex.Matcher;
19. **import** java.util.regex.Pattern;
21. /\*\*
22. \* This class contains the Console Application to run the Premier League Manager.
23. \*
24. \* @author Radhika Ranasinghe
25. \* @version 1.0
26. \* @since 2020-11-15
27. \*/
29. **public** **class** ConsoleApplication {
30. **private** **static** PremierLeagueManager premierLeagueManager = PremierLeagueManager.getInstance();
32. /\*\*
33. \* This method contains the main method of the ConsoleApplication
34. \*
35. \* @param args String array of arguments
36. \*/
37. **public** **static** **void** main(String[] args) {
38. System.out.println("W E L C O M E   T O   T H E   P R E M I E R   L E A G U E   M A N A G E R");
39. // Declaring an int 'seasonYear' and assigning 0
40. **int** seasonYear = 0;
41. **try** {
42. // Prompting the user to input the season start year
43. seasonYear = getUserInputInt("Enter season start year of the Premier League");
44. } **catch** (InputMismatchException e) {
45. // Ensuring the entered year is a valid integer
46. System.out.println("Invalid input, Please enter an Integer!");
47. }
48. // Until the user inputs an year after 1992, prompting to input season year
49. **while** (seasonYear < 1992) {
50. System.out.println("The season start year can be only be an year after 1992!");
51. **try** {
52. seasonYear = getUserInputInt("Enter season start year of the Premier League");
53. } **catch** (InputMismatchException e) {
54. // Catching if the user inputs anything else other an integer
55. System.out.println("Invalid input, Please enter an Integer!");
56. }
57. }
59. // Declaring two strings as 'clubFileName' and 'matchFileName' and assigning them as follows;
60. String clubFileName = ("clubDataFile" + seasonYear + ".txt");
61. String matchFileName = ("matchDataFile" + seasonYear + ".txt");
63. // Retrieving all the data stored
64. **try** {
65. premierLeagueManager.retrieveClubData(clubFileName);
66. premierLeagueManager.retrieveMatchData(matchFileName);
67. } **catch** (IOException | ClassNotFoundException e) {
68. e.printStackTrace();
69. }
70. // Printing the menu
71. menuLoop:
72. **while** (**true**) {
73. System.out.println("\nMenu of the Premier League Manager" +
74. "\n1:   Add a Football Club to the Premier League" +
75. "\n2:   Delete an existing Football Club from the Premier League" +
76. "\n3:   Display statistics of a Football Club in the Premier League" +
77. "\n4:   Display the Premier League Table" +
78. "\n5:   Add a played match to the Premier League" +
79. "\n6:   Open GUI from the console" +
80. "\n7:   Exit the Premier League Manager");
81. **try** {
82. // Prompting the user a message and getting the selection
83. **int** userChoice = getUserInputInt("\nEnter the selection");
85. // For the selection the respective methods are called
86. **switch** (userChoice) {
87. **case** 1:
88. // Adding a club method is called
89. addClub();
90. **try** {
91. // After addition of a club the files are updated
92. premierLeagueManager.saveClubData(clubFileName);
93. premierLeagueManager.saveMatchData(matchFileName);
94. } **catch** (IOException e) {
95. e.printStackTrace();
96. }
97. **break**;
98. **case** 2:
99. // Deletion of a club method is called
100. deleteClub();
101. **try** {
102. // After deletion of a club the file are updated
103. premierLeagueManager.saveClubData(clubFileName);
104. premierLeagueManager.saveMatchData(matchFileName);
105. } **catch** (IOException e) {
106. e.printStackTrace();
107. }
108. **break**;
109. **case** 3:
110. //Displaying statistics of a club method is called
111. displayStatsOfAClub();
112. **break**;
113. **case** 4:
114. // Displaying premier league table method is called
115. displayPremierLeagueTable();
116. **break**;
117. **case** 5:
118. // Adding a played match method is called
119. addAPlayedMatch(seasonYear);
120. **try** {
121. // After addition the files are updated
122. premierLeagueManager.saveClubData(clubFileName);
123. premierLeagueManager.saveMatchData(matchFileName);
124. } **catch** (IOException e) {
125. e.printStackTrace();
126. }
127. **break**;
128. **case** 6:
129. // The GUI is opened in browser
130. String url = "http://localhost:4200/";
132. // Checking if the desktop supported on current platform
133. **if** (Desktop.isDesktopSupported()) {
134. Desktop desktop = Desktop.getDesktop();
135. **try** {
136. // Laughing browser
137. desktop.browse(**new** URI(url));
138. } **catch** (IOException | URISyntaxException e) {
139. e.printStackTrace();
140. }
141. } **else** {
142. Runtime runtime = Runtime.getRuntime();
143. **try** {
144. runtime.exec("xdg-open " + url);
145. } **catch** (IOException e) {
146. e.printStackTrace();
147. }
148. }
149. **break**;
150. **case** 7:
151. **try** {
152. // Saving into all data files
153. premierLeagueManager.saveClubData(clubFileName);
154. premierLeagueManager.saveMatchData(matchFileName);
155. } **catch** (IOException e) {
156. e.printStackTrace();
157. }
158. **break** menuLoop;
159. **default**:
160. // Checking if the user inputs a number between 1 and 7
161. **if** (userChoice == 0) {
162. **break**;
163. } **else** {
164. System.out.println("Please enter a number between 1 and 7!");
165. }
166. }
167. } **catch** (InputMismatchException e) {
168. System.out.println("Invalid input, Please enter an Integer!");
169. }
170. }
172. }
174. /\*\*
175. \* This method takes inputs from the user to add club and then validated before adding to the league
176. \*/
177. **public** **static** **void** addClub() {
178. // Declaring a String called 'clubName' and calling method getUserInputString()
179. String clubName = getUserInputString("Enter the name of the club");
181. // Declaring a String called 'clubLocation' and calling method getUserInputString()
182. String clubLocation = getUserInputString("Enter club location");
184. // Declaring a boolean called 'isPresent' and assigning false
185. **boolean** isPresent = **false**;
187. // Declaring a boolean called 'isAdded' and assigning false
188. **boolean** isAdded = **false**;
190. // Declaring a String called 'regexName'
191. String regexName = "^[a-zA-Z\\s]+";
193. // Declaring a String called 'regexLocation'
194. String regexLocation = "[a-zA-z]\*([,\\s]+[a-z]\*)\*";
196. // Declaring a Pattern called 'patternName'
197. Pattern patternName = Pattern.compile(regexName, Pattern.CASE\_INSENSITIVE);
199. // Declaring a Pattern called 'patternLocation'
200. Pattern patternLocation = Pattern.compile(regexLocation, Pattern.CASE\_INSENSITIVE);
202. // Declaring a Matcher called 'matcherName'
203. Matcher matcherName = patternName.matcher(clubName);
205. // Declaring a Matcher called 'matcherLocation'
206. Matcher matcherLocation = patternLocation.matcher(clubLocation);
208. //Iterating through the clubs in premier league
209. **for** (SportsClub club : premierLeagueManager.getClubsInLeague()) {
210. // Checking if club with same name and location exists
211. // Continue if the name and location is null
212. **if** (club.getName() == **null** || club.getLocation() == **null**) **continue**;
213. // If found, notify with the boolean isPresent
214. **if** (club.getName().equals(clubName) && club.getLocation().equals(clubLocation)) {
215. isPresent = **true**;
216. **break**;
217. }
218. }
219. // If the club is not present
220. **if** (!isPresent) {
221. **if** (matcherName.matches() && matcherLocation.matches()) {
222. FootballClub club = **new** FootballClub(clubName, clubLocation);
223. // Sending to the PremierLeagueManager to add the club
224. isAdded = premierLeagueManager.addClub(club);
225. } **else** {
226. // If the name doesn't match regex
227. System.out.println("Invalid Club Name or Club Location!");
228. }
229. }
231. // If the club is there all registered in the premier league, printing this
232. **if** (isAdded) {
233. System.out.println("The football club has been added to the Premier League!");
234. } **else** **if** (premierLeagueManager.getClubsInLeague().size() >= 20) {
235. // If the club is full with the maximum capacity
236. System.out.println("The Premier League is full!");
237. }
239. }
241. /\*\*
242. \* Method that takes in the user input of the club name to be deleted and passes the club name string to delete
243. \*/
244. **public** **static** **void** deleteClub() {
245. // Checking if there are clubs to be deleted
246. **if** (premierLeagueManager.getClubsInLeague().size() > 0) {
247. System.out.println("List of Football clubs in the Premier League");
249. // Listing out the clubs registered with the premier league manager
250. **for** (SportsClub club : premierLeagueManager.getClubsInLeague()) {
251. System.out.println("\t" + (premierLeagueManager.getClubsInLeague().indexOf(club) + 1) + ": " + club.getName());
252. }
253. // Declaring a String called 'clubName' and calling method getUserInputString()
254. String clubName = getUserInputString("Enter name of the club to be deleted from the list");
256. // Passing the clubName to delete from the PremierLeagueManager and taking the boolean returned from it as isDeleted
257. **boolean** isDeleted = premierLeagueManager.deleteClub(clubName);
259. // Checking if isDeleted is true
260. **if** (isDeleted) {
261. // Printed this if true
262. System.out.println("Successfully deleted the club " + clubName + "!");
263. } **else** {
264. // Printed this if false
265. System.out.println("A Football club with the name '" + clubName + "' is not registered in the Premier League");
266. }
267. } **else** {
268. // If there no clubs to be deleted
269. System.out.println("No registered clubs with the Premier League!");
270. }
271. }
273. /\*\*
274. \* Method to take in user inputs to add a played match and validate the data
275. \*
276. \* @param seasonYear the year user inputs as the season year
277. \*/
278. **public** **static** **void** addAPlayedMatch(**int** seasonYear) {
279. **try** {
280. // Checking if there are more than 1 club to play a match
281. **if** (premierLeagueManager.getClubsInLeague().size() > 1) {
283. // Declaring a FootballClub called 'homeClub'
284. FootballClub homeClub;
286. // Declaring a FootballClub called 'awayClub'
287. FootballClub awayClub;
289. // Listing out the Football clubs in the Premier league
290. System.out.println("List of Football clubs in the Premier League");
291. **for** (SportsClub club : premierLeagueManager.getClubsInLeague()) {
292. System.out.println("\t" + (premierLeagueManager.getClubsInLeague().indexOf(club) + 1) + ": " + club.getName());
293. }
295. // Getting the user inputs as int from the list shown
296. **int** homeClubNum = getUserInputInt("Enter number of the Home Club from the list");
297. **int** awayClubNum = getUserInputInt("Enter number of the Away Club from the list");
299. // Ensuring the taken inputs are not equal numbers
300. **if** (homeClubNum == awayClubNum) {
301. System.out.println("Home club and the away club cannot be the same club!");
303. } **else** **if** (homeClubNum > premierLeagueManager.getClubsInLeague().size() || homeClubNum < 0 ||
304. awayClubNum > premierLeagueManager.getClubsInLeague().size() || awayClubNum < 0) {
305. // The club numbers has to be within the range shown, else this will be printed
306. System.out.println("Invalid Club number!");
307. } **else** {
308. **int**[] commonDays = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
309. **int**[] leapDays = {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
311. // Assigning the selected clubs by the user
312. homeClub = (FootballClub) premierLeagueManager.getClubsInLeague().get(homeClubNum - 1);
313. awayClub = (FootballClub) premierLeagueManager.getClubsInLeague().get(awayClubNum - 1);
315. System.out.println();
317. // Getting user inputs for goals scored by each club as integers
318. **int** homeGoals = getUserInputInt("Enter number of goals scored by the home club");
319. **int** awayGoals = getUserInputInt("Enter number of goals scored by the away club");
321. // Declaring a boolean called 'isLeapYear' and assigning it to false
322. **boolean** isLeapYear = **false**;
324. // Getting user input for the year
325. **int** year = getUserInputInt("Enter year");
327. // The year can only be the season year and one year after the season year
328. **if** (year != seasonYear && year != (seasonYear + 1)) {
329. System.out.println("Please re-enter the match data!");
330. } **else** {
331. // Checking if the year is leap year
332. **if** (year % 4 == 0) {
333. **if** (year % 100 == 0) {
334. **if** (year % 400 == 0) {
335. isLeapYear = **true**;
336. }
337. } **else** {
338. isLeapYear = **true**;
339. }
340. }
341. // Getting user input for the month
342. **int** month = getUserInputInt("Enter month");
344. // Checking if the month is in range
345. **if** (month > 13 || month < 0) {
346. System.out.println("Please re-enter the match data!");
347. } **else** {
348. // Getting user inputs for the day
349. **int** day = getUserInputInt("Enter day");
351. // Declaring a boolean to only added a match if the data is correct
352. **boolean** isCorrect = **true**;
353. **if** (isLeapYear) {
354. // Checking if the day is in range
355. **if** (day > leapDays[month] || day < 0) {
356. System.out.println("Please re-enter the match data!");
357. isCorrect = **false**;
358. }
359. } **else** {
360. // Checking if the day is in range
361. **if** (day > commonDays[month] || day < 0) {
362. System.out.println("Please re-enter the match data!");
363. isCorrect = **false**;
364. }
365. }
367. // Checking if the data is correct
368. **if** (isCorrect) {
369. // Declaring a match with the user input match data
370. Match<FootballClub> match = **new** Match<>(homeClub, awayClub, homeGoals, awayGoals,
371. **new** Date(day, month, year));
372. **boolean** isAdded = **false**;
374. **for** (Match<FootballClub> matchInList : premierLeagueManager.getMatchesInLeague()) {
375. // Checking if the match is already added to the premier league
376. **if** (matchInList.equals(match)) {
377. isAdded = **true**;
378. System.out.println("Match has been already added to the Premier League!");
379. **break**;
380. }
381. }
382. // If its not already added
383. **if** (!isAdded) {
384. // Then it's being added
385. premierLeagueManager.addMatch(match);
386. System.out.println("Successfully added the match!");
387. }
388. }
389. }
390. }
391. }
392. } **else** {
393. // Printing this if less than one club is there
394. System.out.println("Not enough registered clubs to play a match!");
395. }
396. } **catch** (InputMismatchException e) {
397. // Catching when the user inputs anything when asked for an integer
398. System.out.println("Invalid input, Please enter an Integer!");
399. }
400. }
402. /\*\*
403. \* Method that takes in the user input of which club they prefer to get statistics of
404. \*/
405. **public** **static** **void** displayStatsOfAClub() {
406. // Checking if the premier league has registered clubs
407. **if** (premierLeagueManager.getClubsInLeague().size() > 0) {
408. // listing out all the registered clubs in the league
409. System.out.println("\nList of Football clubs in the Premier League");
410. **for** (SportsClub club : premierLeagueManager.getClubsInLeague()) {
411. System.out.println("\t" + (premierLeagueManager.getClubsInLeague().indexOf(club) + 1) + ": " + club.getName());
412. }
413. **try** {
414. // Getting the user input for user selection from the list
415. **int** userSelection = getUserInputInt("\nEnter the number of the preferred club");
417. // Checking if the user selection is in range
418. **if** (userSelection != 0) {
419. // Displaying the statistics
420. String displayStat = premierLeagueManager.getStatsOfAClub((FootballClub) premierLeagueManager.getClubsInLeague().get(userSelection - 1));
421. System.out.println(displayStat);
422. }
423. } **catch** (IndexOutOfBoundsException e) {
424. System.out.println("Invalid Selection!");
425. }
426. } **else** {
427. // Printing this if less than one club is there
428. System.out.println("No registered clubs with the Premier League!");
429. }
430. }
432. /\*\*
433. \* Method to display the premier league table to the console
434. \*/
435. **public** **static** **void** displayPremierLeagueTable() {
436. // Checking if the premier league has registered clubs
437. **if** (premierLeagueManager.getClubsInLeague().size() > 0) {
438. // Printing the Premier league table
439. System.out.println(premierLeagueManager.getStatsOfAllClubs());
440. } **else** {
441. // Printing this if less than one club is there
442. System.out.println("No registered clubs with the Premier League!");
443. }
444. }
446. /\*\*
447. \* Method to get a String input from the user
448. \*
449. \* @param promptMessage the message to prompt when getting the input
450. \* @return String input given by the user
451. \*/
452. **public** **static** String getUserInputString(String promptMessage) {
453. // Declaring a String called 'inputLine'
454. String inputLine;
456. // Declaring a Scanner called 'scanner'
457. Scanner scanner = **new** Scanner(System.in);
459. // Prompting a message to the user
460. System.out.print(promptMessage + ": ");
462. // Taking the user's input from the scanner
463. inputLine = scanner.nextLine();
464. **return** inputLine;
465. }
467. /\*\*
468. \* Method to get user inputs of Integers
469. \*
470. \* @param promptMessage message to display when getting the input from the user
471. \* @return int the input provided by the user
472. \* @throws InputMismatchException indicates that the token retrieved does not match the pattern for the expected type
473. \*/
474. **public** **static** **int** getUserInputInt(String promptMessage) **throws** InputMismatchException {
475. // Declaring a int called 'userChoice'
476. **int** userChoice = 0;
478. // Declaring a Scanner called 'scanner'
479. Scanner scanner = **new** Scanner(System.in);
481. // Prompting a message to the user
482. System.out.print(promptMessage + ": ");
484. // Taking the user's input from the scanner
485. userChoice = scanner.nextInt();
486. **return** userChoice;
487. }

490. }

### Utils Package

#### ApplicationUtil.java

App/utils/ApplicationUtil.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** utils;
10. **import** com.fasterxml.jackson.databind.JsonNode;
11. **import** com.fasterxml.jackson.databind.node.ObjectNode;
12. **import** play.libs.Json;
14. /\*\*
15. \* This is class includes the methods for the Application to use as utilities
16. \*
17. \* @author Radhika Ranasinghe
18. \* @version 1.0
19. \* @since 2020-12-12
20. \*/
22. **public** **class** ApplicationUtil {
23. /\*\*
24. \* This methods is utilized in every instance of creating a response to Json file
25. \* @param response from the Object type
26. \* @param ok boolean to show the state of response
27. \* @return ObjectNode containing the response and the status
28. \*/
29. **public** **static** ObjectNode createResponse(Object response, **boolean** ok) {
30. ObjectNode result = Json.newObject();
31. result.put("status", ok);
32. // Check if the object is in an instance of a string
33. **if** (response **instanceof** String)
34. // then put the response as a String
35. result.put("response", (String) response);
36. // Else set the response as a JsonNode
37. **else** result.set("response", (JsonNode) response );
38. **return** result;
39. }
41. }

### Controllers Package

#### PremierLeagueController.java

App/controllers/PremierLeagueController.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** controllers;
10. **import** com.fasterxml.jackson.databind.JsonNode;
11. **import** com.fasterxml.jackson.databind.ObjectMapper;
12. **import** entities.\*;
13. **import** org.slf4j.Logger;
14. **import** org.slf4j.LoggerFactory;
15. **import** play.libs.Json;
16. **import** play.mvc.Http;
17. **import** play.mvc.Result;
18. **import** utils.ApplicationUtil;
20. **import** java.io.IOException;
21. **import** java.util.ArrayList;
22. **import** java.util.Collections;
23. **import** java.util.Comparator;
24. **import** java.util.List;
26. **import** **static** play.mvc.Results.created;
27. **import** **static** play.mvc.Results.ok;
29. /\*\*
30. \* This is class includes all the controller functionalities.
31. \*
32. \* @author Radhika Ranasinghe
33. \* @version 1.0
34. \* @since 2020-12-12
35. \*/
36. @SuppressWarnings("unchecked")
37. **public** **class** PremierLeagueController {
39. **private** **static** **final** Logger LOGGER = LoggerFactory.getLogger("controller");
41. /\*\*
42. \* Method to add the already generated random match on given year by the user
43. \*
44. \* @param year    given year by the user
45. \* @param request Http.Request
46. \* @return Result containing response created by application util
47. \*/
48. **public** Result createRandomMatch(**int** year, Http.Request request) {
49. **try** {
50. // Loading the data files according year user has given
51. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year + ".txt");
52. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year + ".txt");
54. } **catch** (IOException | ClassNotFoundException e) {
55. LOGGER.debug("An exception has occurred");
56. }
58. // Taking request's body only
59. JsonNode jsonNode = request.body().asJson();
61. // Declaring a Match called 'match' and assigning the jsonNode to it
62. Match<FootballClub> match = Json.fromJson(jsonNode, Match.**class**);
64. // Adding the match to Premier League
65. Match<FootballClub> addedMatch = PremierLeagueManager.getInstance().addMatch(match);
67. **try** {
69. // Saving the data files according to year user has given
70. PremierLeagueManager.getInstance().saveClubData("clubDataFile" + year + ".txt");
71. PremierLeagueManager.getInstance().saveMatchData("matchDataFile" + year + ".txt");
73. } **catch** (IOException e) {
74. LOGGER.debug("An exception has occurred");
75. }
76. // Creating a JsonNode with match
77. JsonNode reply = Json.toJson(addedMatch);
78. **return** created(ApplicationUtil.createResponse(reply, **true**));
79. }
81. /\*\*
82. \* Method to generate a random match on a given year by the user
83. \*
84. \* @param year given year by the user
85. \* @return Result containing response created by application util
86. \*/
87. **public** Result retrieveMatch(**int** year) {
88. **try** {
89. // Loading the data files according year user has given
90. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year + ".txt");
91. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year + ".txt");
93. } **catch** (IOException | ClassNotFoundException e) {
94. LOGGER.debug("An exception has occurred");
95. }
96. // Declaring a JsonNode called 'jsonObject' and generating Random match and converting to Json
97. JsonNode jsonObject = Json.toJson(PremierLeagueManager.getInstance().generateRandomMatch(year));
98. LOGGER.debug("In EmployeeController.retrieve(), result is: {}", jsonObject.toString());
99. **return** ok(ApplicationUtil.createResponse(jsonObject, **true**));
100. }
102. /\*\*
103. \* Method to get all the clubs in the premier league registered in the year given by the user
104. \*
105. \* @param year given by the user
106. \* @return Result containing response created by application util
107. \*/
108. **public** Result listClubs(**int** year) {
109. **try** {
110. // Loading the data files according year user has given
111. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year + ".txt");
112. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year + ".txt");
114. } **catch** (IOException | ClassNotFoundException e) {
115. LOGGER.debug("An exception has occurred");
117. }
118. // Getting the clubs to the list
119. List<SportsClub> result = PremierLeagueManager.getInstance().getClubsInLeague();
121. // Sort the by the points
122. result.sort(Collections.reverseOrder());
123. LOGGER.debug("In PremierLeagueManager.listFootballClubs(), result is: {}", result.toString());
125. //Declaring an object mapper
126. ObjectMapper mapper = **new** ObjectMapper();
128. // Giving the result to object mapper to convert JsonNode
129. JsonNode jsonData = mapper.convertValue(result, JsonNode.**class**);
130. **return** ok(ApplicationUtil.createResponse(jsonData, **true**));
131. }
133. /\*\*
134. \* Method to get all the added matches played by the clubs registered in Premier league in the year given by the user
135. \*
136. \* @param year given by the user
137. \* @return Result containing response created by application util
138. \*/
139. **public** Result listMatches(**int** year) {
140. **try** {
141. // Loading the data files according year user has given
142. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year + ".txt");
143. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year + ".txt");
144. } **catch** (IOException | ClassNotFoundException e) {
145. LOGGER.debug("An exception has occurred");
146. }
147. // Getting the clubs to the list
148. List<Match<FootballClub>> result = PremierLeagueManager.getInstance().getMatchesInLeague();
150. // Sort the by the date
151. Collections.sort(result);
153. LOGGER.debug("In PremierLeagueManager.listMatches(), result is: {}", result.toString());
155. //Declaring an object mapper
156. ObjectMapper mapper = **new** ObjectMapper();
158. // Giving the result to object mapper to convert JsonNode
159. JsonNode jsonData = mapper.convertValue(result, JsonNode.**class**);
160. **return** ok(ApplicationUtil.createResponse(jsonData, **true**));
161. }
163. /\*\*
164. \* Method to sort the clubs registered in the Premier League by their Goals Scored
165. \*
166. \* @param year given by the user
167. \* @return Result containing response created by application util
168. \*/
169. **public** Result sortByGoals(**int** year) {
170. **try** {
171. // Loading the data files according year user has given
172. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year + ".txt");
173. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year + ".txt");
174. } **catch** (IOException | ClassNotFoundException e) {
175. LOGGER.debug("An exception has occurred");
176. }
177. // Getting the clubs to the list
178. List<SportsClub> result = PremierLeagueManager.getInstance().getClubsInLeague();
180. // Creating a Comparator to compare by goals
181. Comparator<SportsClub> compareByGoals = Comparator.comparingInt(o -> ((FootballClub) o).getGoalsScored());
183. // Taking the result and sorting in the descending order
184. result.sort(compareByGoals.reversed());
186. //Declaring an object mapper
187. ObjectMapper mapper = **new** ObjectMapper();
189. // Giving the result to object mapper to convert JsonNode
190. JsonNode jsonData = mapper.convertValue(result, JsonNode.**class**);
191. **return** ok(ApplicationUtil.createResponse(jsonData, **true**));
192. }
194. /\*\*
195. \* Method to sort the clubs registered in the Premier League by their Wins
196. \*
197. \* @param year given by the user
198. \* @return Result containing response created by application util
199. \*/
200. **public** Result sortByWins(**int** year) {
201. **try** {
202. // Loading the data files according year user has given
203. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + year + ".txt");
204. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + year + ".txt");
205. } **catch** (IOException | ClassNotFoundException e) {
206. LOGGER.debug("An exception has occurred");
207. }
208. // Getting the clubs to the list
209. List<SportsClub> result = PremierLeagueManager.getInstance().getClubsInLeague();
211. // Creating a Comparator to compare by wins
212. Comparator<SportsClub> compareByWins = Comparator.comparingInt(o -> ((FootballClub) o).getWins());
214. // Taking the result and sorting in the descending order
215. result.sort(compareByWins.reversed());
217. //Declaring an object mapper
218. ObjectMapper mapper = **new** ObjectMapper();
220. // Giving the result to object mapper to convert JsonNode
221. JsonNode jsonData = mapper.convertValue(result, JsonNode.**class**);
222. **return** ok(ApplicationUtil.createResponse(jsonData, **true**));
223. }
225. }

## Angular Source Code

### Dashboard-home

#### Dashboard-home.component.html

1. **<div** class="grid-container"**>**
2. **<mat-card** class="home-main"**>**
4. **<div** class="home-main-text"**>**
5. **<h1>**WELCOME TO PREMIER LEAGUE MANAGER!**</h1>**
6. **</div>**
7. **<div** class="home-img-container-top"**>**
8. **<div** class="home-page-images"**>**
9. **<img** src="assets/images/svg/Cup.svg" alt="cup"**>**
10. **</div>**
11. **<div** class="home-page-images"**>**
12. **<img** src="assets/images/svg/Red%20Card.svg" alt="red"**>**
13. **</div>**
14. **<div** class="home-page-images"**>**
15. **<img** src="assets/images/svg/Game%20Start.svg" alt="game"**>**
16. **</div>**
17. **<div** class="home-page-images"**>**
18. **<img** src="assets/images/svg/Livello%2010.svg" alt="level"**>**
19. **</div>**
20. **<div** class="home-page-images"**>**
21. **<img** src="assets/images/svg/Penalty%20Kick.svg" alt="penalty"**>**
22. **</div>**
23. **<div** class="home-page-images"**>**
24. **<img** src="assets/images/svg/Player.svg" alt="player"**>**
25. **</div>**
26. **<div** class="home-page-images"**>**
27. **<img** src="assets/images/svg/Foul.svg" alt="foul"**>**
28. **</div>**
29. **<div** class="home-page-images"**>**
30. **<img** src="assets/images/svg/Kick.svg" alt="kick"**>**
31. **</div>**
32. **</div>**
33. **<div** class="home-img-container"**>**
34. **<div** class="home-page-images"**>**
35. **<img** src="assets/images/svg/Cup.svg" alt="cup"**>**
36. **</div>**
37. **<div** class="home-page-images"**>**
38. **<img** src="assets/images/svg/Corner.svg" alt="corner"**>**
39. **</div>**
40. **<div** class="home-page-images"**>**
41. **<img** src="assets/images/svg/Game%20Start.svg" alt="game"**>**
42. **</div>**
43. **<div** class="home-page-images"**>**
44. **<img** src="assets/images/svg/Yellow%20Card.svg" alt="yellow"**>**
45. **</div>**
46. **<div** class="home-page-images"**>**
47. **<img** src="assets/images/svg/Penalty%20Kick.svg" alt="penalty"**>**
48. **</div>**
49. **<div** class="home-page-images"**>**
50. **<img** src="assets/images/svg/Player.svg" alt="player"**>**
51. **</div>**
52. **<div** class="home-page-images"**>**
53. **<img** src="assets/images/svg/Foul.svg" alt="foul"**>**
54. **</div>**
55. **<div** class="home-page-images"**>**
56. **<img** src="assets/images/svg/Kick.svg" alt="kick"**>**
57. **</div>**
58. **</div>**
60. **</mat-card>**
61. **</div>**

#### Dashboard-home.component.css

1. .grid-container {
2. **margin**: 20px;
3. }
5. .dashboard-card {
6. **position**: absolute;
7. top: 15px;
8. left: 15px;
9. right: 15px;
10. bottom: 15px;
11. }
13. .more-button {
14. **position**: absolute;
15. top: 5px;
16. right: 10px;
17. }
19. .dashboard-card-content {
20. **text-align**: center;
21. }
22. mat-card.mat-card.mat-focus-indicator.home-main {
23. **background-image**: linear-gradient(45deg,#147ca021, #2e8bc0);
24. **color**: white;
25. **height**: 78vh;
26. **display**: flex;
27. justify-**content**: center;
28. **margin**: auto;
29. }
30. .home-main-text h1 {
31. **position**: relative;
32. **margin**: auto;
33. **font-size**: 35px !important;
34. **font-weight**: 500;
35. }
36. .home-main-text {
37. **position**: absolute;
38. **height**: 100%;
39. **display**: flex;
40. }
42. .home-page-images img {
43. **width**: 100px;
44. **height**: auto;
45. }
46. .home-img-container {
47. **display**: flex;
48. justify-**content**: space-between;
49. **width**: 95%;
50. **margin**: auto;
51. **position**: absolute;
52. top: 75%;
53. }
55. .home-img-container-top{
56. **display**: flex;
57. justify-**content**: space-between;
58. **width**: 95%;
59. **margin**: auto;
60. **position**: absolute;
61. top: 10%;
62. flex-**direction**: row-reverse;
63. }

#### Dashboard-home.component.ts

1. **import** { Component } from '@angular/core';
3. @Component({
4. selector: 'app-dashboard-home',
5. templateUrl: './dashboard-home.component.html',
6. styleUrls: ['./dashboard-home.component.css']
7. })
8. **export** **class** DashboardHomeComponent {
10. }

Dialog-add-match

#### Dialog-add-match.component.html

1. **<h2** mat-dialog-title**>**Success!**</h2>**
2. **<mat-dialog-content>**The generated random match was added to the premier league.**</mat-dialog-content>**
3. **<mat-dialog-actions>**
4. **<button** mat-button mat-dialog-close="true" routerLink="/home"**>**Okay**</button>**
5. **</mat-dialog-actions>**

#### Dialog-add-match.component.ts

1. **import** { Component, OnInit } from '@angular/core';
3. @Component({
4. selector: 'app-dialog-add-match',
5. templateUrl: './dialog-add-match.component.html',
6. styleUrls: ['./dialog-add-match.component.css']
7. })
8. **export** **class** DialogAddMatchComponent **implements** OnInit {
9. /\*\*
10. \* Default constructor for DialogAddMatchComponent
11. \*/
12. constructor() { }
14. ngOnInit(): **void** {
15. }
17. }

### League-table.component

#### League-table.component.html

1. **<div** class="ltc-wrapper"**>**
2. **<div** class="buttons-ltc-container"**>**
3. **<button** mat-raised-button color="primary" (click)="sortByGoals()" **>**Sort By Goals**</button>**
4. **<button** mat-raised-button color="primary" (click)="sortByWins()"**>**Sort By Wins**</button>**
5. **<button** mat-raised-button color="primary" (click)="sortByPoints()"**>**Sort By Points**</button>**
6. **</div>**
7. **<mat-card>**
8. **<label>**Select year to proceed:
9. **<input** [(ngModel)]="premierLeagueYear" (change)="yearChanges()" type="number" min="1992" MAX="2999" required**/>**
10. **</label>**
11. **</mat-card>**
13. **<div** class="mat-elevation-z8"**>**
14. **<table** mat-table class="full-width-table" matSort aria-label="Elements"**>**
15. <!-- Name Column -->
16. **<ng-container** matColumnDef="name"**>**
17. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Club Name**</th>**
18. **<td** mat-cell \*matCellDef="let row"**>**{{row.name}}**</td>**
19. **</ng-container>**
21. <!-- Location Column -->
22. **<ng-container** matColumnDef="location"**>**
23. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Club Location**</th>**
24. **<td** mat-cell \*matCellDef="let row"**>**{{row.location}}**</td>**
25. **</ng-container>**
27. <!-- Played Matches Column -->
28. **<ng-container** matColumnDef="matchesPlayed"**>**
29. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Played Matches**</th>**
30. **<td** mat-cell \*matCellDef="let row"**>**{{row.matchesPlayed}}**</td>**
31. **</ng-container>**
33. <!-- Wins Column -->
34. **<ng-container** matColumnDef="wins"**>**
35. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Won**</th>**
36. **<td** mat-cell \*matCellDef="let row"**>**{{row.wins}}**</td>**
37. **</ng-container>**
39. <!-- Loss Column -->
40. **<ng-container** matColumnDef="defeats"**>**
41. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Loss**</th>**
42. **<td** mat-cell \*matCellDef="let row"**>**{{row.defeats}}**</td>**
43. **</ng-container>**
45. <!-- Draws Column -->
46. **<ng-container** matColumnDef="draws"**>**
47. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Drawn**</th>**
48. **<td** mat-cell \*matCellDef="let row"**>**{{row.draws}}**</td>**
49. **</ng-container>**

52. <!-- Goals Scored Column -->
53. **<ng-container** matColumnDef="goalsScored"**>**
54. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**GF**</th>**
55. **<td** mat-cell \*matCellDef="let row"**>**{{row.goalsScored}}**</td>**
56. **</ng-container>**

59. <!-- Goals Received Column -->
60. **<ng-container** matColumnDef="goalsReceived"**>**
61. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**GA**</th>**
62. **<td** mat-cell \*matCellDef="let row"**>**{{row.goalsReceived}}**</td>**
63. **</ng-container>**
65. <!-- Goals Difference Column -->
66. **<ng-container** matColumnDef="goalDifference"**>**
67. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**GD**</th>**
68. **<td** mat-cell \*matCellDef="let row"**>**{{row.goalDifference}}**</td>**
69. **</ng-container>**

72. <!-- Current Number of Points Column -->
73. **<ng-container** matColumnDef="points"**>**
74. **<th** mat-header-cell \*matHeaderCellDef mat-sort-header**>**Points**</th>**
75. **<td** mat-cell \*matCellDef="let row"**>**{{row.points}}**</td>**
76. **</ng-container>**
78. **<tr** mat-header-row \*matHeaderRowDef="displayedColumns"**></tr>**
79. **<tr** mat-row \*matRowDef="let row; columns: displayedColumns;"**></tr>**
80. **</table>**
82. **<mat-paginator** #paginator
83. [length]="dataSource?.data.length"
84. [pageIndex]="0"
85. [pageSize]="8"
86. [pageSizeOptions]="[25, 50, 100, 250]"**>**
87. **</mat-paginator>**
88. **</div>**
90. **</div>**

#### League-table.component.css

1. .full-width-table {
2. **width**: 100%;
3. }
4. button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary {
5. **margin**: 10px;
6. **background**: #1a5379;
7. **color**: white;
8. }
9. input.ng-pristine.ng-invalid.ng-touched {
10. **background**: #b1d4e0;
11. border-radius: 5px;
12. **width**: 100px;
13. **height**: 20px;
14. **border**: none;
15. }
16. input.ng-pristine.ng-invalid{
17. **background**: #b1d4e0;
18. border-radius: 5px;
19. **width**: 100px;
20. **height**: 20px;
21. **border**: none;
22. }
23. input.ng-touched.ng-dirty.ng-valid{
24. **background**: #b1d4e0 !important;
25. border-radius: 5px !important;
26. **width**: 100px !important;
27. **height**: 20px !important;
28. **border**: 1px solid black !important;
29. }
31. .ltc-wrapper {
32. **padding**: 50px;
33. **background**: #b1d4e0;
34. **height**: 100vh;
35. }
37. .buttons-ltc-container {
38. **width**: 100%;
39. **display**: flex;
40. justify-**content**: flex-end;
41. }
42. mat-card.mat-card.mat-focus-indicator{
43. **margin-bottom**: 10px;
44. **width**: 96%;
45. **margin-left**: 6px
46. }
47. .mat-elevation-z8 {
48. **width**: 99%;
49. **margin-left**: 5px;
50. }

#### League-table.component.ts

1. **import** {AfterViewInit, Component, OnInit, ViewChild} from '@angular/core';
2. **import** {MatPaginator} from '@angular/material/paginator';
3. **import** {MatSort} from '@angular/material/sort';
4. **import** {MatTable} from '@angular/material/table';
5. **import** {Footballclub} from '../footballclub';
6. **import** {LeagueTableDataSource} from './league-table-datasource';
7. **import** {PremierLeagueService} from '../premier-league.service';
8. **import** {PremierLeagueYearService} from '../premier-league-year.service';
10. @Component({
11. selector: 'app-league-table',
12. templateUrl: './league-table.component.html',
13. styleUrls: ['./league-table.component.css']
14. })
15. **export** **class** LeagueTableComponent **implements** OnInit {
16. @ViewChild(MatPaginator) paginator: MatPaginator;
17. @ViewChild(MatSort) sort: MatSort;
18. @ViewChild(MatTable) table: MatTable<Footballclub>;
19. dataSource: LeagueTableDataSource;
20. clubData: Footballclub;
22. /\*\* Columns displayed in the table. Columns IDs can be added, removed, or reordered. \*/
23. displayedColumns = ['name', 'location', 'matchesPlayed', 'wins', 'defeats', 'draws', 'goalsScored', 'goalsReceived', 'goalDifference', 'points'];
24. premierLeagueYear;
26. /\*\*
27. \* constructor of league table
28. \*/
29. constructor(**private** service: PremierLeagueService, **private** serviceYear: PremierLeagueYearService) {
30. }
32. /\*\*
33. \* when the constructor is called sortByPoints() is called
34. \*/
35. ngOnInit(): any {
36. **this**.sortByPoints();
37. }
39. sortByWins(): **void** {
40. **this**.serviceYear.premierLeagueYear.subscribe((year) => {
41. **this**.service.getFootballClubDataSortedByWins(year).subscribe((r: Footballclub[]) => {
42. **this**.dataSource = **new** LeagueTableDataSource();
43. **this**.dataSource.data = r;
44. **this**.dataSource.sort = **this**.sort;
45. **this**.dataSource.paginator = **this**.paginator;
46. **this**.table.dataSource = **this**.dataSource;
47. });
48. });
49. }
51. sortByGoals(): **void** {
52. **this**.serviceYear.premierLeagueYear.subscribe((year) => {
53. **this**.service.getFootballClubDataSortedByGoals(year).subscribe((r: Footballclub[]) => {
54. **this**.dataSource = **new** LeagueTableDataSource();
55. **this**.dataSource.data = r;
56. **this**.dataSource.sort = **this**.sort;
57. **this**.dataSource.paginator = **this**.paginator;
58. **this**.table.dataSource = **this**.dataSource;
59. });
60. });
61. }
63. sortByPoints(): **void** {
64. **this**.serviceYear.premierLeagueYear.subscribe((year) => {
65. **this**.service.getFootballClubData(year).subscribe((r: Footballclub[]) => {
66. **this**.dataSource = **new** LeagueTableDataSource();
67. **this**.dataSource.data = r;
68. **this**.dataSource.sort = **this**.sort;
69. **this**.dataSource.paginator = **this**.paginator;
70. **this**.table.dataSource = **this**.dataSource;
71. });
72. });
73. }
75. yearChanges(): **void** {
76. **this**.serviceYear.premierLeagueYear.next(String(**this**.premierLeagueYear));
77. }
79. }

#### League-table-datasoure.ts

1. **import** {DataSource} from '@angular/cdk/collections';
2. **import** {MatPaginator} from '@angular/material/paginator';
3. **import** {MatSort} from '@angular/material/sort';
4. **import** {map} from 'rxjs/operators';
5. **import** {Observable, of as observableOf, merge} from 'rxjs';
6. **import** {Footballclub} from '../footballclub';

9. /\*\*
10. \* Data source for the LeagueTable view. This class should
11. \* encapsulate all logic for fetching and manipulating the displayed data
12. \* (including sorting, pagination, and filtering).
13. \*/
14. **export** **class** LeagueTableDataSource **extends** DataSource<Footballclub> {
15. data: any = [];
16. paginator: MatPaginator;
17. sort: MatSort;
19. constructor() {
20. **super**();
21. }
23. /\*\*
24. \* Connect this data source to the table. The table will only update when
25. \* the returned stream emits new items.
26. \* @returns A stream of the items to be rendered.
27. \*/
28. connect(): Observable<Footballclub[]> {
29. // Combine everything that affects the rendered data into one update
30. // stream for the data-table to consume.
31. **const** dataMutations = [
32. observableOf(**this**.data),
33. **this**.paginator.page,
34. **this**.sort.sortChange
35. ];
37. **return** merge(...dataMutations).pipe(map(() => {
38. **return** **this**.getPagedData(**this**.getSortedData([...**this**.data]));
39. }));
40. }
42. /\*\*
43. \*  Called when the table is being destroyed. Use this function, to clean up
44. \* any open connections or free any held resources that were set up during connect.
45. \*/
46. disconnect(): **void** {
47. }
49. /\*\*
50. \* Paginate the data (client-side). If you're using server-side pagination,
51. \* this would be replaced by requesting the appropriate data from the server.
52. \*/
53. **private** getPagedData(data: Footballclub[]): any {
54. **const** startIndex = **this**.paginator.pageIndex \* **this**.paginator.pageSize;
55. **return** data.splice(startIndex, **this**.paginator.pageSize);
56. }
58. /\*\*
59. \* Sort the data (client-side). If you're using server-side sorting,
60. \* this would be replaced by requesting the appropriate data from the server.
61. \*/
62. **private** getSortedData(data: Footballclub[]): any {
63. **if** (!**this**.sort.active || **this**.sort.direction === '') {
64. **return** data;
65. }
67. **return** data.sort((a, b) => {
68. **const** isAsc = **this**.sort.direction === 'asc';
69. **switch** (**this**.sort.active) {
70. **case** 'goalsScored':
71. **return** compare(a.goalsScored, b.goalsScored, isAsc);
72. **default**:
73. **return** 0;
74. }
75. });
76. }
77. }
79. /\*\* Simple sort comparator for example ID/Name columns (for client-side sorting). \*/
80. **function** compare(a: string | number, b: string | number, isAsc: **boolean**): any {
81. **return** (a < b ? -1 : 1) \* (isAsc ? 1 : -1);
82. }

### Nav-side-bar.component

#### Nav-side-bar.component.html

1. **<mat-sidenav-container** class="sidenav-container"**>**
2. **<mat-sidenav** #drawer class="sidenav" fixedInViewport
3. [attr.role]="(isHandset$ | async) ? 'dialog' : 'navigation'"
4. [mode]="(isHandset$ | async) ? 'over' : 'side'"
5. [opened]="(isHandset$ | async) === false"**>**
6. **<mat-toolbar>**Menu**</mat-toolbar>**
7. **<div** id="menu-nav-container"**>**
8. **<mat-nav-list>**
9. **<a** mat-list-item href="/home"**><mat-icon>**home**</mat-icon>**Home**</a>**
10. **<a** mat-list-item href="/league-table"**><mat-icon>**web\_asset**</mat-icon>**League Table**</a>**
11. **<a** mat-list-item href="/random-match"**><mat-icon>**shuffle**</mat-icon>**Random Match**</a>**
12. **<a** mat-list-item href="/played-matches"**><mat-icon>**sports\_soccer**</mat-icon>**Played Matches**</a>**
13. **</mat-nav-list>**
14. **</div>**
15. **</mat-sidenav>**
16. **<mat-sidenav-content>**
17. **<mat-toolbar** color="primary"**>**
18. **<button**
19. type="button"
20. aria-label="Toggle sidenav"
21. mat-icon-button
22. (click)="drawer.toggle()"
23. \*ngIf="isHandset$ | async"**>**
24. **<mat-icon** aria-label="Side nav toggle icon"**>**menu**</mat-icon>**
25. **</button>**
26. **<span>**Premier League Manager **</span>**
27. **</mat-toolbar>**
28. **<router-outlet></router-outlet>**
29. **</mat-sidenav-content>**
30. **</mat-sidenav-container>**

#### Nav-side-bar.component.css

1. @font-face {
2. **font-family**: 'Varela', sans-serif;
3. **src**: url("https://fonts.googleapis.com/css2?family=Varela&display=swap");
4. }
6. .sidenav-container {
7. **height**: 100%;
8. **font-family**: 'Varela', sans-serif;
9. }
11. .sidenav {
12. **width**: 200px;
13. }
15. .sidenav .mat-toolbar {
16. **background**: inherit;
17. }
19. .mat-drawer-inner-container.ng-tns-c159-0 {
20. **background**: #0C2D48 !important;
21. }
22. mat-toolbar.mat-toolbar.ng-tns-c153-0.mat-toolbar-single-row {
23. **color**: white;
24. }
25. mat-sidenav.mat-drawer.mat-sidenav.sidenav.ng-tns-c153-0.ng-trigger.ng-trigger-transform.mat-drawer-side.mat-drawer-opened.mat-sidenav-fixed.ng-star-inserted {
26. **background**: linear-gradient(72deg, #0c2d48, #2e8bc0);
27. **border**: none;
28. }
29. .mat-toolbar.mat-primary {
30. **position**: sticky;
31. top: 0;
32. **z-index**: 1;
33. **background**: linear-gradient(45deg, #2c84b7, #103856);
34. }
35. mat-icon.mat-icon.notranslate.material-icons.mat-icon-no-color {
36. **margin-right**: 8px;
37. }
38. #menu-nav-container {
39. **padding-top**: 200px;
40. }
42. .mat-nav-list {
43. **padding**: 5px;
44. **font-family**: 'Varela', sans-serif;
46. }
48. #menu-nav-container {
49. **padding-top**: 200px;
50. }
52. a.mat-list-item.mat-focus-indicator {
53. **color**: white !important;
54. }
56. mat-toolbar.mat-toolbar.ng-tns-c159-0.mat-toolbar-single-row {
57. **color**: white !important;
58. }

#### Nav-side-bar.component.ts

1. **import** {Component} from '@angular/core';
2. **import** {BreakpointObserver, Breakpoints} from '@angular/cdk/layout';
3. **import** {Observable} from 'rxjs';
4. **import** {map, shareReplay} from 'rxjs/operators';
5. **import** {PremierLeagueYearService} from '../premier-league-year.service';
7. @Component({
8. selector: 'app-nav-side-bar',
9. templateUrl: './nav-side-bar.component.html',
10. styleUrls: ['./nav-side-bar.component.css']
11. })
12. **export** **class** NavSideBarComponent {
13. constructor(**private** breakpointObserver: BreakpointObserver, **private** service: PremierLeagueYearService) {
14. }
16. isHandset$: Observable<**boolean**> = **this**.breakpointObserver.observe(Breakpoints.Handset)
17. .pipe(
18. map(result => result.matches),
19. shareReplay()
20. );
22. }

### Played-matches.component

#### Played.matches.component.html

1. **<div** class="main-wrapper"**>**
2. **<mat-card>**
3. **<label>**Select year to show matches played on the year:
4. **<input** [(ngModel)]="premierLeagueYear" (change)="yearChanges()" type="number" min="1992" MAX="2999" required**/>**
5. **</label>**
6. **</mat-card>**
7. **<div** class="sub-wrapper"**>**
8. **<h1>**Played Matches**</h1>**
9. <!--Search Container-->
10. **<mat-card** class="top-header"**>**
11. **<div>**
12. **<div>**
13. **<label>**Filter Date:**</label>**
14. **<input** [(ngModel)]="userInputDate" type="date" placeholder="date"**>**
15. **<button** mat-raised-button (click)="search()"**>**
16. **<mat-icon>**search**</mat-icon>**
17. Search
18. **</button>**
19. **</div>**
20. **</div>**
21. **</mat-card>**
22. <!------Matches Container ----->
23. **<div** id="match-container"**>**
24. **<div** \*ngFor="let matchPlayed of matchesPlayed" class="match"**>**
25. **<div** class="wrapper-r"**>**
26. **<div** class=" main-container-pm matches-one pm-comp image-pm"**>**
27. **<img** src="assets/images/svg/GoalKeeper.svg" alt="goal-keeper-image"**>**
28. **</div>**
29. **<div** class="matches-one pm-comp"**>**
30. **<div** class="pm-match-data"**>**
31. **<h6>**Home**</h6>** **<h6** class="md-detail"**>**{{matchPlayed.homeClub.name}}**</h6>**
32. **<div** class="md-score"**>**
33. **<p>**{{matchPlayed.goalsHomeScored}}**</p>**
34. **</div>**
35. **</div>**
36. **<div** class="pm-match-data"**>**
37. **<h6>**Away**</h6>** **<h6** class="md-detail"**>**{{matchPlayed.awayClub.name}}**</h6>**
38. **<div** class="md-score second-player"**>**
39. **<p>**{{matchPlayed.goalsAwayScored}}**</p>**
40. **</div>**
41. **</div>**
42. **</div>**
43. **<div** class="matches-one pm-score"**>**
44. **<p>**{{matchPlayed.matchDate.month | monthFormat}}  **</p>**
45. **<h6>**{{matchPlayed.matchDate.day}}**</h6>**
46. **</div>**
47. **</div>**
48. **</div>**
49. **</div>**
50. **</div>**
51. **</div>**

#### Played-matches.component.css

1. mat-form-field.mat-form-field {
2. **font-size**: 12px;
3. }
5. .main-wrapper {
6. **padding**: 5%;
7. **background**: #B1D4E0;
8. **height**: 105vh;
9. }
11. mat-card.mat-card.mat-focus-indicator.top-header {
12. **margin-bottom**: 10px;
13. border-radius: 10px;
14. **width**: 97%;
15. **font-size**: 20px;
16. **font-weight**: 500;
17. **margin-left**: 5px;
18. **padding-right**: 8px;
19. **color**: #1a5379;
20. }
22. .matches-one.pm-comp.image-pm img {
23. **width**: 100px;
24. **margin-right**: -20px;
25. **margin-top**: 6px;
26. **margin-bottom**: 2px;
27. }
29. .sub-wrapper {
30. **padding**: 2%;
31. **padding-bottom**: 7%;
32. **background**: #1a5379;
33. border-radius: 15px;
34. }
36. .sub-wrapper h1 {
37. **margin-left**: 5px;
38. }
40. .main-section-border {
41. border-radius: 20px;
42. **width**: 49% !important;
43. **margin-left**: 3px;
44. **background-color**: white;
45. **margin-bottom**: 0% !important;
46. }
48. .matches-one.pm-comp h6 {
49. **font-size**: 18px;
50. **text-transform**: uppercase;
51. **font-weight**: 500;
52. **margin-left**: 15px;
53. }
55. .matches-one.pm-score h2 {
56. **font-size**: 30px;
57. }
59. mat-grid-tile#title1 {
60. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
61. }
63. mat-grid-tile#title2 {
64. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
65. }
67. mat-grid-tile#title3 {
68. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
69. }
71. mat-grid-tile#title4 {
72. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
73. }
75. mat-grid-tile#title5 {
76. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
77. }
79. mat-grid-tile#title6 {
80. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
81. }
83. mat-grid-list.mat-grid-list.rad-main-sect {
84. **padding-bottom**: calc(1 \* ((18% - 0.5px) \* 1) + 0px + 0px) !important;
85. }
87. .mat-grid-tile .mat-figure {
88. justify-**content**: space-evenly !important;
89. **padding-top**: 5px;
90. }
92. .wrapper-r {
93. **display**: flex;
94. justify-**content**: space-evenly;
95. **width**: 100%;
96. **padding-top**: 10px;
97. }
99. .matches-one.pm-score {
100. **margin-top**: 18px;
101. **background**: #94c6f2;
102. border-radius: 21px;
103. **width**: 75px;
104. **height**: 70px;
105. **margin**: auto;
106. **padding**: 10px;
107. **display**: flex;
108. flex-**direction**: column;
109. **padding-bottom**: 15px;
110. **margin-top**: 9px;
111. }
113. .matches-one.pm-score h6 {
114. **font-size**: 40px;
115. **margin**: auto;
116. **margin-top**: -2px;
118. }
120. .matches-one {
121. **margin**: auto;
122. **padding-bottom**: 10px;
123. }
125. button.glbl-btn-style {
126. **float**: right;
127. **padding**: 10px;
128. **font-size**: 20px;
129. **margin**: 10px;
130. **width**: 100px;
131. **border**: none;
132. border-radius: 2px;
133. -webkit-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
134. -moz-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
135. box-shadow: 0px 0px 5px 0px rgba(0, 0, 0, 0.22);
136. }
138. .matches-one.pm-comp p {
139. **font-size**: 15px;
140. **text-transform**: uppercase;
141. **font-family**: 'Roboto';
142. **font-weight**: 500;
143. **text-align**: center;
144. }
146. .matches-one.pm-score p {
147. **margin**: auto;
148. **margin-top**: 1px;
149. }
151. mat-grid-list.mat-grid-list.rad-main-sect.played-match-eryother-col {
152. **margin-top**: -13px;
153. }

156. .pm-match-data {
157. **display**: flex;
158. justify-**content**: space-between;
159. **margin**: auto;
160. **margin-top**: 20px;
161. **width**: 100%;
162. }
164. h6.md-detail {
165. **text-transform**: capitalize !important;
166. }
168. .md-score {
169. **background**: #c6c6c6;
170. **margin**: auto;
171. **padding**: 8px;
172. border-radius: 5px;
173. **margin-top**: -10px;
174. **width**: 7%;
175. }
177. .md-score p {
178. **margin**: 0px !important;
179. }
181. h6.md-detail {
182. **width**: 150px;
183. }
185. mat-card.mat-card.mat-focus-indicator {
186. border-radius: 15px;
187. **margin-bottom**: 10px;
188. }
190. input.ng-pristine.ng-invalid.ng-touched {
191. **background**: #b1d4e0;
192. border-radius: 5px;
193. **width**: 100px;
194. **height**: 30px !important;
195. **border**: none !important;
196. **padding-left**: 5px;
197. **padding-right**: 5px;
198. }
200. input.ng-pristine.ng-invalid {
201. **background**: #b1d4e0;
202. border-radius: 5px;
203. **width**: 100px;
204. **height**: 30px !important;
205. **border**: none !important;
206. **padding-left**: 5px;
207. **padding-right**: 5px;
208. }
210. input.ng-touched.ng-dirty.ng-valid {
211. **background**: #b1d4e0 !important;
212. border-radius: 5px !important;
213. **width**: 100px !important;
214. **height**: 30px !important;
215. **border**: none !important;
216. **padding-left**: 5px;
217. **padding-right**: 5px;
218. }
220. input.ng-dirty.ng-touched.ng-invalid {
222. }
224. button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary {
225. **margin**: 10px;
226. **background**: #b1d4e0;
227. **color**: #1a5379;
228. }
230. .rm-col.pm-comp.image-pm {
231. **margin-left**: -40px;
232. **margin-right**: 65px;
233. }
235. .wrapper-r.rm-VS h1 b {
236. **color**: #acbf65;
237. }
239. mat-card.mat-card.mat-focus-indicator label {
240. **color**: #103957;
241. }
243. .pm-grid.ng-star-inserted:nth-child(n+3) {
244. **margin-top**: 0px !important;
245. }
247. .pm-grid.ng-star-inserted {
248. **margin-top**: 0px !important;
249. }
251. #match-container {
252. **display**: grid;
253. grid-template-columns: 50% 50%;
254. }
256. .match {
257. **background-color**: white;
258. **margin**: 5px;
259. border-radius: 15px;
260. }
262. .match-img {
263. row-span: 3;
264. }
266. .match-date {
267. row-span: 2;
268. }

271. input.ng-pristine.ng-valid.ng-touched {
272. **background**: #b1d4e0 !important;
273. border-radius: 5px !important;
274. **width**: 140px !important;
275. **height**: 30px !important;
276. **border**: none !important;
277. **padding-left**: 5px;
278. **padding-right**: 5px;
279. **margin-left**: 10px;
280. **margin-right**: 10px;
281. }
283. input.ng-pristine.ng-valid {
284. **background**: #b1d4e0 !important;
285. border-radius: 5px !important;
286. **width**: 140px !important;
287. **height**: 30px !important;
288. **border**: none !important;
289. **padding-left**: 5px;
290. **padding-right**: 5px;
291. **margin-left**: 10px;
292. **margin-right**: 10px;
293. }
295. input.ng-pristine.ng-invalid {
296. **background**: #b1d4e0 !important;
297. border-radius: 5px !important;
298. **width**: 140px !important;
299. **height**: 30px !important;
300. **border**: none !important;
301. **padding-left**: 5px;
302. **padding-right**: 5px;
303. **margin-left**: 10px;
304. **margin-right**: 10px;
305. }
307. input.ng-valid.ng-dirty.ng-touched {
308. **background**: #b1d4e0 !important;
309. border-radius: 5px !important;
310. **width**: 140px !important;
311. **height**: 30px !important;
312. **border**: none !important;
313. **padding-left**: 5px;
314. **padding-right**: 5px;
315. **margin-left**: 10px;
316. **margin-right**: 10px;
317. }
319. button.mat-focus-indicator.mat-raised-button.mat-button-base {
320. **margin**: 10px;
321. **background**: #1a5379;
322. **color**: white;
323. }
325. .sub-wrapper h1 {
326. **color**: white;
327. }

#### Played-matches.component.ts

1. **import** {Component, OnInit} from '@angular/core';
2. **import** {PremierLeagueService} from '../premier-league.service';
3. **import** {PremierLeagueYearService} from '../premier-league-year.service';
5. @Component({
6. selector: 'app-played-matches',
7. templateUrl: './played-matches.component.html',
8. styleUrls: ['./played-matches.component.css']
9. })
10. **export** **class** PlayedMatchesComponent **implements** OnInit {
11. userInputDate: string;
12. matchesPlayed = [];
13. matchesPlayedMaster = [];
14. premierLeagueYear;
16. constructor(**private** service: PremierLeagueService, **private** serviceYear: PremierLeagueYearService) {
17. }
19. ngOnInit(): **void** {
20. **this**.serviceYear.premierLeagueYear.subscribe((year) => {
21. **this**.userInputDate = **this**.service.getPlayedMatchesData(year).subscribe((res) => {
22. **this**.matchesPlayed = **this**.matchesPlayedMaster = res;
23. });
24. });
25. }

28. search(): **void** {
29. console.log(**this**.userInputDate);
31. **const** date = **this**.userInputDate.split('-');
33. **this**.matchesPlayed = [];
34. **this**.matchesPlayedMaster.forEach(playedMatch => {
35. **if** (playedMatch.matchDate.year === Number(date[0])
36. && playedMatch.matchDate.month === Number(date[1])
37. && playedMatch.matchDate.day === Number(date[2])
38. ) {
39. **this**.matchesPlayed.push(playedMatch);
40. }
41. });
42. console.log(**this**.matchesPlayed);
43. }
45. yearChanges(): **void** {
46. **this**.serviceYear.premierLeagueYear.next(String(**this**.premierLeagueYear));
47. }
48. }

### Random-match.component

#### Random-match.component.html

1. **<div** class="main-wrapper"**>**
2. **<mat-card>**
3. **<label>**Select year to generate a random match:
4. **<input** [(ngModel)]="premierLeagueYear" (change)="yearChanges()" type="number" min="1992" MAX="2999" required**/>**
5. **</label>**
6. **</mat-card>**
7. **<div** class="sub-wrapper"**>**
8. **<h1>**Randomly generated match**</h1>**
9. **<ng-container** \*ngIf="randomMatch"**>**
10. **<mat-card** class="top-header"**>**
11. **<p>**Date : {{randomMatch.matchDate.month | monthsFormat}} {{randomMatch.matchDate.day}}
12. , {{randomMatch.matchDate.year}}**</p>**
13. **</mat-card>**
14. **<div** class="rm-main-wrapper"**>**
15. **<div** class="rm-row"**>**
16. **<div** class="rm-col"**>**
17. **<div** class="wrapper-r"**>**
18. **<div** class="rm-col rad-name image-rad"**>**
19. **<img** alt="image-club-h" src="assets/images/svg/Referee.svg"**>**
20. **</div>**
21. **<div** class="rm-col rad-name"**>**
22. **<h6>**Selected Home Club**</h6>**
23. **<p>**{{randomMatch.homeClub.name}}**</p>**
24. **</div>**
25. **<div** class="rm-col rad-score"**>**
26. **<p>**Score**</p>**
27. **<h6>**{{randomMatch.goalsHomeScored}}**</h6>**
28. **</div>**
29. **</div>**
30. **</div>**
31. **<div** class="rm-col"**>**
32. **<div** class="wrapper-r rm-VS"**>**
33. **<h1><b>**VS**</b></h1>**
34. **</div>**
35. **</div>**
36. **<div** class="rm-col"**>**
37. **<div** class="wrapper-r"**>**
38. **<div** class="rm-col rad-name image-rad"**>**
39. **<img** alt="image-club-a" src="assets/images/svg/Foul.svg"**>**
40. **</div>**
41. **<div** class="rm-col rad-name"**>**
42. **<h6>**Selected Away Club**</h6>**
43. **<p>**{{randomMatch.awayClub.name}}**</p>**
44. **</div>**
45. **<div** class="rm-col rad-score"**>**
46. **<p>**Score**</p>**
47. **<h6>**{{randomMatch.goalsAwayScored}}**</h6>**
48. **</div>**
49. **</div>**
50. **</div>**
51. **</div>**
52. **</div>**
53. **<div** class="btn-section"**>**
54. **<button** color="primary" mat-raised-button (click)="addMatch()"**>**Add Match**</button>**
55. **<button** color="primary" mat-raised-button (click)="generateRandomMatch()"**>**Generate Random Match**</button>**
56. **</div>**
57. **</ng-container>**
58. **</div>**
59. **</div>**

#### Random-match.component.css

1. .main-wrapper {
2. **padding**: 5%;
3. **background**: #B1D4E0;
4. **height**: 73%;
5. }
7. mat-card.mat-card.mat-focus-indicator.top-header {
8. **margin-bottom**: 10px;
9. border-radius: 10px;
10. **width**: 97%;
11. **font-size**: 20px;
12. **font-weight**: 500;
13. **margin-left**: 5px;
14. **padding-right**: 8px;
15. **color**: #1a5379;
16. }
18. .sub-wrapper h1 {
19. **color**: white !important;
20. }
22. .rm-col.rad-name.image-rad img {
23. **width**: 120px;
24. **margin-right**: -60px;
25. }
27. .sub-wrapper {
28. **padding**: 2%;
29. **padding-bottom**: 7%;
30. **background**: #1a5379;
31. border-radius: 15px;
32. }
34. .sub-wrapper h1 {
35. **margin-left**: 5px;
36. }
38. .main-section-border {
39. border-radius: 20px;
40. **width**: 49% !important;
41. **margin-left**: 3px;
42. }
44. #title1 {
45. **background-color**: white;
46. }
48. #title2 {
49. **background-color**: white;
50. }
52. .rm-col.rad-name h6 {
53. **font-size**: 18px;
54. **text-transform**: uppercase;
55. **font-weight**: 500;
56. }
58. .rm-col.rad-score h2 {
59. **font-size**: 30px;
60. }
62. mat-grid-tile#title1 {
63. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
64. }
66. mat-grid-tile#title2 {
67. **padding-top**: calc((16% - 0.5px) \* 1 + 0px) !important;
68. }
70. mat-grid-list.mat-grid-list.rad-main-sect {
71. **padding-bottom**: calc(1 \* ((18% - 0.5px) \* 1) + 0px + 0px) !important;
72. }
74. .mat-grid-tile .mat-figure {
75. justify-**content**: space-evenly !important;
76. **padding-top**: 5px;
77. }
79. .wrapper-r {
80. **display**: flex;
81. justify-**content**: space-evenly;
82. **width**: 100%;
83. **padding-top**: 10px;
84. }
86. .rm-col.rad-score {
87. **background**: #94c6f2;
88. border-radius: 20px;
89. **width**: 80px;
90. **height**: 80px;
91. **margin**: auto;
92. **padding**: 10px;
93. **display**: flex;
94. flex-**direction**: column;
95. **margin-top**: 6px !important;
96. }
98. .rm-col.rad-score h6 {
99. **font-size**: 40px;
100. **margin**: auto;
101. **margin-top**: -2px;
103. }
105. .rm-row {
106. **display**: flex;
107. **width**: 99%;
108. justify-**content**: space-evenly;
109. **margin**: auto;
110. **height**: 100%;
111. **background**: white;
112. /\* margin: 10px; \*/
113. **margin-left**: 6px;
114. border-radius: 10px;
115. **padding**: 10px 0px 10px 0px;
116. **color**: #1a5379;
117. }
119. .rm-col {
120. **margin**: auto;
121. **margin-left**: 10px !important;
122. }
124. button.glbl-btn-style {
125. **float**: right;
126. **padding**: 10px;
127. **font-size**: 20px;
128. **margin**: 10px;
129. **width**: 100px;
130. **border**: none;
131. border-radius: 2px;
132. -webkit-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
133. -moz-box-shadow: 1px 1px 5px 0px rgba(0, 0, 0, 0.75);
134. box-shadow: 0px 0px 5px 0px rgba(0, 0, 0, 0.22);
135. }
137. .rm-col.rad-name p {
138. **font-size**: 15px;
139. **text-transform**: uppercase;
140. **font-family**: 'Roboto';
141. **font-weight**: 500;
142. }
144. .rm-col.rad-score p {
145. **margin**: auto;
146. **margin-top**: 1px;
147. }
149. mat-card.mat-card.mat-focus-indicator {
150. border-radius: 15px;
151. **margin-bottom**: 10px;
152. }
154. input.ng-pristine.ng-invalid.ng-touched {
155. **background**: #b1d4e0;
156. border-radius: 5px;
157. **width**: 100px;
158. **height**: 20px;
159. **border**: none;
160. }
162. input.ng-pristine.ng-invalid {
163. **background**: #b1d4e0;
164. border-radius: 5px;
165. **width**: 100px;
166. **height**: 20px;
167. **border**: none;
168. }
170. input.ng-touched.ng-dirty.ng-valid {
171. **background**: #b1d4e0 !important;
172. border-radius: 5px !important;
173. **width**: 100px !important;
174. **height**: 20px !important;
175. **border**: 1px solid black !important;
176. }
178. button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary {
179. **margin**: 10px;
180. **background**: #b1d4e0;
181. **color**: #1a5379;
182. }
184. .rm-col.rad-name.image-rad {
185. **margin-left**: -40px;
186. **margin-right**: 65px;
187. }
189. .wrapper-r.rm-VS h1 b {
190. **color**: #acbf65;
191. }
193. mat-card.mat-card.mat-focus-indicator label {
194. **color**: #103957;
195. }

#### Random-match.component.ts

1. **import** {Component, OnInit} from '@angular/core';
2. **import** {PremierLeagueService} from '../premier-league.service';
3. **import** {Match} from '../match';
4. **import** {PremierLeagueYearService} from '../premier-league-year.service';
5. **import** {MatDialog} from '@angular/material/dialog';
6. **import** {DialogAddMatchComponent} from '../dialog-add-match/dialog-add-match.component';
8. @Component({
9. selector: 'app-random-match',
10. templateUrl: './random-match.component.html',
11. styleUrls: ['./random-match.component.css']
12. })
13. **export** **class** RandomMatchComponent **implements** OnInit {
14. randomMatch: Match;
15. premierLeagueYear;
17. constructor(**private** service: PremierLeagueService, **private** serviceYear: PremierLeagueYearService, **public** dialog: MatDialog) {
18. }
20. ngOnInit(): **void** {
21. **this**.generateRandomMatch();
22. }
24. addMatch(): **void** {
25. **this**.serviceYear.premierLeagueYear.subscribe((year) => {
26. **this**.service.postRandomMatchData(year, **this**.randomMatch).subscribe();
27. **this**.dialog.open(DialogAddMatchComponent);
28. });
29. }
31. yearChanges(): **void** {
32. **this**.serviceYear.premierLeagueYear.next(String(**this**.premierLeagueYear));
33. }
35. generateRandomMatch(): **void** {
36. **this**.serviceYear.premierLeagueYear.subscribe((year) => {
37. **this**.service.getRandomMatchData(year).subscribe((res) => {
38. **this**.randomMatch = res;
39. });
40. });
41. }
42. }

### App.component

#### App.component.html

1. **<app-nav-side-bar></app-nav-side-bar>**

#### App.component.ts

1. **import** { Component } from '@angular/core';

4. @Component({
5. selector: 'app-root',
6. templateUrl: './app.component.html',
7. styleUrls: ['./app.component.css']
8. })
9. **export** **class** AppComponent {
10. title = 'Premier League';
11. }

#### App.mmodule.ts

1. **import** {BrowserModule} from '@angular/platform-browser';
2. **import** {NgModule} from '@angular/core';
4. **import** {Routes} from '@angular/router';
5. **import** {AppRoutingModule} from './app-routing.module';
6. **import** {AppComponent} from './app.component';
7. **import** {BrowserAnimationsModule} from '@angular/platform-browser/animations';
8. **import** {NavSideBarComponent} from './nav-side-bar/nav-side-bar.component';
9. **import** {LayoutModule} from '@angular/cdk/layout';
10. **import** {MatToolbarModule} from '@angular/material/toolbar';
11. **import** {MatButtonModule} from '@angular/material/button';
12. **import** {MatSidenavModule} from '@angular/material/sidenav';
13. **import** {MatIconModule} from '@angular/material/icon';
14. **import** {MatListModule} from '@angular/material/list';
15. **import** {LeagueTableComponent} from './league-table/league-table.component';
16. **import** {MatTableModule} from '@angular/material/table';
17. **import** {MatPaginatorModule} from '@angular/material/paginator';
18. **import** {MatSortModule} from '@angular/material/sort';
19. **import** {RandomMatchComponent} from './random-match/random-match.component';
20. **import** {MatCardModule} from '@angular/material/card';
21. **import** {PlayedMatchesComponent} from './played-matches/played-matches.component';
22. **import** {HttpClientModule} from '@angular/common/http';
23. **import** {MatFormFieldModule} from '@angular/material/form-field';
24. **import** {MatSelectModule} from '@angular/material/select';
25. **import** {FormsModule, ReactiveFormsModule} from '@angular/forms';
26. **import** {MatInputModule} from '@angular/material/input';
27. **import** {MatDialogModule} from '@angular/material/dialog';
28. **import** {DialogAddMatchComponent} from './dialog-add-match/dialog-add-match.component';
29. **import** {MonthFormatPipe} from './month-format.pipe';
30. **import** {MonthsFormatPipe} from './months-format.pipe';
31. **import** {DashboardHomeComponent} from './dashboard-home/dashboard-home.component';

34. **const** routes: Routes = [
35. {}
36. ];
38. @NgModule({
39. declarations: [
40. AppComponent,
41. NavSideBarComponent,
42. LeagueTableComponent,
43. RandomMatchComponent,
44. PlayedMatchesComponent,
45. DialogAddMatchComponent,
46. MonthFormatPipe,
47. MonthsFormatPipe,
48. DashboardHomeComponent
49. ],
50. entryComponents: [DialogAddMatchComponent],
51. imports: [
52. BrowserModule,
53. AppRoutingModule,
54. HttpClientModule,
55. BrowserAnimationsModule,
56. LayoutModule,
57. MatToolbarModule,
58. MatButtonModule,
59. MatSidenavModule,
60. MatIconModule,
61. MatListModule,
62. MatTableModule,
63. MatPaginatorModule,
64. MatSortModule,
65. MatCardModule,
66. MatFormFieldModule,
67. MatSelectModule,
68. ReactiveFormsModule,
69. MatInputModule,
70. MatDialogModule,
71. FormsModule,
72. ],
73. providers: [],
74. bootstrap: [AppComponent]
75. })
76. **export** **class** AppModule {
77. }

#### App-routing.module.ts

1. **import** {NgModule} from '@angular/core';
2. **import** {RouterModule, Routes} from '@angular/router';
3. **import** {LeagueTableComponent} from './league-table/league-table.component';
4. **import** {RandomMatchComponent} from './random-match/random-match.component';
5. **import** {PlayedMatchesComponent} from './played-matches/played-matches.component';
6. **import** {DashboardHomeComponent} from './dashboard-home/dashboard-home.component';
8. **const** routes: Routes = [
9. {path: 'home', component: DashboardHomeComponent},
10. {path: 'league-table', component: LeagueTableComponent},
11. {path: 'random-match', component: RandomMatchComponent},
12. {path: 'played-matches', component: PlayedMatchesComponent},
13. {path: '\*\*', redirectTo: 'home'}
14. ];
16. @NgModule({
17. imports: [RouterModule.forRoot(routes)],
18. exports: [RouterModule]
19. })
20. **export** **class** AppRoutingModule {
21. }

#### Footballclub.ts

1. **export** **interface** Footballclub {
2. name: string;
3. location: string;
4. wins: number;
5. defeats: number;
6. matchesPlayed: number;
7. draws: number;
8. goalsScored: number;
9. goalsReceived: number;
10. goalDifference: number;
11. points: number;
13. }

#### Match.ts

1. **import** {Footballclub} from './footballclub';
3. **export** **interface** Match {
4. homeClub: Footballclub;
5. awayClub: Footballclub;
6. goalsHomeScored: number;
7. goalsAwayScored: number;
8. matchDate: {
9. day: number;
10. month: number;
11. year: number;
12. };
13. }

#### Month-format.pipe.ts

1. **import** {Pipe, PipeTransform} from '@angular/core';
3. @Pipe({
4. name: 'monthFormat'
5. })
6. **export** **class** MonthFormatPipe **implements** PipeTransform {
7. transform(monthNum: number): string {
8. **if** (monthNum === 1) {
9. **return** 'JAN';
10. } **else** **if** (monthNum === 2) {
11. **return** 'FEB';
12. } **else** **if** (monthNum === 3) {
13. **return** 'MAR';
14. } **else** **if** (monthNum === 4) {
15. **return** 'APR';
16. } **else** **if** (monthNum === 5) {
17. **return** 'MAY';
18. } **else** **if** (monthNum === 6) {
19. **return** 'JUN';
20. } **else** **if** (monthNum === 7) {
21. **return** 'JUL';
22. } **else** **if** (monthNum === 8) {
23. **return** 'AUG';
24. } **else** **if** (monthNum === 9) {
25. **return** 'SEP';
26. } **else** **if** (monthNum === 10) {
27. **return** 'OCT';
28. } **else** **if** (monthNum === 11) {
29. **return** 'NOV';
30. } **else** {
31. **return** 'DEC';
32. }
33. }
34. }

#### Months-format.pipe.ts

1. **import** {Pipe, PipeTransform} from '@angular/core';
3. @Pipe({
4. name: 'monthsFormat'
5. })
6. **export** **class** MonthsFormatPipe **implements** PipeTransform {
7. transform(monthsNum: number): string {
8. **if** (monthsNum === 1) {
9. **return** 'January';
10. } **else** **if** (monthsNum === 2) {
11. **return** 'February';
12. } **else** **if** (monthsNum === 3) {
13. **return** 'March';
14. } **else** **if** (monthsNum === 4) {
15. **return** 'April';
16. } **else** **if** (monthsNum === 5) {
17. **return** 'May';
18. } **else** **if** (monthsNum === 6) {
19. **return** 'June';
20. } **else** **if** (monthsNum === 7) {
21. **return** 'July';
22. } **else** **if** (monthsNum === 8) {
23. **return** 'August';
24. } **else** **if** (monthsNum === 9) {
25. **return** 'September';
26. } **else** **if** (monthsNum === 10) {
27. **return** 'October';
28. } **else** **if** (monthsNum === 11) {
29. **return** 'November';
30. } **else** {
31. **return** 'December';
32. }
33. }
34. }

#### Premier-league.service.ts

1. **import** {Injectable} from '@angular/core';
2. **import** {HttpClient} from '@angular/common/http';
4. **import** {map} from 'rxjs/operators';
5. **import** {Observable} from 'rxjs/index';
6. **import** {Match} from './match';
8. @Injectable({
9. providedIn: 'root'
10. })
11. **export** **class** PremierLeagueService {
13. **private** randomMatchUrl = 'http://localhost:9000/random-match/';
14. **private** footballClubUrl = 'http://localhost:9000/football-clubs/';
15. **private** footballMatchesUrl = 'http://localhost:9000/football-matches/';
16. **private** footballMatchesSortWinsUrl = 'http://localhost:9000/football-clubs-sort-by-wins/';
17. **private** footballMatchesSortGoalsUrl = 'http://localhost:9000/football-clubs-sort-by-goals/';
19. constructor(**private** http: HttpClient) {
20. }
22. **public** getFootballClubData(year: string): any {
23. **return** **this**.http.get(**this**.footballClubUrl + year).pipe(
24. map((res: ResType) => res.response)
25. );
26. }
28. **public** getRandomMatchData(year: string): any {
29. **return** **this**.http.get(**this**.randomMatchUrl + year).pipe(
30. map((res: ResType) => res.response)
31. );
32. }
34. **public** getPlayedMatchesData(year: string): any {
35. **return** **this**.http.get(**this**.footballMatchesUrl + year).pipe(
36. map((res: ResType) => res.response)
37. );
38. }
40. **public** getFootballClubDataSortedByWins(year: string): any {
41. **return** **this**.http.get(**this**.footballMatchesSortWinsUrl + year).pipe(
42. map((res: ResType) => res.response)
43. );
44. }
46. **public** getFootballClubDataSortedByGoals(year: string): any {
47. **return** **this**.http.get(**this**.footballMatchesSortGoalsUrl + year).pipe(
48. map((res: ResType) => res.response)
49. );
50. }
52. /\*\*
53. \* this should be corrected with correct URL used in PLAY framework
54. \*/
55. **public** postRandomMatchData(year: string, randMatch: Match): Observable<Match> {
56. **return** **this**.http.post<Match>(**this**.randomMatchUrl + year, randMatch);
57. }
58. }
60. **interface** ResType {
61. status: **boolean**;
62. response: any[];
63. }

#### Premier-league-year.service.ts

1. **import** {Injectable} from '@angular/core';
2. **import** {BehaviorSubject} from 'rxjs';
4. @Injectable({
5. providedIn: 'root'
6. })
7. **export** **class** PremierLeagueYearService {
8. premierLeagueYear: BehaviorSubject<string> = **new** BehaviorSubject<string>('');
9. }

### Root

#### Index.html

1. <!doctype html**>**
2. **<html** lang="en"**>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**Premier League Manager**</title>**
6. **<base** href="/"**>**
7. **<meta** name="viewport" content="width=device-width, initial-scale=1"**>**
8. **<link** rel="icon" type="image/x-icon" href="favicon.ico"**>**
9. **<link** href="https://fonts.googleapis.com/css?family=Roboto:300,400,500&display=swap" rel="stylesheet"**>**
10. **<link** href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet"**>**
11. **</head>**
12. **<body** class="mat-typography"**>**
13. **<app-root></app-root>**
14. **</body>**
15. **</html>**

#### Main.ts

1. **import** { enableProdMode } from '@angular/core';
2. **import** { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
4. **import** { AppModule } from './app/app.module';
5. **import** { environment } from './environments/environment';
7. **if** (environment.production) {
8. enableProdMode();
9. }
11. platformBrowserDynamic().bootstrapModule(AppModule)
12. .**catch**(err => console.error(err));

#### Style.css

1. html, body { **height**: 100%; }
2. body { **margin**: 0; **font-family**: Roboto, "Helvetica Neue", sans-serif; }

# Testing

## JUnit Source Code

### Entities

#### PremierLeagueManagerTest.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** entities;
10. **import** org.junit.jupiter.api.Test;
12. **import** java.io.IOException;
14. **import** **static** org.junit.jupiter.api.Assertions.\*;
16. /\*\*
17. \* This is Test class of the PremierLeagueManager Class
18. \*
19. \* @author Radhika Ranasinghe
20. \* @version 1.0
21. \* @since 2020-12-27
22. \*/
23. **class** PremierLeagueManagerTest {
25. /\*\*
26. \* Test method to add a club in a given year
27. \*/
28. @Test
29. **void** addClub() {
30. // Clearing the clubs in the premier league
31. PremierLeagueManager.getInstance().getClubsInLeague().clear();
33. // Valid Test Case for adding a club
34. FootballClub clubLiverpool = **new** FootballClub("Liverpool", "Liverpool");
35. PremierLeagueManager.getInstance().addClub(clubLiverpool); // Actual size
36. assertEquals(1, PremierLeagueManager.getInstance().getClubsInLeague().size());
38. // Valid Test Case for adding a club
39. FootballClub evertonClub = **new** FootballClub("Everton", "Liverpool");
40. PremierLeagueManager.getInstance().addClub(evertonClub);
41. assertEquals(2, PremierLeagueManager.getInstance().getClubsInLeague().size());
43. //Adding 20 clubs to the league
44. **for** (**int** i = 0; i < 20; i++) {
45. PremierLeagueManager.getInstance().addClub(**new** FootballClub("club " + i, "location " + i));
46. }
47. assertEquals(20, PremierLeagueManager.getInstance().getClubsInLeague().size());
49. }
51. /\*\*
52. \* Test method to delete a club in given year
53. \*/
54. @Test
55. **void** deleteClub() {
56. // Clearing the clubs in the premier league
57. PremierLeagueManager.getInstance().getClubsInLeague().clear();
59. // Entering a correct name to delete a club
60. FootballClub evertonClub = **new** FootballClub("Everton", "Liverpool");
61. PremierLeagueManager.getInstance().addClub(evertonClub);
62. PremierLeagueManager.getInstance().deleteClub("Everton");
63. assertEquals(0, PremierLeagueManager.getInstance().getClubsInLeague().size());
65. // Entering a wrong name to delete a delete a club
66. FootballClub liverpoolClub = **new** FootballClub("Liverpool", "Liverpool");
67. PremierLeagueManager.getInstance().addClub(liverpoolClub);
68. PremierLeagueManager.getInstance().deleteClub("liver");
69. assertEquals(1, PremierLeagueManager.getInstance().getClubsInLeague().size());
71. }
73. /\*\*
74. \* Test method to get statistics of a club
75. \*/
76. @Test
77. **void** getStatsOfAClub() {
78. // Clearing the clubs in the premier league
79. PremierLeagueManager.getInstance().getClubsInLeague().clear();
81. // Getting a statistics of a valid club
82. FootballClub club = **new** FootballClub("Test a", "test a", 1, 2, 3, 4, 5, 6, 7, 8);
83. PremierLeagueManager.getInstance().addClub(club);
84. String output = "\n" +
85. "Statistics of the club Test a (test a)\n" +
86. "\t> Number of Wins\t: 1\n" +
87. "\t> Number of Draws\t: 3\n" +
88. "\t> Number of Defeats\t: 2\n" +
89. "\t> Goals Scored\t\t: 4\n" +
90. "\t> Goals Against\t\t: 5\n" +
91. "\t> Goal Difference\t: -1\n" +
92. "\t> Matched Played\t: 8\n" +
93. "\t> Current Points\t: 7";
94. assertEquals(output, PremierLeagueManager.getInstance().getStatsOfAClub(club));
95. }
97. /\*\*
98. \* Test method to add a match in a given year
99. \*/
100. @Test
101. **void** addMatch() {
102. // Clearing the clubs in the premier league
103. PremierLeagueManager.getInstance().getClubsInLeague().clear();
105. // Entering a valid match
106. FootballClub clubA = **new** FootballClub("club A", "location A"); // Home Club
107. FootballClub clubB = **new** FootballClub("club B", "location B"); // Away Club
108. PremierLeagueManager.getInstance().addClub(clubA);
109. PremierLeagueManager.getInstance().addClub(clubB);
110. Match<FootballClub> match = **new** Match<FootballClub>(clubA, clubB, 5, 4, **new** Date(4, 5, 2020));
111. PremierLeagueManager.getInstance().addMatch(match);
112. assertEquals(1, PremierLeagueManager.getInstance().getMatchesInLeague().size());
113. // Check if the respective clubs are updated accordingly
114. assertAll(
115. () -> assertEquals(1, clubA.getWins()),
116. () -> assertEquals(1, clubB.getDefeats()),
117. () -> assertEquals(3, clubA.getPoints()),
118. () -> assertEquals(0, clubB.getPoints()),
119. () -> assertEquals(1, clubA.getMatchesPlayed()),
120. () -> assertEquals(1, clubB.getMatchesPlayed()),
121. () -> assertEquals(5, clubA.getGoalsScored()),
122. () -> assertEquals(4, clubB.getGoalsScored()),
123. () -> assertEquals(4, clubA.getGoalsReceived()),
124. () -> assertEquals(5, clubB.getGoalsReceived())
125. );
127. // Entering the same match but doesn't get added to the match array list
128. PremierLeagueManager.getInstance().addMatch(match);
129. assertEquals(1, PremierLeagueManager.getInstance().getMatchesInLeague().size());
131. // Entering a draw match
132. Match<FootballClub> match2 = **new** Match<FootballClub>(clubA, clubB, 3, 3, **new** Date(30, 10, 2020));
133. PremierLeagueManager.getInstance().addMatch(match2);
135. // Check if the respective clubs are updated accordingly
136. assertAll(
137. () -> assertEquals(1, clubA.getDraws()),
138. () -> assertEquals(1, clubB.getDraws()),
139. () -> assertEquals(4, clubA.getPoints()),
140. () -> assertEquals(1, clubB.getPoints()),
141. () -> assertEquals(2, clubA.getMatchesPlayed()),
142. () -> assertEquals(2, clubB.getMatchesPlayed()),
143. () -> assertEquals(8, clubA.getGoalsScored()),
144. () -> assertEquals(7, clubB.getGoalsScored()),
145. () -> assertEquals(7, clubA.getGoalsReceived()),
146. () -> assertEquals(8, clubB.getGoalsReceived())
147. );
148. }
150. /\*\*
151. \* Test method to get statistics of all clubs
152. \*/
153. @Test
154. **void** getStatsOfAllClubs() {
155. // Clearing the clubs in the premier league
156. PremierLeagueManager.getInstance().getClubsInLeague().clear();
158. FootballClub clubA = **new** FootballClub("club A", "location A"); // Home Club
159. FootballClub clubB = **new** FootballClub("club B", "location B"); // Away Club
160. PremierLeagueManager.getInstance().addClub(clubA);
161. PremierLeagueManager.getInstance().addClub(clubB);
162. Match<FootballClub> match = **new** Match<FootballClub>(clubA, clubB, 4, 5, **new** Date(4, 5, 2020));
163. PremierLeagueManager.getInstance().addMatch(match);
164. String output = "\n" +
165. "\t\tT H E   P R E M I E R   L E A G U E   T A B L E\n" +
166. "\n" +
167. ">\tWins are shown with '+' \n" +
168. ">\tLosses are shown with '-' \n" +
169. ">\tDraws are shown with '\*'\n" +
170. ">\tMatch data not found is shown with '/' \n" +
171. "\n" +
172. "+----------+-----------+---------------+----------------+-----+------+-------+----+----+----+--------+-----------------+\n" +
173. "| Position | Club Name | Club Location | Played Matches | Won | Loss | Drawn | GF | GA | GD | Points | Last 5 Matches  |\n" +
174. "+----------+-----------+---------------+----------------+-----+------+-------+----+----+----+--------+-----------------+\n" +
175. "| 1        | club B    | location B    | 1              | 1   | 0    | 0     | 5  | 4  | 1  | 3      |  +  /  /  /  /  |\n" +
176. "| 2        | club A    | location A    | 1              | 0   | 1    | 0     | 4  | 5  | -1 | 0      |  -  /  /  /  /  |\n" +
177. "+----------+-----------+---------------+----------------+-----+------+-------+----+----+----+--------+-----------------+\n";
179. assertEquals(output, PremierLeagueManager.getInstance().getStatsOfAllClubs());
181. }
183. /\*\*
184. \* Test method to unit test four methods of storing data
185. \* saveClubData()
186. \* saveMatchData()
187. \* retrieveClubData()
188. \* retrieveMatchData()
189. \*/
190. @Test
191. **void** databaseTest() {
192. // Clearing the clubs in the premier league
193. PremierLeagueManager.getInstance().getClubsInLeague().clear();
194. PremierLeagueManager.getInstance().getMatchesInLeague().clear();
196. // Adding two clubs and a match to the Premier league
197. FootballClub clubA = **new** FootballClub("club A", "location A"); // Home Club
198. FootballClub clubB = **new** FootballClub("club B", "location B"); // Away Club
199. PremierLeagueManager.getInstance().addClub(clubA);
200. PremierLeagueManager.getInstance().addClub(clubB);
202. Match<FootballClub> matchA = **new** Match<FootballClub>(clubA, clubB, 3, 8, **new** Date(4, 5, 2020));
203. PremierLeagueManager.getInstance().addMatch(matchA);
205. // Saving the clubs and matches in the premier league
206. **try** {
207. PremierLeagueManager.getInstance().saveClubData("clubDataFile" + 500 + ".txt");
208. PremierLeagueManager.getInstance().saveMatchData("matchDataFile" + 500 + ".txt");
209. } **catch** (IOException e) {
210. e.printStackTrace();
211. }
212. // Clearing the clubs in the premier league
213. PremierLeagueManager.getInstance().getClubsInLeague().clear();
214. PremierLeagueManager.getInstance().getMatchesInLeague().clear();
216. // Loading the clubs and matches in the premier league
217. **try** {
218. PremierLeagueManager.getInstance().retrieveClubData("clubDataFile" + 500 + ".txt");
219. PremierLeagueManager.getInstance().retrieveMatchData("matchDataFile" + 500 + ".txt");
221. } **catch** (IOException | ClassNotFoundException e) {
222. e.printStackTrace();
224. }
225. // If the saving and loading works correctly, clubA should be load back to index 0
226. assertEquals(clubA, PremierLeagueManager.getInstance().getClubsInLeague().get(0));
228. // If the saving and loading works correctly, clubB should be load back to index 1
229. assertEquals(clubB, PremierLeagueManager.getInstance().getClubsInLeague().get(1));
231. // If the saving and loading works correctly, size should be equal to 2
232. assertEquals(2, PremierLeagueManager.getInstance().getClubsInLeague().size());
234. // If the saving and loading works correctly, match should be load back to index 0
235. assertEquals(matchA, PremierLeagueManager.getInstance().getMatchesInLeague().get(0));
237. // If the saving and loading works correctly, size should be equal to 1
238. assertEquals(1, PremierLeagueManager.getInstance().getMatchesInLeague().size());

241. }
243. /\*\*
244. \* Method to test the randomly generated match
245. \*/
246. @Test
247. **void** generateRandomMatch() {
248. // Clearing the clubs in the premier league
249. PremierLeagueManager.getInstance().getClubsInLeague().clear();
251. //When no clubs are there in the premier league, a match is not generated
252. Match<FootballClub> matchRand1 = PremierLeagueManager.getInstance().generateRandomMatch(2020);
253. assertNull(matchRand1);
255. //If only one club is there,a match is not generated
256. FootballClub clubA = **new** FootballClub("club A", "location A");
257. PremierLeagueManager.getInstance().addClub(clubA);
258. Match<FootballClub> matchRand2 = PremierLeagueManager.getInstance().generateRandomMatch(2020);
259. assertNull(matchRand2);
261. //there should be more than 2 clubs in the premier league to generate a random match
262. FootballClub clubB = **new** FootballClub("club B", "location B");
264. // the return type of the method is proved as follows
265. PremierLeagueManager.getInstance().addClub(clubB);
266. Match<FootballClub> matchRand3 = PremierLeagueManager.getInstance().generateRandomMatch(2020);
267. assertEquals(Match.**class**, matchRand3.getClass());
269. }
271. }

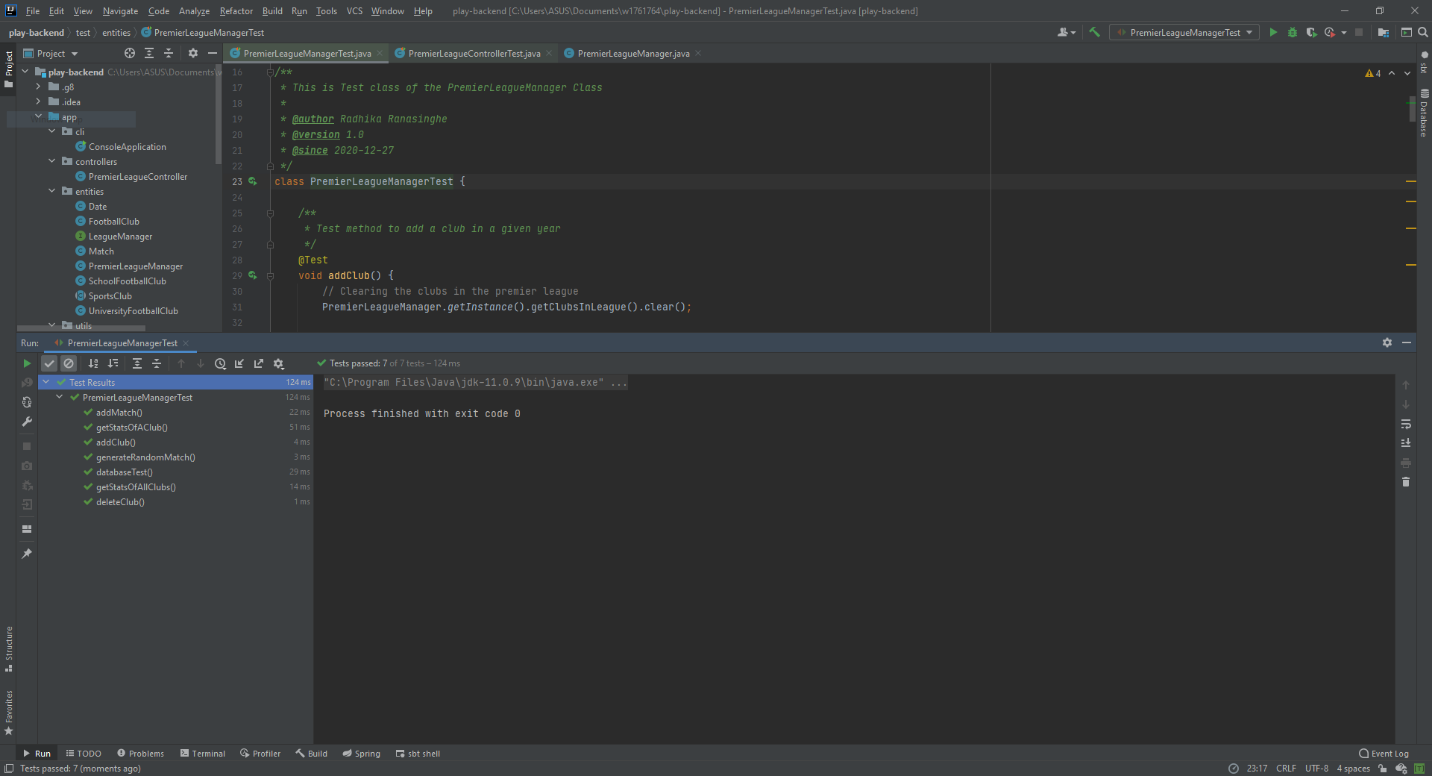


Figure 4: Test results of the PremierLeagueManagerTest.java

### Controllers

#### PremierLeagueControllerTest.java

1. /\*
2. \* Name     :   Radhika Ranasinghe
3. \* UoW ID   :   w1761764
4. \* "I confirm that I understand what plagiarism / collusion / contract cheating is and have read and understood the
5. \* section on Assessment Offences in the Essential Information for Students. The work that I have submitted is entirely
6. \* my own. Any work from other authors is duly referenced and acknowledged."
7. \*/
8. **package** controllers;
10. **import** akka.util.ByteString;
11. **import** com.fasterxml.jackson.core.JsonProcessingException;
12. **import** com.fasterxml.jackson.databind.JsonNode;
13. **import** com.fasterxml.jackson.databind.ObjectMapper;
14. **import** entities.\*;
15. **import** org.junit.jupiter.api.BeforeEach;
16. **import** org.junit.jupiter.api.Test;
17. **import** play.Application;
18. **import** play.http.HttpEntity;
19. **import** play.inject.guice.GuiceApplicationBuilder;
20. **import** play.libs.Json;
21. **import** play.mvc.Http;
22. **import** play.mvc.Result;
23. **import** play.test.Helpers;
25. **import** java.io.IOException;
26. **import** java.util.ArrayList;
28. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
29. **import** **static** play.mvc.Http.Status.CREATED;
30. **import** **static** play.mvc.Http.Status.OK;
31. **import** **static** play.test.Helpers.POST;
32. **import** **static** play.test.Helpers.route;
34. /\*\*
35. \* This is Test class of the PremierLeagueController Class
36. \*
37. \* @author Radhika Ranasinghe
38. \* @version 1.0
39. \* @since 2020-12-27
40. \*/
41. **class** PremierLeagueControllerTest {
42. // Declaring a Test year
43. **private** **final** **int** year = 200;
45. // Declaring four test clubs
46. **private** FootballClub clubA;
47. **private** FootballClub clubB;
48. **private** FootballClub clubC;
49. **private** FootballClub clubD;
51. // Declaring three test matches
52. **private** Match<FootballClub> matchA;
53. **private** Match<FootballClub> matchB;
54. **private** Match<FootballClub> matchC;
56. /\*\*
57. \* Initialize test fixtures before each test method
58. \*/
59. @BeforeEach
60. **void** creation() {
61. // Declaring PremierLeagueManager called 'premierLeagueManager'
62. PremierLeagueManager premierLeagueManager = PremierLeagueManager.getInstance();
64. //Clearing the club array and match array
65. premierLeagueManager.getClubsInLeague().clear();
66. premierLeagueManager.getMatchesInLeague().clear();
68. // Adding test clubs to the premier league
69. clubA = **new** FootballClub("club A", "location A");
70. clubB = **new** FootballClub("club B", "location B");
71. clubC = **new** FootballClub("club C", "location C");
72. clubD = **new** FootballClub("club D", "location D");
73. PremierLeagueManager.getInstance().addClub(clubA);
74. PremierLeagueManager.getInstance().addClub(clubB);
75. PremierLeagueManager.getInstance().addClub(clubC);
76. PremierLeagueManager.getInstance().addClub(clubD);
78. // Adding test matches to the premier league
79. matchA = **new** Match<FootballClub>(clubA, clubB, 5, 4, **new** Date(30, 10, year));
80. PremierLeagueManager.getInstance().addMatch(matchA);
82. matchB = **new** Match<FootballClub>(clubC, clubD, 6, 3, **new** Date(27, 9, year));
83. PremierLeagueManager.getInstance().addMatch(matchB);
85. matchC = **new** Match<FootballClub>(clubB, clubC, 2, 2, **new** Date(5, 5, year));
86. PremierLeagueManager.getInstance().addMatch(matchC);
88. // Saving a test file to the premier league
89. **try** {
90. premierLeagueManager.saveClubData("clubDataFile" + year + ".txt");
91. premierLeagueManager.saveMatchData("matchDataFile" + year + ".txt");
92. } **catch** (IOException ignored) {
93. }
94. }
96. /\*\*
97. \* Test method to retrieve random Match on the given year
98. \*/
99. @Test
100. **void** retrieveMatch() {
101. Result result = **new** PremierLeagueController().retrieveMatch(year);
102. assertEquals(OK, result.status());
103. }

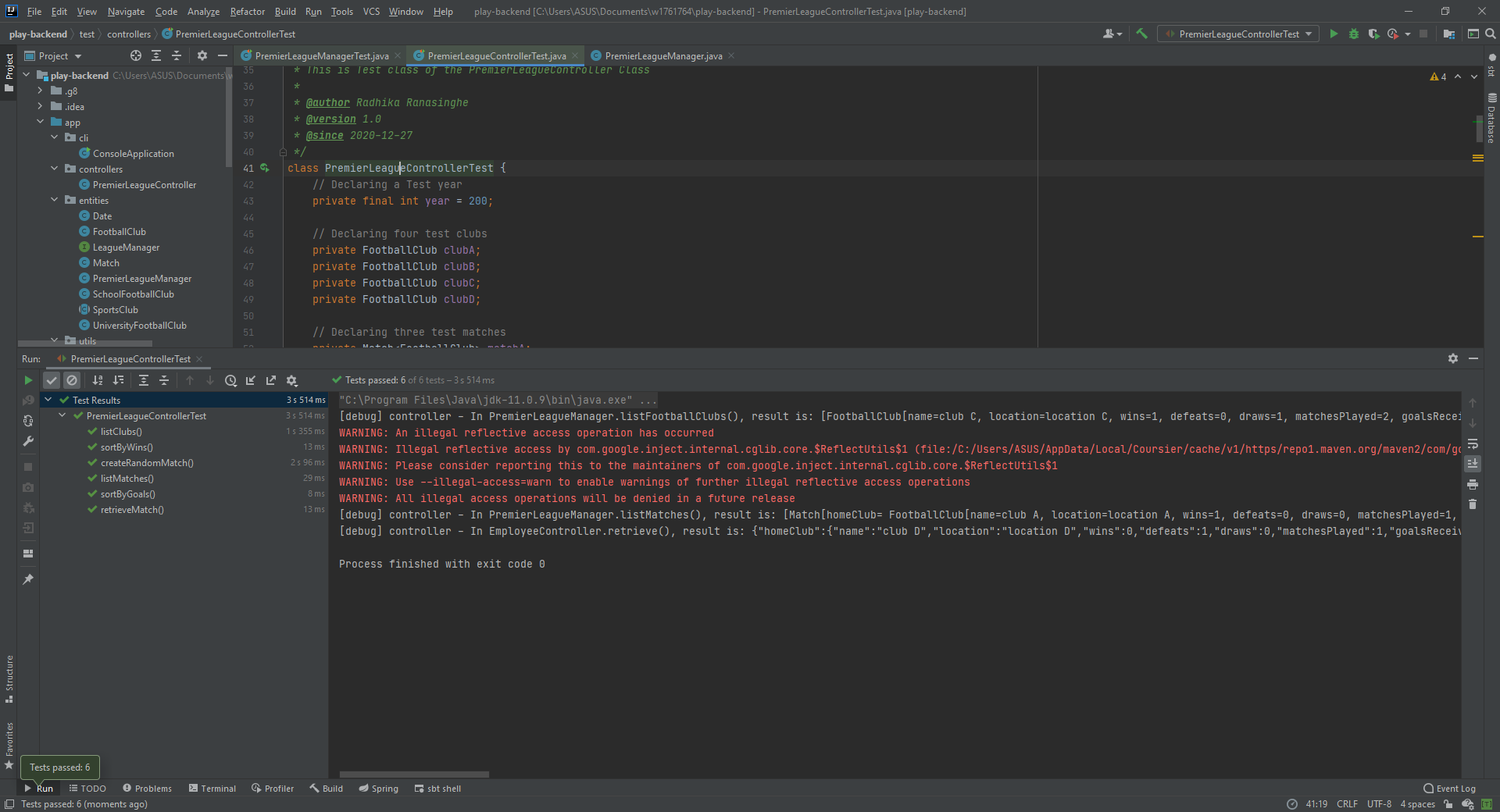
106. /\*\*
107. \* Test method to retrieve all the clubs in the premier league in the given year
108. \*/
109. @Test
110. **void** listClubs() {
111. **try** {
112. // Result output given out by the listClubs method
113. Result result = **new** PremierLeagueController().listClubs(year);
115. // Taking the body of the result which is a ByteString
116. ByteString responseBody = ((HttpEntity.Strict) result.body()).data();
117. // Decoding the body of the result to string
118. String res = responseBody.decodeString("UTF-8");
120. // Declaring an Object Mapper
121. ObjectMapper mapper = **new** ObjectMapper();
122. JsonNode actualObj = mapper.readTree(res);
124. // Taking only the field response out of the Json
125. JsonNode body = actualObj.get("response");
126. // Adding each Football club to a List
127. ArrayList<SportsClub> list = **new** ArrayList<>();
128. **for** (**int** i = 0; i < body.size(); i++) {
129. FootballClub footballClub = Json.fromJson(body.get(i), FootballClub.**class**);
130. list.add(footballClub);
131. }
133. // After sorting by points, the expected outcomes are as follows
134. assertEquals(clubC, list.get(0));
135. assertEquals(clubA, list.get(1));
136. assertEquals(clubB, list.get(2));
137. assertEquals(clubD, list.get(3));
139. // Checking match status
140. assertEquals(OK, result.status());
142. } **catch** (JsonProcessingException e) {
143. e.printStackTrace();
144. }
146. }
148. /\*\*
149. \* Test method to retrieve all the matches in the premier league in the given year
150. \*/
151. @Test
152. **void** listMatches() {
153. **try** {
154. // Result output given out by the listMatches method
155. Result result = **new** PremierLeagueController().listMatches(year);
157. // Taking the body of the result which is a ByteString
158. ByteString responseBody = ((HttpEntity.Strict) result.body()).data();
159. // Decoding the body of the result to string
160. String res = responseBody.decodeString("UTF-8");
162. // Declaring an Object Mapper
163. ObjectMapper mapper = **new** ObjectMapper();
164. JsonNode actualObj = mapper.readTree(res);
166. // Taking only the field response out of the Json
167. JsonNode body = actualObj.get("response");
169. ArrayList<Match<FootballClub>> list = **new** ArrayList<>();
171. **for** (**int** i = 0; i < body.size(); i++) {
172. Match<FootballClub> match = Json.fromJson(body.get(i), Match.**class**);
173. list.add(match);
174. }
176. // The added matches are listed as follows
177. assertEquals(matchA, list.get(0));
178. assertEquals(matchB, list.get(1));
179. assertEquals(matchC, list.get(2));
181. // Checking match status
182. assertEquals(OK, result.status());
183. } **catch** (JsonProcessingException e) {
184. e.printStackTrace();
185. }
187. }
189. /\*\*
190. \* Test method to retrieve all the clubs sorted according goals in the given year
191. \*/
192. @Test
193. **void** sortByGoals() {
194. **try** {
195. // Result output given out by the sortByGoals method
196. Result result = **new** PremierLeagueController().sortByGoals(year);
198. // Taking the body of the result which is a ByteString
199. ByteString responseBody = ((HttpEntity.Strict) result.body()).data();
200. // Decoding the body of the result to string
201. String res = responseBody.decodeString("UTF-8");
203. // Declaring an Object Mapper
204. ObjectMapper mapper = **new** ObjectMapper();
205. JsonNode actualObj = mapper.readTree(res);
207. // Taking only the field response out of the Json
208. JsonNode body = actualObj.get("response");
209. // Adding each Football club to a List
210. ArrayList<SportsClub> list = **new** ArrayList<>();
211. **for** (**int** i = 0; i < body.size(); i++) {
212. FootballClub footballClub = Json.fromJson(body.get(i), FootballClub.**class**);
213. list.add(footballClub);
214. }
216. // After sorting by goals, the expected outcomes are as follows
217. assertEquals(clubC, list.get(0));
218. assertEquals(clubB, list.get(1));
219. assertEquals(clubA, list.get(2));
220. assertEquals(clubD, list.get(3));
222. // Checking match status
223. assertEquals(OK, result.status());
225. } **catch** (JsonProcessingException e) {
226. e.printStackTrace();
227. }
228. }
230. /\*\*
231. \* Test method to retrieve all the clubs sorted according wins in the given year
232. \*/
233. @Test
234. **void** sortByWins() {
235. **try** {
236. // Result output given out by the sortByWins method
237. Result result = **new** PremierLeagueController().sortByWins(year);
239. // Taking the body of the result which is a ByteString
240. ByteString responseBody = ((HttpEntity.Strict) result.body()).data();
241. // Decoding the body of the result to string
242. String res = responseBody.decodeString("UTF-8");
244. // Declaring an Object Mapper
245. ObjectMapper mapper = **new** ObjectMapper();
246. JsonNode actualObj = mapper.readTree(res);
248. // Taking only the field response out of the Json
249. JsonNode body = actualObj.get("response");
250. // Adding each Football club to a List
251. ArrayList<SportsClub> list = **new** ArrayList<>();
252. **for** (**int** i = 0; i < body.size(); i++) {
253. FootballClub footballClub = Json.fromJson(body.get(i), FootballClub.**class**);
254. list.add(footballClub);
255. }
257. // After sorting by wins, the expected outcomes are as follows
258. assertEquals(clubA, list.get(0));
259. assertEquals(clubC, list.get(1));
260. assertEquals(clubB, list.get(2));
261. assertEquals(clubD, list.get(3));
263. // Checking match status
264. assertEquals(OK, result.status());
266. } **catch** (JsonProcessingException e) {
267. e.printStackTrace();
268. }
269. }
271. /\*\*
272. \* Test method to create a a POST request with the random match in the given year
273. \*/
274. @Test
275. **void** createRandomMatch() {
276. // Using Guice for dependency injection
277. Application application = **new** GuiceApplicationBuilder().build();
278. // Starts the test
279. Helpers.start(application);
280. // Generating a random match to be added
281. Match<FootballClub> randMatch = PremierLeagueManager.getInstance().generateRandomMatch(year);
282. //Converting the match to a JsonNode
283. JsonNode jsonNode = Json.toJson(randMatch);
285. // Creating a request to send the JsonNode as a Post request
286. Http.RequestBuilder request = **new** Http.RequestBuilder()
287. .bodyJson(jsonNode)
288. .method(POST)
289. .uri(routes.PremierLeagueController.createRandomMatch(year).url());
290. Result result = route(application, request);
291. assertEquals(CREATED, result.status());
292. }
293. }

Figure 5: Results of the PremierLeagueControllerTest.java

# Assumptions

* Premier League Manager can only have a single instance of it.
* Premier League Season Year can only be a year after 1992.
* Premier League club name can only have a name with only alphabetical characters and spaces.
* Premier League club location can only have a name with only alphabetical characters and spaces.
* Premier League can only register up to 20 clubs only.
* When a club is added to the Premier League Manager it is added to data storing files immediately as well.
* When a club is deleted from the Premier League Manager it is updated right after the deletion.
* GUI can be opened from CLI itself by giving the selection.
* Random match generation makes a club take a score of only up to 15.

# Conclusion

The source code and the report were successfully completed with a fully functioning Command Line Interface and Graphical User Interface taking many assumptions into consideration.

# References

Beach, J., 2021. *Planetb | Syntax Highlight Code In Word Documents*. [online] Planetb.ca. Available at: <http://www.planetb.ca/syntax-highlight-word> [Accessed 4 January 2021].

Staruml.io. 2021. *Staruml*. [online] Available at: <https://staruml.io/> [Accessed 4 January 2021].