CS4222/CS5052 Project Submission Documentation

# Group Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Project Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Please use this checklist to ensure that you have all the items for your submission. You should deliver these items in accordance with the Delivery Instructions outlined on this page.

* **Signed percentages of the work done by each student (equal work = 25% each, assuming 4 persons in a group) (cf. Page 3 of this document)**
* **Completed Project Methods/Person Contribution Table (cf. Page 4 of this document)**
* **Printed Documented Source Code (printed in landscape mode)**
* **Completed Project Group Meetings Deliverable (cf. Page 2 of this document)**

**Delivery Instructions**

* **Email Documented Source Code to annette.mcelligott@ul.ie**

The group's documented source code, data files, Java documentation and so forth should be emailed to me by the project leader, by the deadline for this project. Each of the other group members should be copied on this email. The subject of this email should be CS4222/CS5052 Project X Group Y where X is the Project Number and Y is the Group's Number.

* **By the deadline, the Project Leader should put the following items in a see-through plastic pocket organiser and then place them in the relevant box underneath the stairs in the CS Building.**
  + Printed Documented Source Code (printed in landscape mode)
  + Signed percentages of the work done by each student (cf. Page 3 of this document)
  + Completed Project Methods/Person Contribution Table (also signed by each person in the group) (cf. Page 4 of this document)
  + Completed Project Group Meetings Deliverable (cf. Page 2 of this document)

**CS4222/CS5052: Project Group Meetings Deliverable**

# Group Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Project Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**You should also use this document to detail any problems encountered by the group.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Meeting Number** | **In Attendance** | **Assumptions Made and Decisions Taken/Agreed** |
| **8/3/18** | **1** | **Donal, Eoin, Victor** | **Outline of methods decided and divided** |
| **12/3/18** | **2** | **Donal, Eoin, Victor** | **Basic code implemented into menu** |
| **20/3/18** | **3** | **Donal, Victor** | **Create a league completed/Edited Login** |
| **21/3/18** | **4** | **Donal, Eoin, Victor** | **Brainstormed ideas fixtures and results** |
| **22/3/18** | **5** | **Donal, Eoin, Victor** | **Final tasks divided and bug testing.** |

|  |  |
| --- | --- |
| **Student ID** | **Student Name (Signature)** |
| **17206413** |  |
| **17238838** |  |
| **17241499** |  |

**CS4222/CS5052: Signed percentages of the work done by each student**

# Group Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Project Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Student Name and Student Signature** | **Percentage of marks** |
| **17238838** |  | **34** |
| **17206413** |  | **33** |
| **17241499** |  | **33** |

**CS4222/CS5052: Project Methods/Person Contribution**

**Group Number \_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**TM1:** *Victor*

**TM2:** *Donal*

**TM3:** *Eoin*

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **TM1** | **TM2** | **TM3** |
| Main | 5 | 5 | 90 |
| checkAmountOfLeagues | 100 | 0 | 0 |
| createLeague | 100 | 0 | 0 |
| generateFixtures | 100 | 0 | 0 |
| getNumberOfTeams | 100 | 0 | 0 |
| viewLeaderboard | 33 | 34 | 33 |
| readFilesIntoArrayLists | 25 | 25 | 50 |
| createEmptyLeaderboard | 33 | 33 | 34 |
| processResults | 34 | 33 | 33 |
| recordFixtureResultForHomeTeam | 33 | 34 | 33 |
| recordFixtureResultForAwayTeam | 33 | 33 | 34 |
| orderLeaderboard | 34 | 33 | 33 |
| displayLeaderboard | 33 | 34 | 33 |
| viewLeagues | 0 | 100 | 0 |
| viewTeams | 0 | 100 | 0 |
| viewFixtures | 0 | 20 | 80 |
| addResults | 0 | 20 | 80 |
| verifyLogin | 0 | 100 | 0 |

**Signatures of Group Members**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **import java.util.\*;**
2. **import java.io.\*;**
3. **import javax.swing.\*;**
4. **public class Leagues{**
5. **/\*\***
6. **\* Global variables that store String arrays and Array Lists,**
7. **\* as well as stores the Admin number of the currently logged in**
8. **\* user and tells other methods which leagues to show.**
9. **\*/**
10. **private static final String[] mainMenu = { "Create League", "View and Edit", "Log Out" };**
11. **private static final String[] subMenu = {"View Leagues", "View Teams", "View Leaderboard", "View Fixtures", "Add Results", "Back to Main Menu" };**
12. **private static Scanner x;**
13. **private static Scanner y;**
14. **private static Scanner z;**
15. **private static Scanner findAdminNum;**
16. **private static int currentAdminNum;**
17. **private static File accountInfo = new File ("userInfo.txt");**
18. **private static File leagueInfo = new File ("leagueInfo.txt");**
19. **public static ArrayList<ArrayList<String>> teams;**
20. **public static ArrayList<ArrayList<Integer>> fixtures;**
21. **public static ArrayList<ArrayList<Integer>> results;**
22. **public static int [][] leaderBoard;**
23. **/\*\***
24. **\* The main is used as a menu for selecting the methods**
25. **\* While boolean main is true the user will always be returned to the main menu**
26. **\* after finishing one action.**
27. **\* As is the same for boolean sub and the corresponding sub menu.**
28. **\*/**
29. **public static void main(String[] args) throws IOException {**
30. **verifyLogin();**
31. **boolean main = true;**
32. **while (main) {**
33. **boolean sub = true;**
34. **String section = (String) JOptionPane.showInputDialog(null, "Menu","",JOptionPane.QUESTION\_MESSAGE, null, mainMenu, mainMenu[0]);**
35. **if(section=="Create League") {**
36. **createLeague();**
37. **}else if(section=="View and Edit") {**
38. **while (sub) {**
39. **String subSection = (String) JOptionPane.showInputDialog(null, "Menu","",JOptionPane.QUESTION\_MESSAGE, null, subMenu, subMenu[0]);**
40. **if(subSection=="View Leaderboard"){**
41. **viewLeaderboard();**
42. **}else if(subSection=="View Fixtures"){**
43. **viewFixtures();**
44. **}else if(subSection=="Add Results") {**
45. **addResults();**
46. **}else if(subSection =="View Teams") {**
47. **viewTeams();**
48. **}else if(subSection =="View Leagues") {**
49. **viewLeagues();**
50. **}**
51. **else {**
52. **sub = false;**
53. **}**
54. **}**
55. **}else**
56. **main = false;**
57. **}**
58. **JOptionPane.showMessageDialog(null, "You have been logged out");**
59. **System.exit(0);**
60. **}**
61. **/\*\***
62. **\* Scans "leagueInfo.txt" for the latest league number,**
63. **\* to be used by some other methods such as createLeague**
64. **\*/**
65. **public static int checkAmmountOfLeagues() throws IOException {**
66. **int leagueCounter = 0;**
67. **String lineFromFile;**
68. **String fileElements[];**
69. **Scanner leagueChecker = new Scanner(leagueInfo);**
70. **while (leagueChecker.hasNext())**
71. **{**
72. **lineFromFile = leagueChecker.nextLine();**
73. **fileElements = lineFromFile.split(",");**
74. **leagueCounter =Integer.parseInt(fileElements[0]);**
75. **}**
76. **leagueChecker.close();**
77. **return leagueCounter;**
78. **}**
79. **/\*\***
80. **\* During creation the new league number is assigned from the value returned**
81. **\* from checkAmmountOfLeagues method + 1.**
82. **\* Asks for input of league name.**
83. **\* Asks for input of team names until all team slots are filled,**
84. **\* a simple loop inside the method only allows a-zA-Z characters.**
85. **\* Upon creation, all teams are printed out for verification and printed to file.**
86. **\* Executes fixture generation.**
87. **\* File format: "(league number) + \_participants.txt"**
88. **\*/**
89. **public static void createLeague() throws IOException {**
90. **//JOptionPane, asks for League name and number of teams, automatically asigns league number based on how many already exist**
91. **// JOptionPane, asks to input team names corresponding to number of teams**
92. **leagueInfo.createNewFile();**
93. **String leagueName;**
94. **int leagueNumber = checkAmmountOfLeagues();**
95. **int numberOfTeams;**
96. **String adminNumber = "admin";**
97. **String fileTeamNames = "";**
98. **String teamNames = "";**
99. **FileWriter outputStream = new FileWriter(leagueInfo,true);**
100. **FileWriter outputStream2 = new FileWriter((leagueNumber+1) + "\_participants.txt",true);**
101. **leagueNumber++;**
102. **PrintWriter pw = new PrintWriter(outputStream);**
103. **PrintWriter pw2 = new PrintWriter(outputStream2);**
104. **leagueName = (String) JOptionPane.showInputDialog(null,"Please enter League Name","");**
105. **numberOfTeams = Integer.parseInt(JOptionPane.showInputDialog(null,leagueName + "\n How many teams are to be in the league?"));**
106. **String[] leagueTeamNames = new String[numberOfTeams];**
107. **for (int i = 0; i < numberOfTeams;i++) {**
108. **leagueTeamNames[i] = (String) (JOptionPane.showInputDialog(null,"("+ (i + 1) + ")" + " Enter team name"));**
109. **if (leagueTeamNames[i].matches(".\*[^a-zA-Z].\*")) { //Only Alphabetical letters**
110. **JOptionPane.showMessageDialog(null,"Team names may only contain letters!");**
111. **i--; //If input is wrong then then the loop doesn't go forward.**
112. **continue;**
113. **}**
114. **teamNames += (i + 1) + ". " + leagueTeamNames[i] + "\n";**
115. **fileTeamNames = (i + 1) + "," + leagueTeamNames[i];**
116. **pw2.println(fileTeamNames);**
117. **}**
118. **JOptionPane.showMessageDialog(null,teamNames);**
119. **String LeagueDetails = leagueNumber + "," + leagueName + "," + currentAdminNum;**
120. **pw.println(LeagueDetails);**
121. **pw.close();**
122. **pw2.close();**
123. **generateFixtures();**
124. **}**
125. **/\*\***
126. **\* Fixtures are generated right after creating a league,**
127. **\* calls getNumberOfTeams method to get number of teams**
128. **\* Checks for odd number of teams and creates a "dummy" fixture.**
129. **\* int MatchCounter keeps track of match number, for example there will be 780 in a 20 team league.**
130. **\* Keeps track of Home and Away teams.**
131. **\* Prints to file.**
132. **\* File format: "(league number) + \_fixtures.txt"**
133. **\*/**
134. **public static void generateFixtures() throws IOException {**
135. **int numberOfTeams, totalNumberOfRounds, numberOfMatchesPerRound;**
136. **int roundNumber, matchNumber, homeTeamNumber = 0, awayTeamNumber = 0, even, odd;**
137. **boolean additionalTeamIncluded = false;**
138. **int leagueNumber = checkAmmountOfLeagues();**
139. **File f = new File(leagueNumber + "\_fixtures.txt");**
140. **FileWriter fixt = new FileWriter(f,true);**
141. **PrintWriter pwFixt = new PrintWriter(fixt);**
142. **int selection;**
143. **int matchCounter = 1;**
144. **String [][] fixtures;**
145. **String [][] revisedFixtures;**
146. **String [] elementsOfFixture;**
147. **String fixtureAsText;**
148. **selection = getNumberOfTeams();**
149. **if (selection != 0)**
150. **{**
151. **numberOfTeams = selection;**
152. **if (numberOfTeams % 2 == 1) {**
153. **numberOfTeams++;**
154. **additionalTeamIncluded = true;**
155. **}**
156. **totalNumberOfRounds = numberOfTeams - 1;**
157. **numberOfMatchesPerRound = numberOfTeams / 2;**
158. **fixtures = new String[totalNumberOfRounds][numberOfMatchesPerRound];**
159. **for (roundNumber = 0; roundNumber < totalNumberOfRounds; roundNumber++) {**
160. **for (matchNumber = 0; matchNumber < numberOfMatchesPerRound; matchNumber++) {**
161. **homeTeamNumber = (roundNumber + matchNumber) % (numberOfTeams - 1);**
162. **awayTeamNumber = (numberOfTeams - 1 - matchNumber + roundNumber) % (numberOfTeams - 1);**
163. **if (matchNumber == 0)**
164. **awayTeamNumber = numberOfTeams - 1;**
165. **fixtures[roundNumber][matchNumber] = (homeTeamNumber + 1) + "," + (awayTeamNumber + 1);**
166. **}**
167. **}**
168. **revisedFixtures = new String[totalNumberOfRounds][numberOfMatchesPerRound];**
169. **even = 0;**
170. **odd = numberOfTeams / 2;**
171. **for (int i = 0; i < fixtures.length; i++) {**
172. **if (i % 2 == 0)**
173. **revisedFixtures[i] = fixtures[even++];**
174. **else**
175. **revisedFixtures[i] = fixtures[odd++];**
176. **}**
177. **fixtures = revisedFixtures;**
178. **for (roundNumber = 0; roundNumber < fixtures.length; roundNumber++) {**
179. **if (roundNumber % 2 == 1) {**
180. **fixtureAsText = fixtures[roundNumber][0];**
181. **elementsOfFixture = fixtureAsText.split(",");**
182. **fixtures[roundNumber][0] = elementsOfFixture[1] + "," + elementsOfFixture[0];**
183. **}**
184. **}**
185. **for(int x = 0; x<fixtures.length; x++) {**
186. **for(int y = 0; y < fixtures[x].length; y++) {**
187. **pwFixt.println(matchCounter + "," + fixtures[x][y]);**
188. **matchCounter++;**
189. **}**
190. **}**
191. **}**
192. **if (selection != 0)**
193. **{**
194. **numberOfTeams = selection;**
195. **if (numberOfTeams % 2 == 1)**
196. **{**
197. **numberOfTeams++;**
198. **additionalTeamIncluded = true;**
199. **}**
200. **totalNumberOfRounds = numberOfTeams - 1;**
201. **numberOfMatchesPerRound = numberOfTeams / 2;**
202. **fixtures = new String[totalNumberOfRounds][numberOfMatchesPerRound];**
203. **for (roundNumber = 0; roundNumber < totalNumberOfRounds; roundNumber++)**
204. **{**
205. **for (matchNumber = 0; matchNumber < numberOfMatchesPerRound; matchNumber++)**
206. **{**
207. **homeTeamNumber = (roundNumber + matchNumber) % (numberOfTeams - 1);**
208. **awayTeamNumber = (numberOfTeams - 1 - matchNumber + roundNumber) % (numberOfTeams - 1);**
209. **if (matchNumber == 0)**
210. **awayTeamNumber = numberOfTeams - 1;**
211. **fixtures[roundNumber][matchNumber] = (awayTeamNumber + 1) + "," + (homeTeamNumber + 1);**
212. **}**
213. **}**
214. **revisedFixtures = new String[totalNumberOfRounds][numberOfMatchesPerRound];**
215. **even = 0;**
216. **odd = numberOfTeams / 2;**
217. **for (int i = 0; i < fixtures.length; i++)**
218. **{**
219. **if (i % 2 == 0)**
220. **revisedFixtures[i] = fixtures[even++];**
221. **else**
222. **revisedFixtures[i] = fixtures[odd++];**
223. **}**
224. **fixtures = revisedFixtures;**
225. **for (roundNumber = 0; roundNumber < fixtures.length; roundNumber++)**
226. **{**
227. **if (roundNumber % 2 == 1)**
228. **{**
229. **fixtureAsText = fixtures[roundNumber][0];**
230. **elementsOfFixture = fixtureAsText.split(",");**
231. **fixtures[roundNumber][0] = elementsOfFixture[1] + "," + elementsOfFixture[0];**
232. **}**
233. **}**
234. **for(int x = 0; x<fixtures.length; x++) {**
235. **for(int y = 0; y < fixtures[x].length; y++) {**
236. **pwFixt.println(matchCounter + "," + fixtures[x][y]);**
237. **matchCounter++;**
238. **}**
239. **}**
240. **}**
241. **pwFixt.close();**
242. **}**
243. **/\*\***
244. **\* This method is called during generating fixtures,**
245. **\* scans "(league number)\_participants.txt" for number of teams**
246. **\* present. Returns value.**
247. **\*/**
248. **public static int getNumberOfTeams() throws IOException {**
249. **int numberOfnumberOfTeams = 0;**
250. **Scanner in;**
251. **int leagueNumber = checkAmmountOfLeagues();**
252. **String lineFromFile;**
253. **String fileElements[];**
254. **File x = new File (leagueNumber + "\_participants.txt");**
255. **in = new Scanner(x);**
256. **while (in.hasNext())**
257. **{**
258. **lineFromFile = in.nextLine();**
259. **fileElements = lineFromFile.split(",");**
260. **numberOfnumberOfTeams = Integer.parseInt(fileElements[0]);**
261. **}**
262. **if (numberOfnumberOfTeams < 2) {**
263. **JOptionPane.showMessageDialog(null,"Error. Team number < 2", "Error. Team number < 2", 2);**
264. **}**
265. **in.close();**
266. **return numberOfnumberOfTeams;**
267. **}**
268. **public static void viewLeaderboard() throws IOException {**
269. **boolean readFile;**
270. **readFile = readFilesIntoArrayLists();**
271. **if (!readFile)**
272. **System.out.println("One or more files do not exist.");**
273. **else**
274. **{**
275. **createEmptyLeaderBoard();**
276. **processResults();**
277. **orderLeaderBoard();**
278. **displayLeaderboard();**
279. **}**
280. **}**
281. **public static boolean readFilesIntoArrayLists() throws IOException {**
282. **int leagueNumber =Integer.parseInt(JOptionPane.showInputDialog(null, "Which league would you like to view?"));**
283. **String fileElements[];**
284. **File inputFile1 = new File(leagueNumber + "\_participants.txt");**
285. **File inputFile2 = new File(leagueNumber + "\_fixtures.txt");**
286. **File inputFile3 = new File(leagueNumber + "\_outcomes.txt");**
287. **teams = new ArrayList<ArrayList<String>>();**
288. **teams.add(new ArrayList<String>());**
289. **teams.add(new ArrayList<String>());**
290. **fixtures = new ArrayList<ArrayList<Integer>>();**
291. **fixtures.add(new ArrayList<Integer>());**
292. **fixtures.add(new ArrayList<Integer>());**
293. **fixtures.add(new ArrayList<Integer>());**
294. **results = new ArrayList<ArrayList<Integer>>();**
295. **results.add(new ArrayList<Integer>());**
296. **results.add(new ArrayList<Integer>());**
297. **results.add(new ArrayList<Integer>());**
298. **if (inputFile1.exists() && inputFile2.exists() && inputFile3.exists()) {**
299. **Scanner in;**
300. **in = new Scanner(inputFile1);**
301. **while(in.hasNext())**
302. **{**
303. **fileElements = (in.nextLine()).split(",");**
304. **teams.get(0).add(fileElements[0]);**
305. **teams.get(1).add(fileElements[1]);**
306. **}**
307. **in.close();**
308. **in = new Scanner(inputFile2);**
309. **while(in.hasNext())**
310. **{**
311. **fileElements = (in.nextLine()).split(",");**
312. **fixtures.get(0).add(Integer.parseInt(fileElements[0]));**
313. **fixtures.get(1).add(Integer.parseInt(fileElements[1]));**
314. **fixtures.get(2).add(Integer.parseInt(fileElements[2]));**
315. **}**
316. **in.close();**
317. **in = new Scanner(inputFile3);**
318. **while(in.hasNext()) {**
319. **fileElements = (in.nextLine()).split(",");**
320. **results.get(0).add(Integer.parseInt(fileElements[0]));**
321. **results.get(1).add(Integer.parseInt(fileElements[1]));**
322. **results.get(2).add(Integer.parseInt(fileElements[2]));**
323. **}**
324. **in.close();**
325. **return true;**
326. **}**
327. **else**
328. **return false;**
329. **}**
330. **public static void createEmptyLeaderBoard()**
331. **{**
332. **// find out the number of teams/players which will determine**
333. **// the number of rows**
334. **int rows = teams.get(0).size();**
335. **int columns = 14;**
336. **leaderBoard = new int[rows][columns];**
337. **// place team numbers in column 0 of leader board**
338. **for (int i = 0; i < leaderBoard.length; i++)**
339. **leaderBoard[i][0] = Integer.parseInt(teams.get(0).get(i));**
340. **}**
341. **public static void processResults()**
342. **{**
343. **int fixtureNumber, homeTeamScore, awayTeamScore, homeTeamNumber, awayTeamNumber;**
344. **int position;**
345. **for (int i = 0; i < results.get(0).size(); i++)**
346. **{**
347. **fixtureNumber = results.get(0).get(i);**
348. **homeTeamScore = results.get(1).get(i);**
349. **awayTeamScore = results.get(2).get(i);**
350. **position = fixtures.get(0).indexOf(fixtureNumber);**
351. **homeTeamNumber = fixtures.get(1).get(position);**
352. **awayTeamNumber = fixtures.get(2).get(position);**
353. **if (homeTeamScore == awayTeamScore)**
354. **{**
355. **recordFixtureResultForHomeTeam(homeTeamNumber,0,1,0,homeTeamScore,awayTeamScore,1);**
356. **recordFixtureResultForAwayTeam(awayTeamNumber,0,1,0,homeTeamScore,awayTeamScore,1);**
357. **}**
358. **else if (homeTeamScore > awayTeamScore)**
359. **{**
360. **recordFixtureResultForHomeTeam(homeTeamNumber,1,0,0,homeTeamScore,awayTeamScore,3);**
361. **recordFixtureResultForAwayTeam(awayTeamNumber,0,0,1,homeTeamScore,awayTeamScore,0);**
362. **}**
363. **else**
364. **{**
365. **recordFixtureResultForHomeTeam(homeTeamNumber,0,0,1,homeTeamScore,awayTeamScore,0);**
366. **recordFixtureResultForAwayTeam(awayTeamNumber,1,0,0,homeTeamScore,awayTeamScore,3);**
367. **}**
368. **}**
369. **}**
370. **public static void recordFixtureResultForHomeTeam(int hTN, int w, int d, int l, int hTS, int aTS, int p)**
371. **{**
372. **leaderBoard[hTN-1][1]++; // gamesPlayed**
373. **leaderBoard[hTN-1][2]+= w; // homeWin**
374. **leaderBoard[hTN-1][3]+= d; // homeDraw**
375. **leaderBoard[hTN-1][4]+= l; // homeLoss**
376. **leaderBoard[hTN-1][5]+= hTS; // homeTeamScore**
377. **leaderBoard[hTN-1][6]+= aTS; // awayTeamScore**
378. **leaderBoard[hTN-1][12] += (hTS - aTS); // goalDifference**
379. **leaderBoard[hTN-1][13] += p; // points**
380. **}**
381. **public static void recordFixtureResultForAwayTeam(int aTN, int w, int d, int l, int hTS, int aTS, int p)**
382. **{**
383. **leaderBoard[aTN-1][1]++; // gamesPlayed**
384. **leaderBoard[aTN-1][7]+= w; // awayWin**
385. **leaderBoard[aTN-1][8]+= d; // awayDraw**
386. **leaderBoard[aTN-1][9]+= l; // awayLoss**
387. **leaderBoard[aTN-1][10]+= aTS; // awayTeamScore**
388. **leaderBoard[aTN-1][11]+= hTS; // homeTeamScore**
389. **leaderBoard[aTN-1][12] += (aTS - hTS); // goalDifference**
390. **leaderBoard[aTN-1][13] += p; // points**
391. **}**
392. **public static void orderLeaderBoard()**
393. **{**
394. **int [][] temp = new int[leaderBoard.length][leaderBoard[0].length];**
395. **boolean finished = false;**
396. **while (!finished)**
397. **{**
398. **finished = true;**
399. **for (int i = 0; i < leaderBoard.length - 1; i++)**
400. **{**
401. **if (leaderBoard[i][13] < leaderBoard[i + 1][13])**
402. **{**
403. **for (int j = 0; j < leaderBoard[i].length; j++)**
404. **{**
405. **temp[i][j] = leaderBoard[i][j];**
406. **leaderBoard[i][j] = leaderBoard[i + 1][j];**
407. **leaderBoard[i + 1][j] = temp[i][j];**
408. **}**
409. **finished = false;**
410. **}**
411. **}**
412. **}**
413. **}**
414. **public static void displayLeaderboard()**
415. **{**
416. **int aTeamNumber;**
417. **String aTeamName, formatStringTeamName;**
418. **String longestTeamName = teams.get(1).get(0);**
419. **int longestTeamNameLength = longestTeamName.length();**
420. **for (int i = 1; i < teams.get(1).size(); i++)**
421. **{**
422. **longestTeamName = teams.get(1).get(i);**
423. **if (longestTeamNameLength < longestTeamName.length())**
424. **longestTeamNameLength = longestTeamName.length();**
425. **}**
426. **formatStringTeamName = "%-" + (longestTeamNameLength + 2) + "s";**
427. **System.out.printf(formatStringTeamName,"Team Name");**
428. **System.out.println(" GP HW HD HL GF GA AW AD AL GF GA GD TP");**
429. **for (int i = 0; i < leaderBoard.length; i++)**
430. **{**
431. **aTeamNumber = leaderBoard[i][0];**
432. **aTeamName = teams.get(1).get(aTeamNumber - 1);**
433. **System.out.printf(formatStringTeamName, aTeamName);**
434. **System.out.printf("%4d", leaderBoard[i][1]);**
435. **System.out.printf("%4d", leaderBoard[i][2]);**
436. **System.out.printf("%4d", leaderBoard[i][3]);**
437. **System.out.printf("%4d", leaderBoard[i][4]);**
438. **System.out.printf("%4d", leaderBoard[i][5]);**
439. **System.out.printf("%4d", leaderBoard[i][6]);**
440. **System.out.printf("%4d", leaderBoard[i][7]);**
441. **System.out.printf("%4d", leaderBoard[i][8]);**
442. **System.out.printf("%4d", leaderBoard[i][9]);**
443. **System.out.printf("%4d", leaderBoard[i][10]);**
444. **System.out.printf("%4d", leaderBoard[i][11]);**
445. **System.out.printf("%5d", leaderBoard[i][12]);**
446. **System.out.printf("%5d", leaderBoard[i][13]);**
447. **System.out.println();**
448. **}**
449. **}**
450. **/\*\***
451. **\* Scans "leagueInfo.txt",**
452. **\* prints to console.**
453. **\*/**
454. **public static void viewLeagues() throws IOException {**
455. **Scanner in;**
456. **String lineFromFile;**
457. **in = new Scanner(leagueInfo);**
458. **while(in.hasNext())**
459. **{**
460. **lineFromFile = in.nextLine();**
461. **System.out.println(lineFromFile);**
462. **}**
463. **in.close();**
464. **}**
465. **/\*\* \*User is asked to enter the name of the league he would like to view**
466. **\*If the league name exists in leagueInfo file and he is an admin of that league then the teams will**
467. **\*be displayed.**
468. **\*When checking the team name and admin number matches,the league number is also stored in a variable**
469. **\*This is used to view the team list which would be stored in a text file called (LeagueNumber + \_participants.txt)**
470. **\*/**
471. **public static void viewTeams() throws IOException {**
472. **String inputLeagueName = JOptionPane.showInputDialog(null,"Enter the name of the league you would like to view");**
473. **int tempLeagueNum, tempAdminNum;**
474. **String tempLeagueName;**
475. **Scanner in;**
476. **Scanner on;**
477. **String lineFromFile;**
478. **String lineFromFile2;**
479. **String FileElements[];**
480. **in = new Scanner(leagueInfo);**
481. **boolean found = false;**
482. **while(in.hasNext() && ! found) {**
483. **lineFromFile =in.nextLine();**
484. **FileElements = lineFromFile.split(",");**
485. **tempLeagueNum = Integer.parseInt(FileElements[0]);**
486. **tempLeagueName = FileElements[1];**
487. **tempAdminNum = Integer.parseInt(FileElements[2]);**
488. **if(inputLeagueName.trim().equals(tempLeagueName) && tempAdminNum == currentAdminNum) {**
489. **found = true;**
490. **File y = new File(tempLeagueNum + "\_participants.txt");**
491. **on = new Scanner(y);**
492. **while(on.hasNext()) {**
493. **lineFromFile2 = on.nextLine();**
494. **System.out.println (lineFromFile2);**
495. **}**
496. **on.close();**
497. **}**
498. **}**
499. **in.close();**
500. **}**
501. **/\*\* \*user enters league number,checks if league number matches admin number of the user**
502. **\*Then reads team names from participants file and adds to array**
503. **\*Then reads team names and assigns them a number,this number -1 will be equal to their**
504. **\*position in the fixtures file,allowing is to replace the numbers with the actual team names**
505. **\*Fixture numbers and the the two teams playing against eachother are then printed**
506. **\*/**
507. **public static void viewFixtures() throws IOException {**
508. **int inputLeagueNum = Integer.parseInt(JOptionPane.showInputDialog(null,"What number league would like to view the fixtures of?"));**
509. **int tempLeagueNum, tempAdminNum, homeTeamNum, awayTeamNum, fixtureNum;**
510. **String tempLeagueName;**
511. **Scanner in; // scanner for finding current league**
512. **Scanner on; // scanner for going through the fixtures file**
513. **Scanner an; // scanner for going through the participants file**
514. **String lineFromFile, lineFromFile2, homeTeamName, awayTeamName;**
515. **String fileElements[];**
516. **String fileElements2[];**
517. **ArrayList<String> allParicipantDetails = new ArrayList<String>();**
518. **in = new Scanner(leagueInfo);**
519. **boolean found = false;**
520. **while(in.hasNext() && ! found) {**
521. **lineFromFile =in.nextLine();**
522. **fileElements = lineFromFile.split(",");**
523. **tempLeagueNum = Integer.parseInt(fileElements[0]);**
524. **tempLeagueName = fileElements[1];**
525. **tempAdminNum = Integer.parseInt(fileElements[2]);**
526. **if(inputLeagueNum==tempLeagueNum && tempAdminNum == currentAdminNum) { //find correct league and ensure admin has access**
527. **found = true;**
528. **File x = new File(tempLeagueNum + "\_participants.txt");**
529. **File y = new File(tempLeagueNum + "\_fixtures.txt");**
530. **on = new Scanner(y);**
531. **an = new Scanner(x);**
532. **while(an.hasNext()) {**
533. **lineFromFile2 = an.nextLine();**
534. **fileElements2 = lineFromFile2.split(",");**
535. **allParicipantDetails.add(fileElements2[1]); // adds only names of teams to the array**
536. **} // team 1 is in position 0, team 2 in pos 1, etc**
537. **while(on.hasNext()) {**
538. **lineFromFile = on.nextLine();**
539. **fileElements = lineFromFile.split(",");**
540. **fixtureNum = Integer.parseInt(fileElements[0]);**
541. **homeTeamNum = Integer.parseInt(fileElements[1]);**
542. **awayTeamNum = Integer.parseInt(fileElements[2]);**
543. **homeTeamName=allParicipantDetails.get(homeTeamNum-1); //gets the name sof the teams based on the postion of**
544. **awayTeamName=allParicipantDetails.get(awayTeamNum-1); // the teaam in the arraylist**
545. **System.out.println (fileElements[0] + ". " + homeTeamName + " v " + awayTeamName); // displays the fixture**
546. **}**
547. **on.close();**
548. **an.close();**
549. **}**
550. **}**
551. **in.close();**
552. **}**
553. **/\*\* \*User is asked to input number of league he would like to check**
554. **\*Checks if the league number exists and if it matches the users admin number**
555. **\*Team names are recieved through the particpants file and are matched with their**
556. **\*corresponding numbers in fixtures**
557. **\*Results are displayed using the fixture number,user is asked to input the home and away team scores which is then printed to**
558. **\*the file 'outcomes'**
559. **\*When the fixture file has no next line,(no more fitxtures available) the loop will end and the user will not be asked to**
560. **\*input anymore results.**
561. **\*/**
562. **public static void addResults() throws IOException { // similiar to the view fixtures method with the addition of another scanner**
563. **int inputLeagueNum = Integer.parseInt(JOptionPane.showInputDialog(null,"What number league would like to add results to?"));**
564. **int tempLeagueNum, tempAdminNum, homeTeamNum, awayTeamNum, fixtureNum, printAwayScore, printHomeScore;**
565. **int currentFixtureNum = 1;**
566. **Scanner in;**
567. **Scanner on;**
568. **Scanner an;**
569. **Scanner un; // scanner to determin the current fixture that is to be input**
570. **String lineFromFile, lineFromFile2, homeTeamName, awayTeamName, tempLeagueName, fix;**
571. **String fileElements[];**
572. **String fileElements2[];**
573. **ArrayList<String> allParicipantDetails = new ArrayList<String>();**
574. **in = new Scanner(leagueInfo);**
575. **boolean found = false;**
576. **while(in.hasNext() && ! found) {**
577. **lineFromFile =in.nextLine();**
578. **fileElements = lineFromFile.split(",");**
579. **tempLeagueNum = Integer.parseInt(fileElements[0]);**
580. **tempLeagueName = fileElements[1];**
581. **tempAdminNum = Integer.parseInt(fileElements[2]);**
582. **if(inputLeagueNum==tempLeagueNum && tempAdminNum == currentAdminNum) {**
583. **found = true;**
584. **File x = new File(tempLeagueNum + "\_participants.txt");**
585. **File y = new File(tempLeagueNum + "\_fixtures.txt");**
586. **File z = new File(tempLeagueNum + "\_outcomes.txt");**
587. **FileWriter printed = new FileWriter(z,true);**
588. **PrintWriter out = new PrintWriter(printed);**
589. **on = new Scanner(y);**
590. **an = new Scanner(x);**
591. **un = new Scanner(z);**
592. **while(an.hasNext()) {**
593. **lineFromFile2 = an.nextLine();**
594. **fileElements2 = lineFromFile2.split(",");**
595. **allParicipantDetails.add(fileElements2[1]);**
596. **}**
597. **while(un.hasNext()) { // only the earliest of fixtures can be inputed one at a time**
598. **un.nextLine();**
599. **currentFixtureNum++;**
600. **}**
601. **un.close();**
602. **while(on.hasNext()) {**
603. **lineFromFile = on.nextLine();**
604. **fileElements = lineFromFile.split(",");**
605. **fixtureNum = Integer.parseInt(fileElements[0]);**
606. **homeTeamNum = Integer.parseInt(fileElements[1]);**
607. **awayTeamNum = Integer.parseInt(fileElements[2]);**
608. **homeTeamName=allParicipantDetails.get(homeTeamNum-1);**
609. **awayTeamName=allParicipantDetails.get(awayTeamNum-1);**
610. **fix = (fixtureNum + ". " + homeTeamName + " v " + awayTeamName);**
611. **if (fixtureNum==currentFixtureNum) {**
612. **printHomeScore = Integer.parseInt(JOptionPane.showInputDialog(null, fix + "\n Home Team Score:"));**
613. **// users input the score of each team**
614. **printAwayScore = Integer.parseInt(JOptionPane.showInputDialog(null, fix + "\n Away Team Score:"));**
615. **out.println(fixtureNum + "," + printHomeScore + "," + printAwayScore);**
616. **}**
617. **}**
618. **out.close();**
619. **on.close();**
620. **an.close();**
621. **}**
622. **}**
623. **in.close();**
624. **}**
625. **/\*\***
626. **\* If the "userInfo.txt" file is empty or does not exist,**
627. **\* prompts user to create a new account, otherwise gives option**
628. **\* to create a new account or login to an existing one.**
629. **\* While loop allows 3 attempts to login.**
630. **\* During accountCreation the last Admin number is scanned**
631. **\* to check for which number to assign to the new account.**
632. **\***
633. **\* After login the users admin number is assigned to the global variable**
634. **\* "currentAdminNum"**
635. **\* Prints out logged in users name + admin number.**
636. **\*/**
637. **public static void verifyLogin() throws IOException {**
638. **String userName = "", userPassword = "";**
639. **String newUser = "";**
640. **FileWriter fw = new FileWriter(accountInfo,true);**
641. **PrintWriter pw3 = new PrintWriter(fw);**
642. **ArrayList<String> userNamesAndPasswords = new ArrayList<String>();**
643. **ArrayList<String> allUserDetails = new ArrayList<String>();**
644. **Scanner in;**
645. **String lineFromFile;**
646. **String fileElements[];**
647. **String message1 = "Please enter your username";**
648. **String message2 = "Please enter your password";**
649. **String message3 = "Invalid input, please re-try";**
650. **String message4 = "Invalid input, no more attempts";**
651. **int chance = 1;**
652. **int createdAdminNumber = 0;**
653. **int selectedOption;**
654. **int adminNumber, position;**
655. **boolean validInput = false, validAccessDetails = false;**
656. **String aUser = "", tempUserDetails;**
657. **if(accountInfo.length() == 0) {**
658. **userName = JOptionPane.showInputDialog("Please enter desired username");**
659. **userPassword = JOptionPane.showInputDialog("Please enter desired password");**
660. **adminNumber = 1;**
661. **newUser = (userName + "," + userPassword + "," + adminNumber);**
662. **pw3.println(newUser);**
663. **System.out.println("Login Created, relaunch");**
664. **pw3.close();**
665. **System.exit(0);**
666. **}**
667. **selectedOption = JOptionPane.showConfirmDialog(null,"Have you already created an account?","LOGIN",JOptionPane.YES\_NO\_OPTION);**
668. **if (selectedOption == JOptionPane.NO\_OPTION) {**
669. **userName = JOptionPane.showInputDialog("Please enter desired username");**
670. **userPassword = JOptionPane.showInputDialog("Please enter desired password");**
671. **in = new Scanner(accountInfo);**
672. **while(in.hasNext()) {**
673. **lineFromFile = in.nextLine();**
674. **fileElements = lineFromFile.split(",");**
675. **createdAdminNumber = ((Integer.parseInt(fileElements[2])) + 1);**
676. **}**
677. **in.close();**
678. **newUser = (userName + "," + userPassword + "," + createdAdminNumber);**
679. **pw3.println(newUser);**
680. **JOptionPane.showMessageDialog(null,"Login Created, relaunch");**
681. **pw3.close();**
682. **System.exit(0);**
683. **} else if (selectedOption == JOptionPane.YES\_OPTION) {**
684. **in = new Scanner(accountInfo);**
685. **while(in.hasNext()) {**
686. **lineFromFile = in.nextLine();**
687. **fileElements = lineFromFile.split(",");**
688. **userNamesAndPasswords.add(fileElements[0] + "," + fileElements[1]);**
689. **allUserDetails.add(lineFromFile);**
690. **}**
691. **in.close();**
692. **} else**
693. **System.out.println("User file not found");**
694. **while((!(validInput)) && (chance <= 3)) {**
695. **userName = JOptionPane.showInputDialog(null, message1);**
696. **if (userName != null) {**
697. **userPassword = JOptionPane.showInputDialog(null, message2);**
698. **if (userPassword != null) {**
699. **aUser = userName + "," + userPassword;**
700. **if (userNamesAndPasswords.contains(aUser))**
701. **{**
702. **validInput = true;**
703. **validAccessDetails = true;**
704. **} else {**
705. **chance += 1;**
706. **if (chance <= 3)**
707. **JOptionPane.showMessageDialog(null, message3);**
708. **else {**
709. **JOptionPane.showMessageDialog(null, message4);**
710. **System.exit(0);**
711. **}**
712. **}**
713. **} else**
714. **validInput = true;**
715. **} else**
716. **validInput = true;**
717. **}**
718. **if (validAccessDetails) {**
719. **position = userNamesAndPasswords.indexOf(aUser);**
720. **tempUserDetails = allUserDetails.get(position);**
721. **fileElements = tempUserDetails.split(",");**
722. **adminNumber = Integer.parseInt(fileElements[2]);**
723. **currentAdminNum = adminNumber;**
724. **System.out.println("Hello " + userName + currentAdminNum);**
725. **} else**
726. **System.out.print("Goodbye");**
727. **}**
728. **}**