CIS2109/2149 Object Oriented Programming

Coursework Report

Contents

[Analysis and Design 3](#_Toc532046512)

[Identification 3](#_Toc532046513)

[Nouns accepted as objects: 3](#_Toc532046514)

[Not accepted: 3](#_Toc532046515)

[Added: 3](#_Toc532046516)

[Table of Entities: 4](#_Toc532046517)

[Use-Case Diagram 6](#_Toc532046518)

[Class Diagram 7](#_Toc532046519)

[Application Development 8](#_Toc532046520)

[A. GUI 8](#_Toc532046521)

[B. Add/Remove Teams 10](#_Toc532046522)

[C. Add/Remove Players To/From The Teams 16](#_Toc532046523)

[D. Match Creation 20](#_Toc532046524)

[E. League Table Based On Scores 22](#_Toc532046525)

[F. Match Report 23](#_Toc532046526)

[Data Storing/Loading 26](#_Toc532046527)

[Error Handling 28](#_Toc532046528)

[Extra features 32](#_Toc532046529)

[Reset Scores 32](#_Toc532046530)

[Sufficient Player Checking 33](#_Toc532046531)

[Team Editing Menu 34](#_Toc532046532)

[Program and OO Paradigm 36](#_Toc532046533)

[Reflection 36](#_Toc532046534)

[Bibliography 37](#_Toc532046535)

# Analysis and Design

## Identification

### Nouns accepted as objects:

* Player
* Team
* Coach
* Manager
* Referee
* Match

### Not accepted:

* Role – This is an attribute of a player
* Captain – This is an attribute of the team
* Game – Means the same as a match
* Officials – Refers to referees and coaches?
* Outcome – This is the collection of attributes of a match

### Added:

* Person – This will be a more generalised object for the different entities such as players, coaches, managers and referees because they all share certain attributes, so it will serve as the super class for those entities.
* League – Where everything will be managed, and temporarily stored such as teams, matches and all the different entities.
* DataHandler – This class will contain the methods necessary to save and load the required data.
* Launcher – This class will be the main class and will launch the program.

## Table of Entities:

|  |  |  |
| --- | --- | --- |
| Object | Attributes | Behaviours |
| Person | name: String  age: int | Serves as template for other sub classes |
| Player | role: String  shirtNumber: int | Sub class of person  Will be added to a specific team or to a list of players with no team |
| Coach |  | Sub class of person. Will be added to a specific team, or to a list of coaches that have no team assigned |
| Manager |  | Sub class of person. Will be added to a specific team, or to a list of managers that have no team assigned |
| Team | name: String  coach: Coach  -manager: Manager  players: ArrayList<Player>  - leagueScore: int  wins: int  losses: int  ties: int  captain: Player | Will be able to add and store the players to the team.  Needs to check if it has enough players to play a match, set and get a team captain, reset scores and remove players from the team. |
| Referee |  | Sub class of person. Will participate in matches. Stored in a list of referees |
| Match | matchTitle: String  matchReferee: Referee  homeTeam: Team  awayTeam: Team  homeTeamGoals: int  awayTeamGoals: int | Matches will be created composed of two teams, one referee and with the information of the goals of each team. |
| League | leagueTeams: ArrayList<Team>  leagueMatches: ArrayList<Match>  leaguePlayers: ArrayList<Player>  playersWithNoTeam: ArrayList<Player>  leagueReferees: ArrayList<Referee>  leagueCoaches: ArrayList<Coach>  availableCoaches: ArrayList<Coach>  leagueManagers: ArrayList<Manager>  availableManagers:ArrayList<Manager> | Display a league table based on scores  It will check if a team as played home against the other team, will reset the scores of every team  It will store coaches, managers, players, teams, referees and matches, and will also be able to delete them.  Will perform checks to see if an entity already exists |
| DataHandler |  | It will save and load all the required entities: Teams, Matches, Coaches, Players, Managers and Referees |
| Lancher | availableCoachesPath: String  availableManagersPath: String  coachesPath: String  dataHandler: DataHandler  league: League  managersPath: String  playersPathl: String  refereesPath: String  teamsPath: String | It will create an instance of league and launch the GUI  It will also hand the data to/from the DataHandler Class to the League class instance |

## Use-Case Diagram

After identifying the entities of the program the following Use-Case Diagram was created:

## Class Diagram

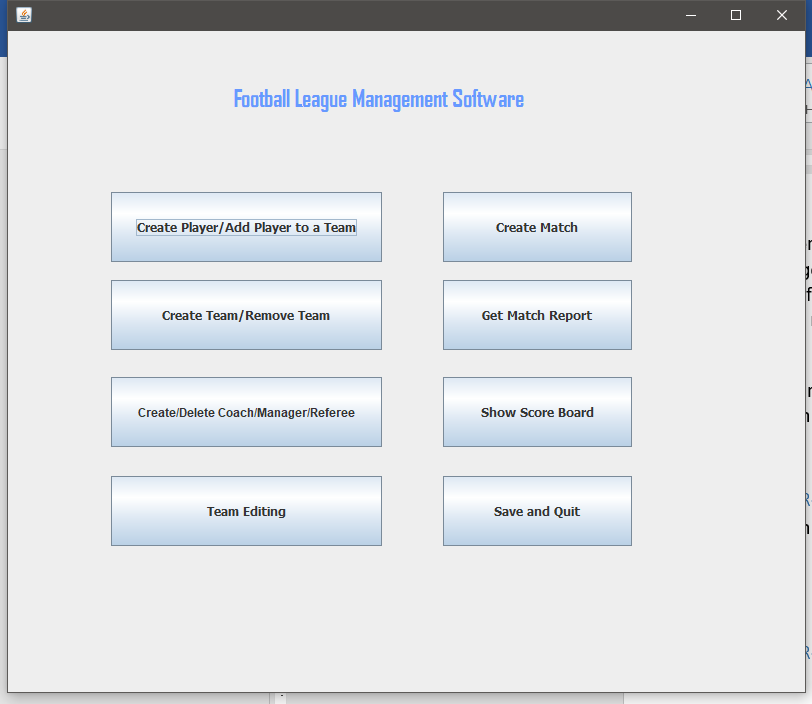
Below is the class diagram for the program:



# Application Development

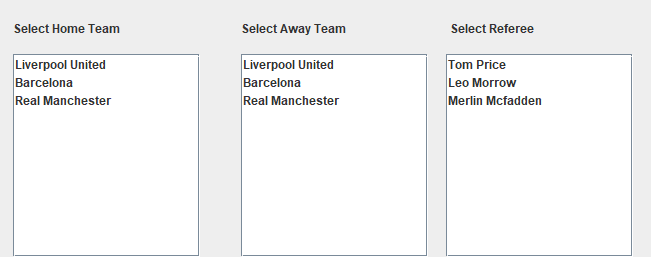
## A. GUI

The entire program uses a graphical user interface from which the user can navigate between the different forms that will allow him to do different tasks and functions.



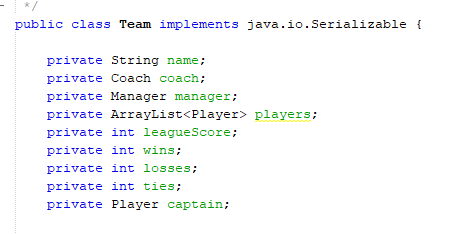
The main menu is the starting point of the program, and from here the user can press any of the buttons depending on what he desires to do.

Lists were chosen to both display objects such as players, teams and matches and also to get user input in order to prevent errors. This way, the user only has to select an item from the list, and getting the select index from the list it is possible to easily get said match, team, player and others from the respective arrays.



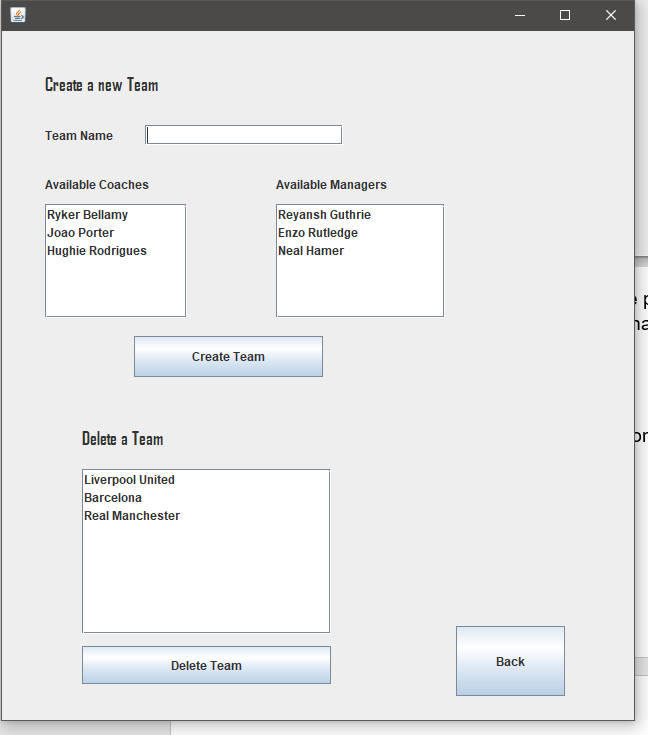
## B. Add/Remove Teams

The class Team has the following attributes:

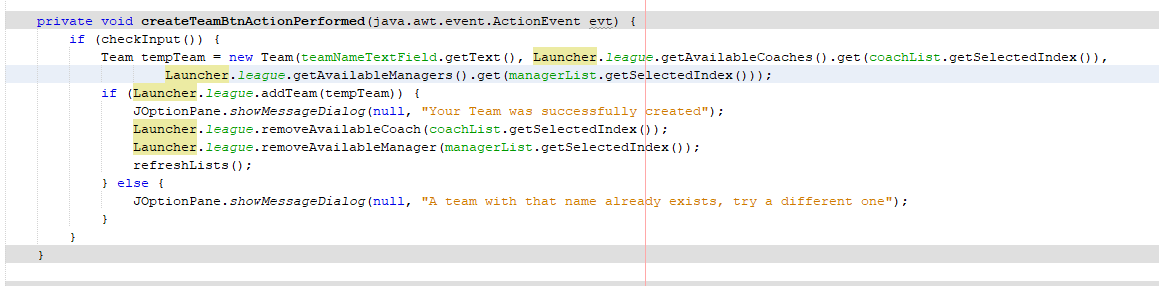


There are variables for win losses and ties because it is assumed that for a match to occur these attributes should be stored and will be used to provide the score for each team.

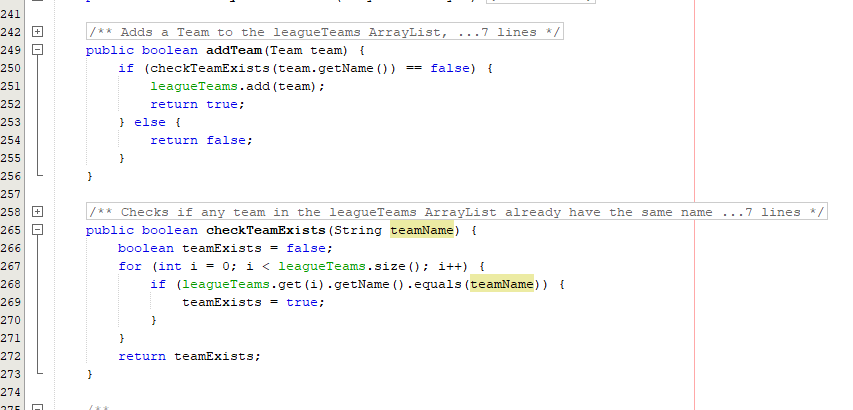
Creation and deletion of Teams can be done in the Create Team/Remove Team form.



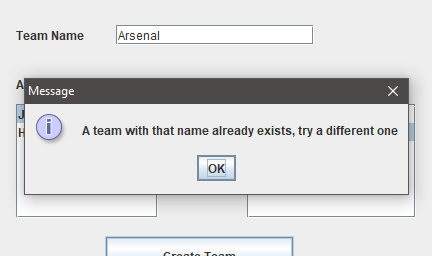
To create a team, the user will need to input a name, and then select from the list of coaches and managers. A coach and manager are both attributes of a team because the scenario stated that a team will consist of 13 players, one manager and one coach. When the user presses the create team button the following code is executed:



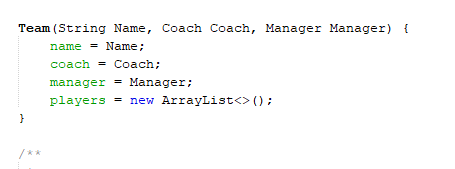
This references the addTeam method in the League Class:



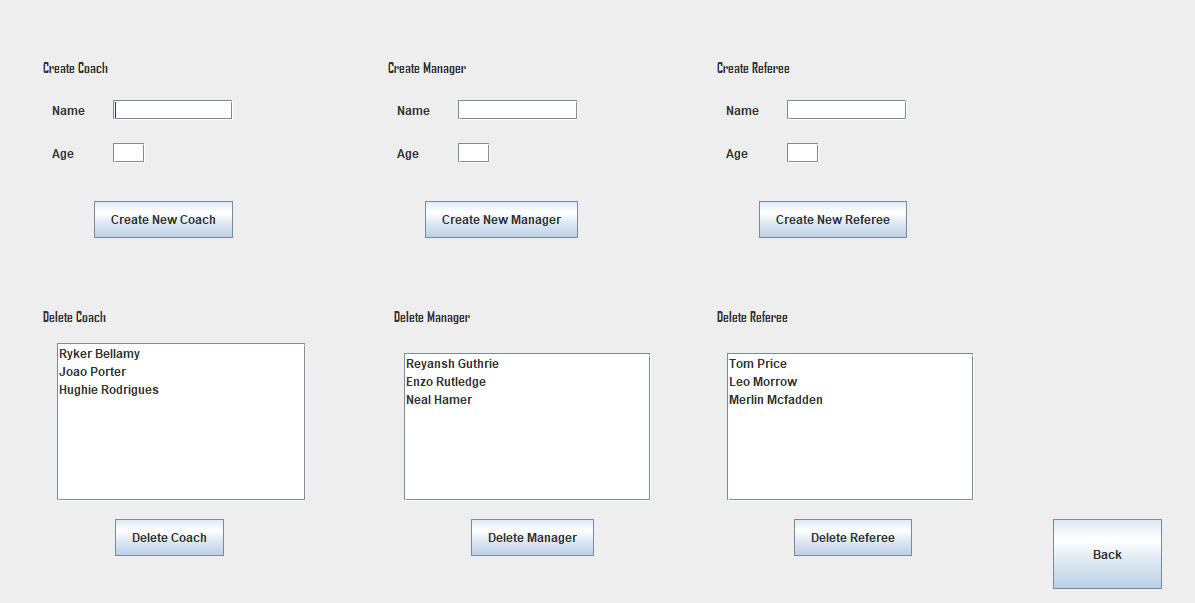
When the addTeam method returns false, it means that a team with the same name already exists, so user feedback is given, and the team is not created.



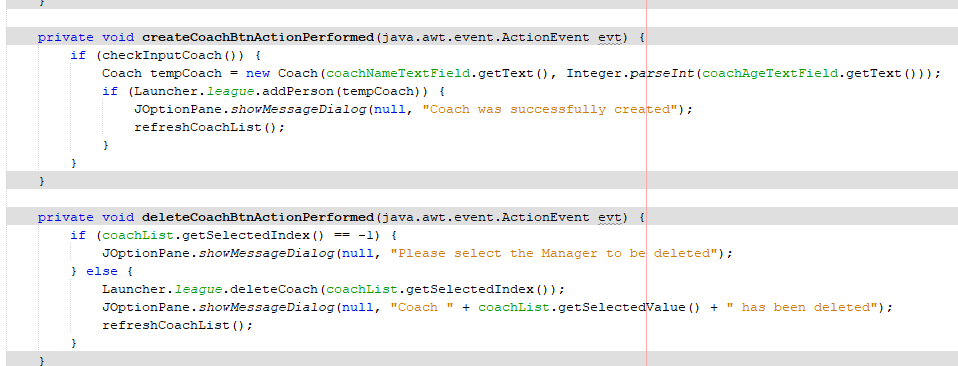
Team constructor:



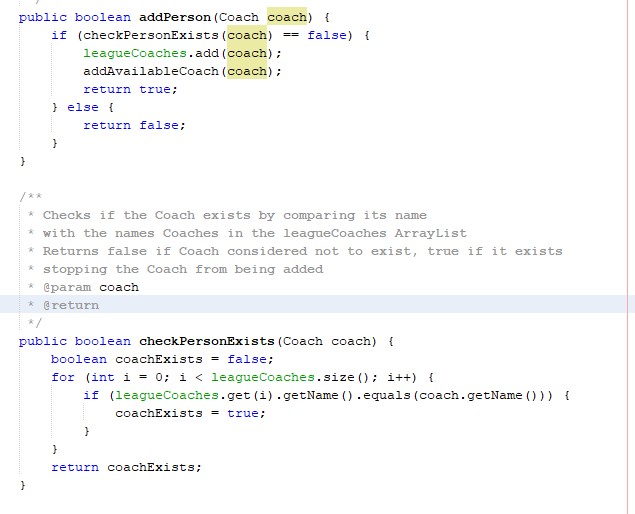
Because to create a team, a coach and manager is required, a form for this function was created when the user clicks the Create/Delete Coach/Manager/Referee button on the main menu:



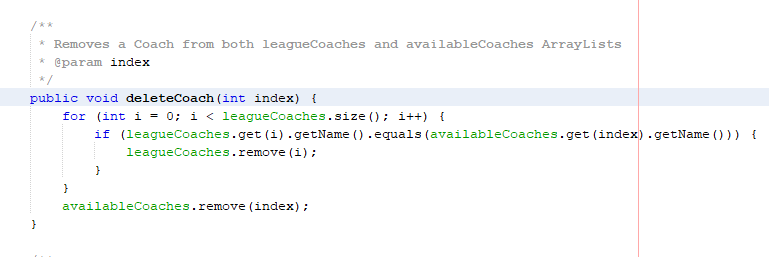
The process is the same for any of the three entities, however they are stored in their respective ArrayLists. The code for when the create new or delete buttons are pressed can be found below:



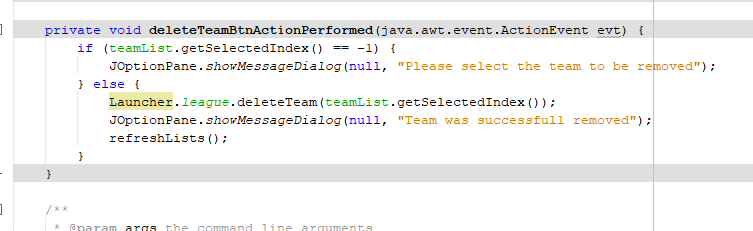
The code above references to the following methods in the League Class:



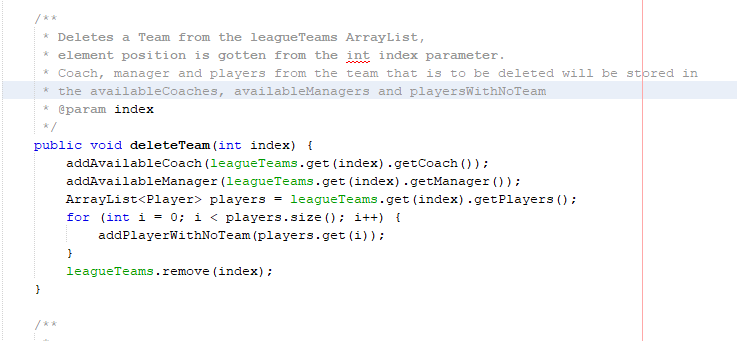
The addPerson method is overloaded for each different entity such as Player, Coach, Manager or Referee



To remove a team, the user can select any from the list of teams, press the delete team button and the following code is executed:



That code references to the deleteTeam method in the League Class:



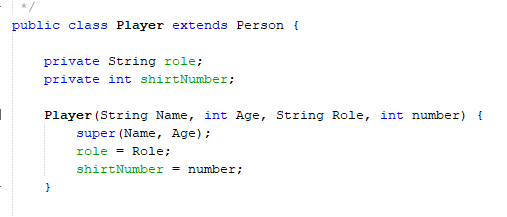
This not only removes the team from the leagueTeams ArrayList but also, puts the coach,manager and players from said team in the availableCoaches, availableManagers and playersWithNoTeam ArrayLists so they can used for other teams.

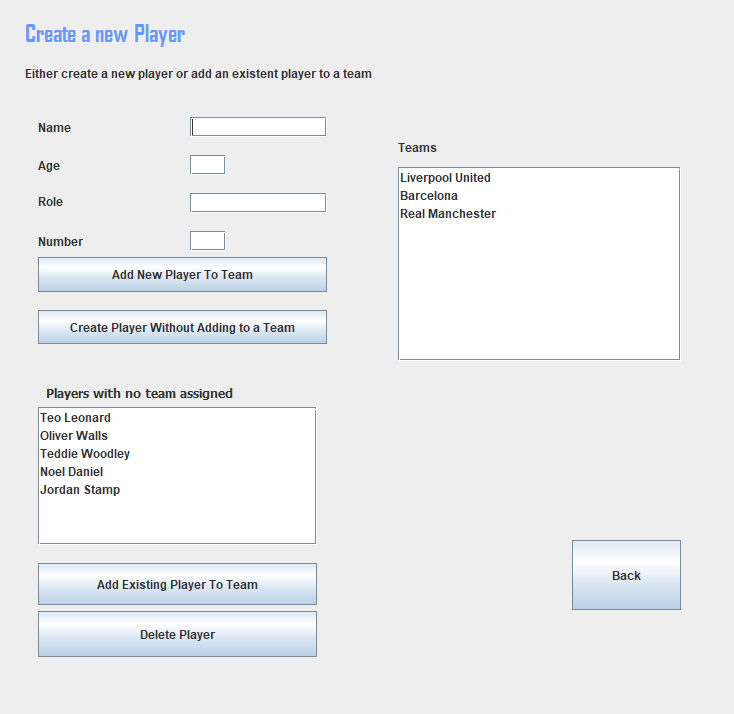
## C. Add/Remove Players To/From The Teams

A player can be created, and added straight away to a Team, or can be created and not be assigned to a team straight away, which is the list at the left bottom, from this list, the user can add those players that have no teams assigned to a team and they will be removed from the available list.

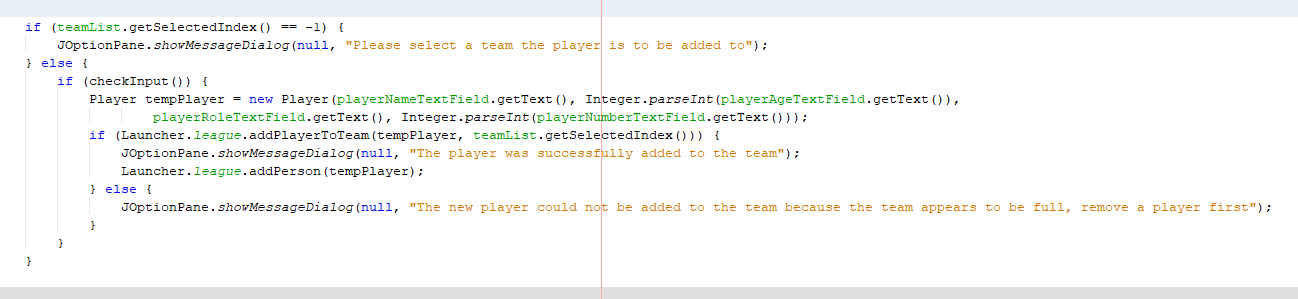
Players take name, age, role and number as attributes which are input by the user in the respective text fields.

Player Constructor:

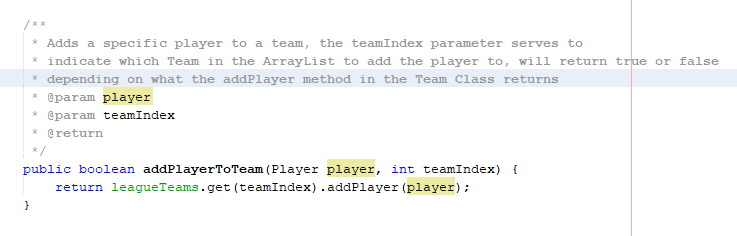




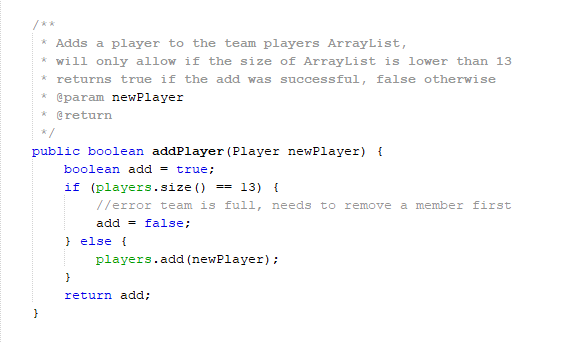
When the user presses the Add New Player To Team button the follow code is executed:



The method that adds the player to a certain team, takes the selected index of the team list and gives this as parameter to the addPlayerToTeam method in the League Class, the method can be found below:

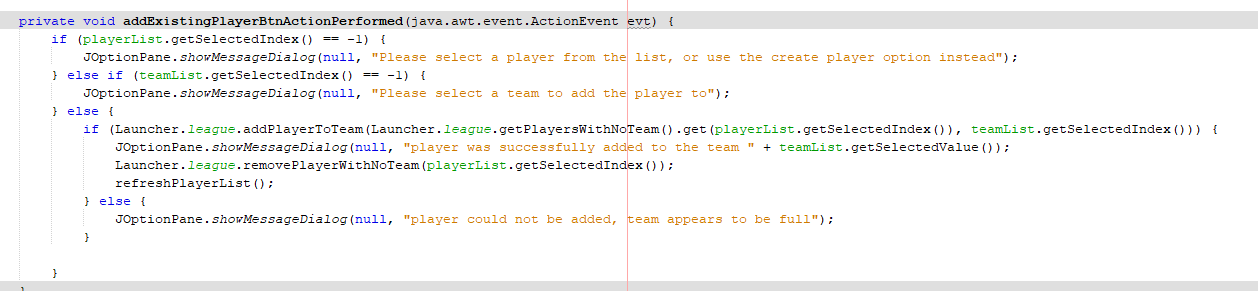


This references the following code in the Team Class:

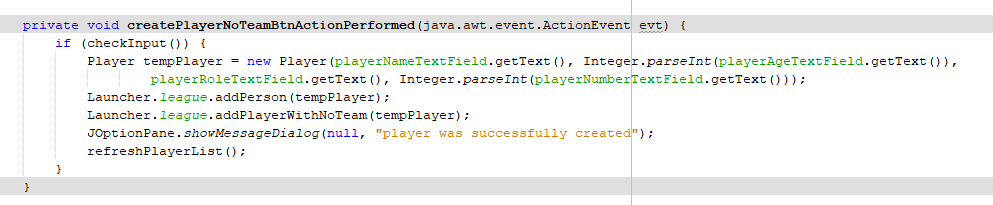


Because the scenario stated that a team consists of 13 players the addPlayer method will return false if the players ArrayList size is 13 stopping the user from adding more players to the team.

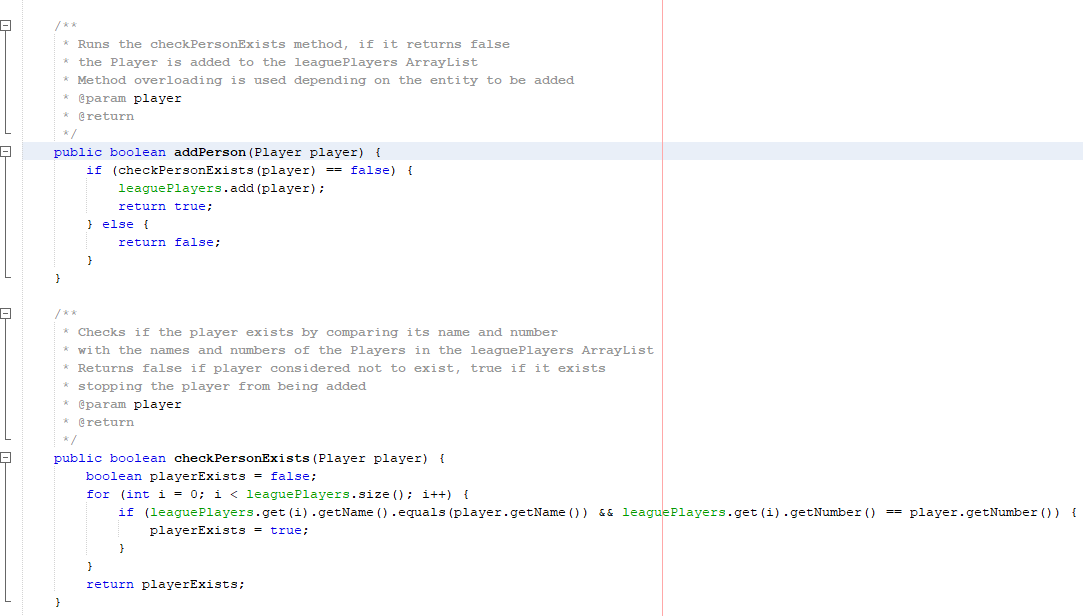
When adding a player to a team from the players with no team list as an alternative, it will get the player from the playersWithNoTeams ArrayList instead, and feed it to the addPlayerToTeam method in the League Class.

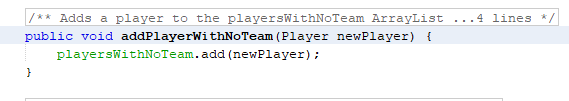


If the user decides instead to just create the player without adding it to a team the following code will execute:

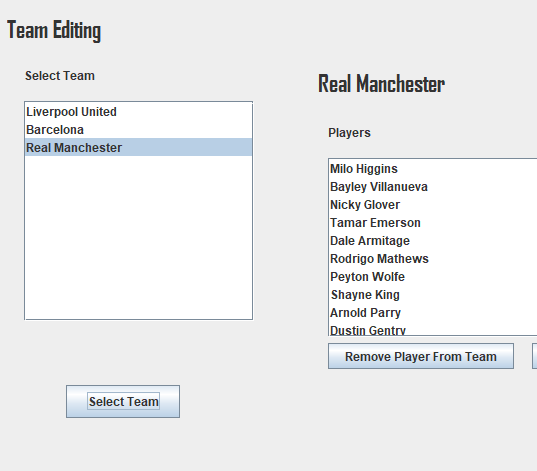
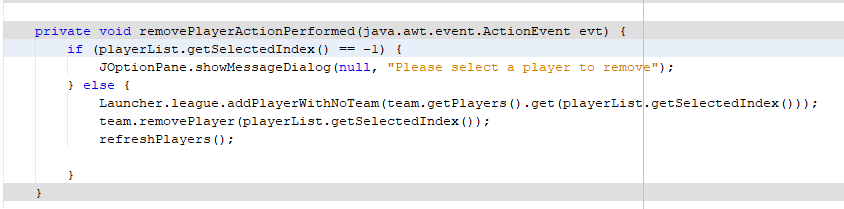


This references the following code in the League Class:

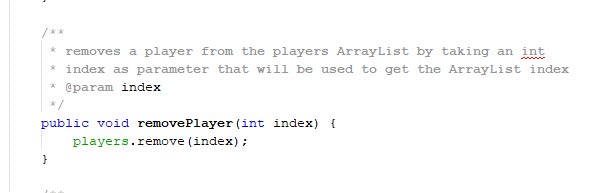




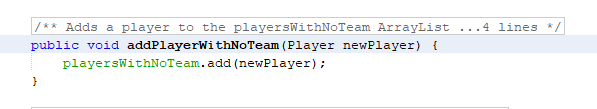
Removing a Player from a Team can be done in the Team Editing Form, when the user selects a team a list will show with every player that is on that team. The user selects the player that wants to remove from the team and presses the remove player from team button. This player will then be stored in the playersWithNoTeam ArrayList, ready to be added to another team.

This button references the following code in the Team Class:

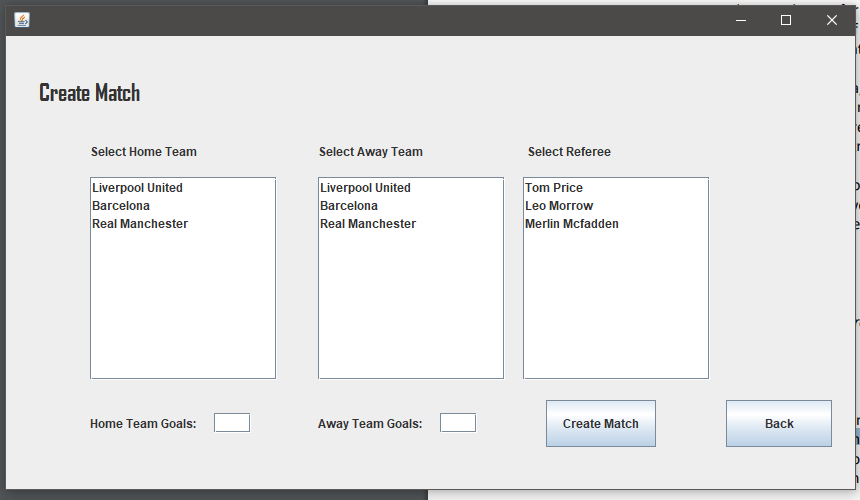


And the following code in the League class:

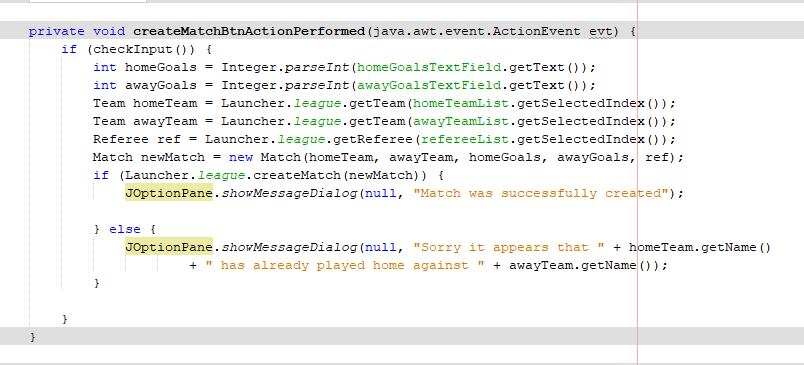


## D. Match Creation

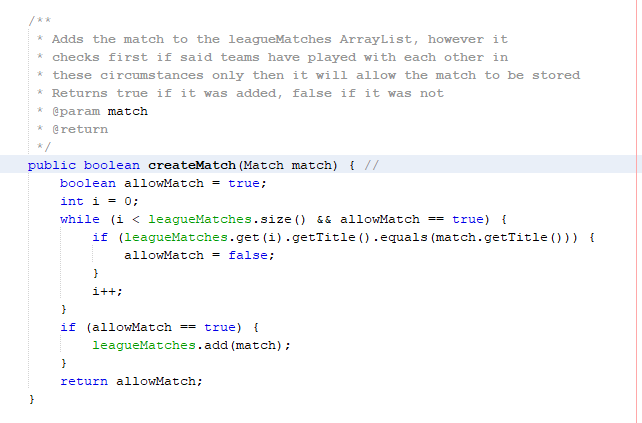
A match can be created by using the Create Match form:

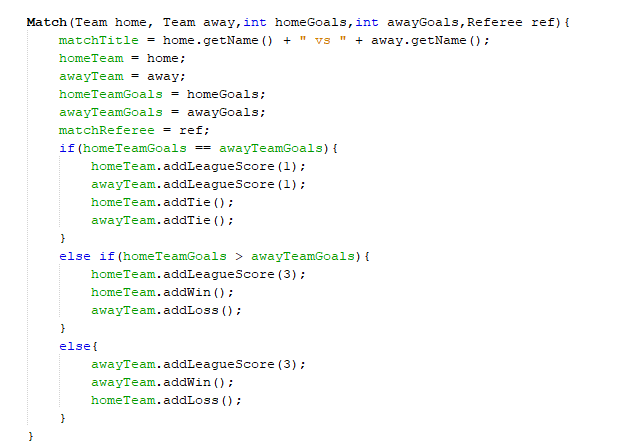


Here the user selects the Home, Away Team and Referee from the respective lists, then input the goals of both teams, and press the create match button. The following code runs:



If the checkInput method returns true, then it means the user has input all the correct information to create a match. This then references to the following code in the League Class:

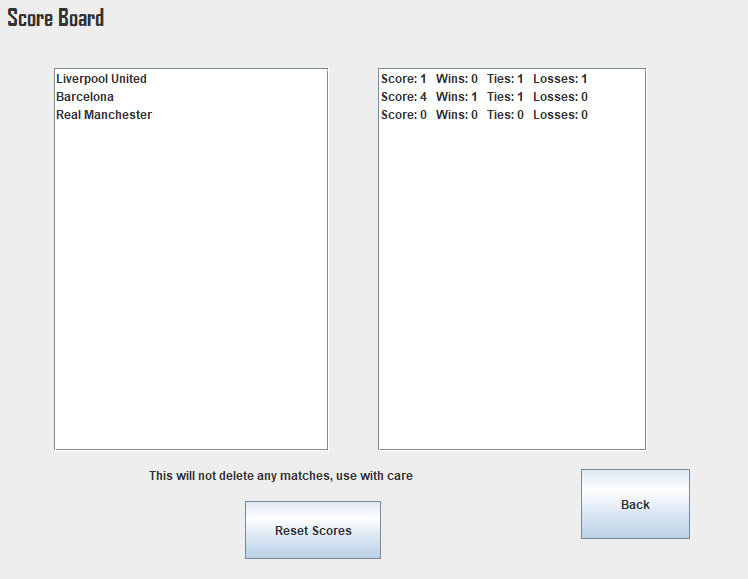


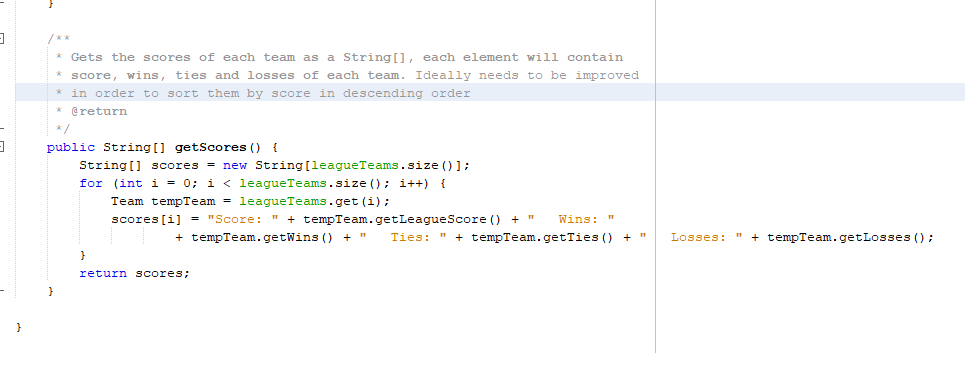
Match constructor: 

When a match is created, the home Team, away Team and Referee are stored, so are the respective goals of each team. Each team will have scores added and a win loss or tie depending on the goals. A Match title will also be created stating the Team vs the other Team.

## E. League Table Based On Scores

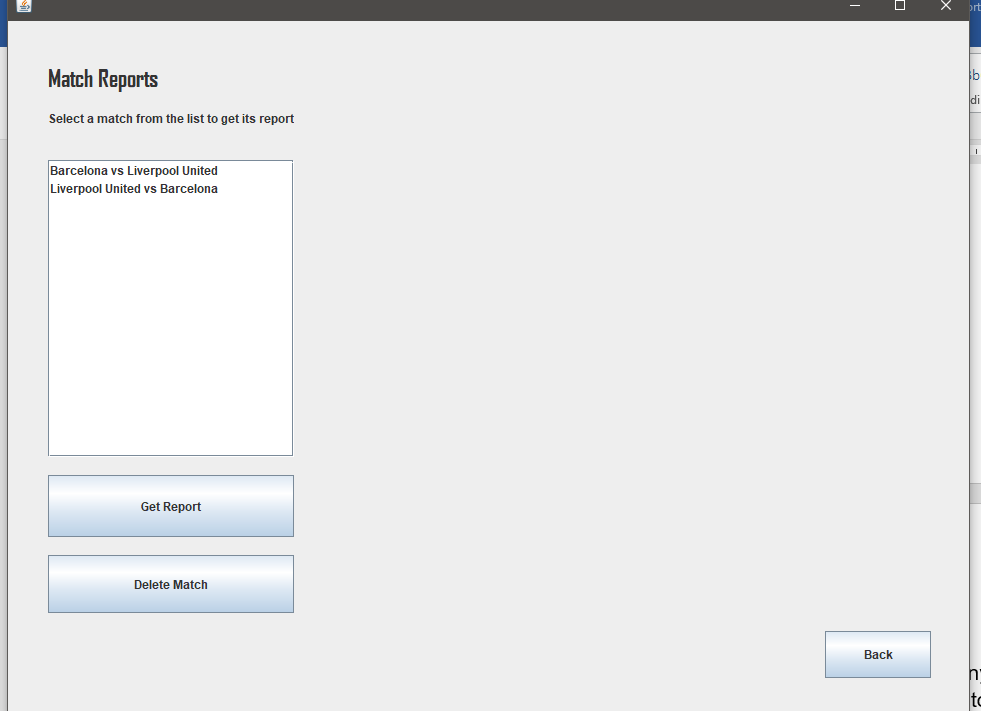
The program successfully creates a league table based on scores, it gets each team, and displays score, wins, ties and losses for each one. However, this feature is very poorly implemented because it could be massively improved by sorting scores in descending order. The first list displays the teams, and the second one, the score information of each.



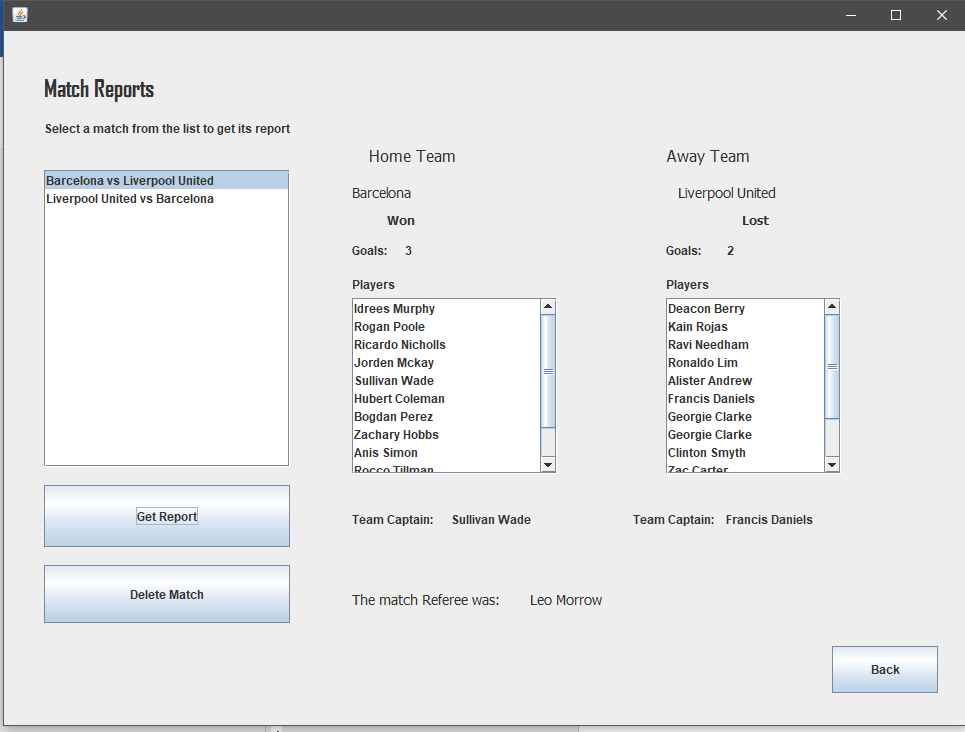


## F. Match Report

The user can get a print of any match by using the Get Match Report button from the Main Menu. It will take them to the following form:

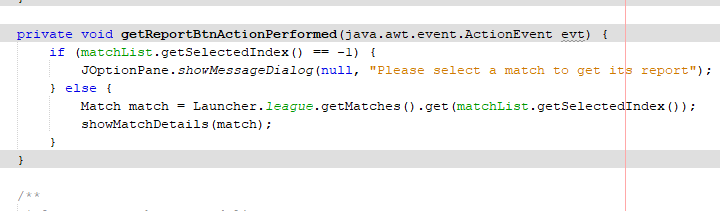


When a Match is selected the following information is displayed:

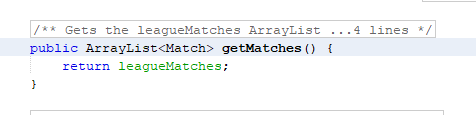


The user can find specific information for the home and away team such as goals, who won, the players of each team, the team captain and the match referee.

The following code is executed when the Get Report button is pressed:

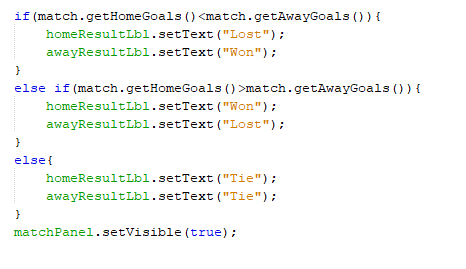


This references the following code in the League Class which returns the ArrayList matches and from there it gets the specific match using the selected index from the list in the GUI.

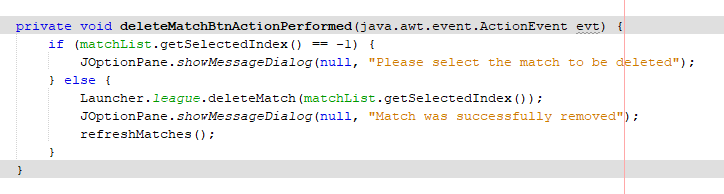


Then with the match acquired, the showMatchDetails method is executed:

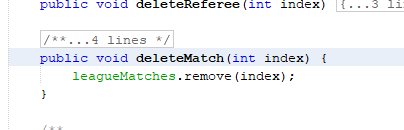




The user can also choose to delete a certain match as an extra feature:



Code Referenced in the League Class:



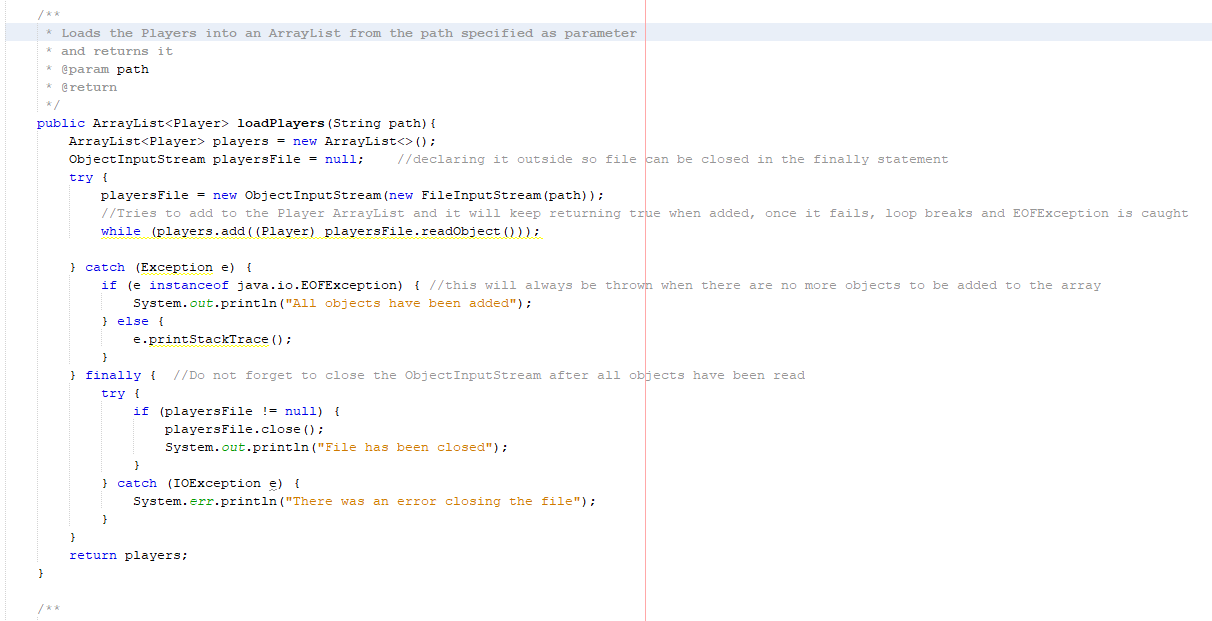
This is not a perfect refined feature because, it does not dynamically change the scores of each team. So even though the match is deleted, the final scores remain the same. This could lead to inaccurate information.

## Data Storing/Loading

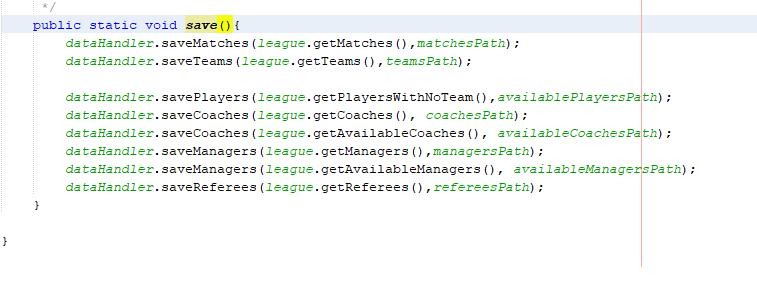
A dedicated Class named DataHandler was created with the purpose of storing and loading all the data, for this program that included every team, players, managers, coaches, referees and matches. The fact that all the data required to be saved were in specific ArrayLists in the League class made it easier to save everything and load everything back to them. The path of each file is saved as a variable in the Launcher class so these can be easily changed when needed.

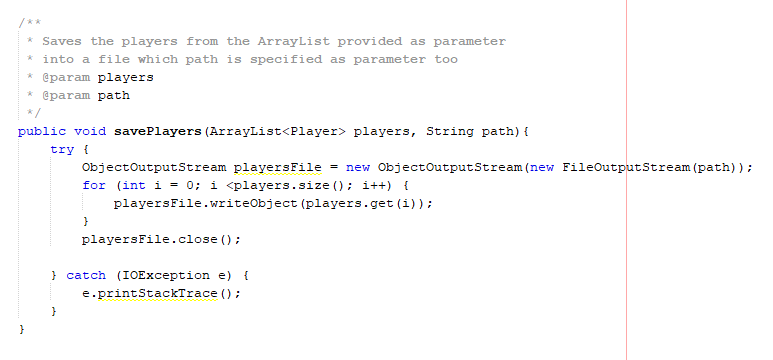


When the program starts, the main method creates a new league instance loading every array with the respective data, using the DataHandler class.



When the user presses the save and quit button the following code is executed:



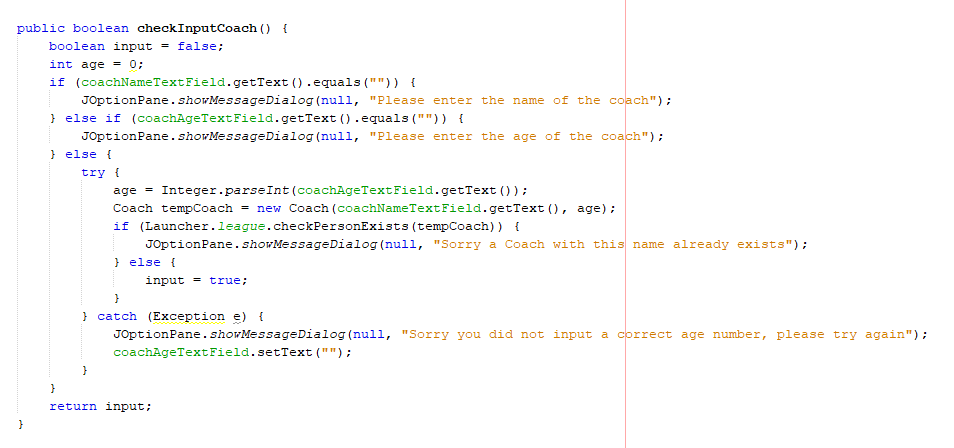


To

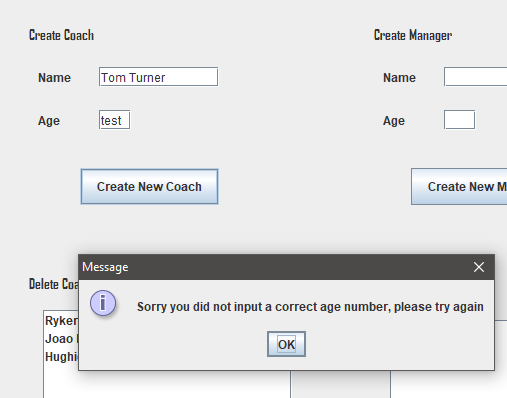
# Error Handling

Error handling occurs whenever the user needs to input something or perform certain actions, such as when trying to create a player, team, officials or when creating a match. User feedback is given appropriately to further help the user understand what went wrong. **Why is this helpful?**

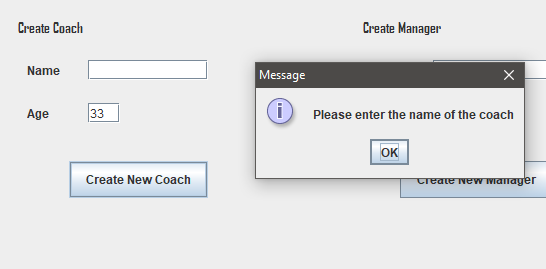
When text fields are used to store data into integer variables, appropriately error catching was used:



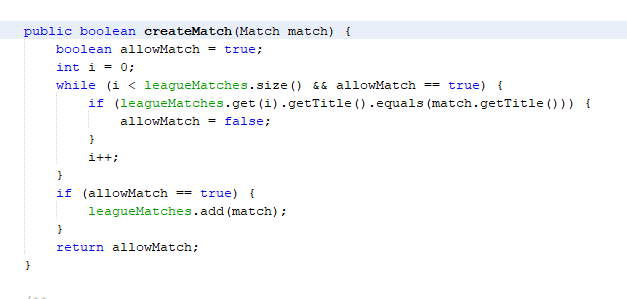
If the user does not input a correct integer, a pop up message will tell the user where this happened, and the respective text field is cleared.

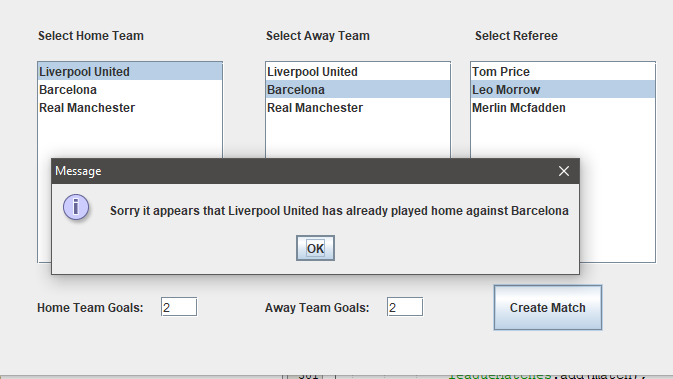


Similar pop ups will show if any of the text fields are blank:

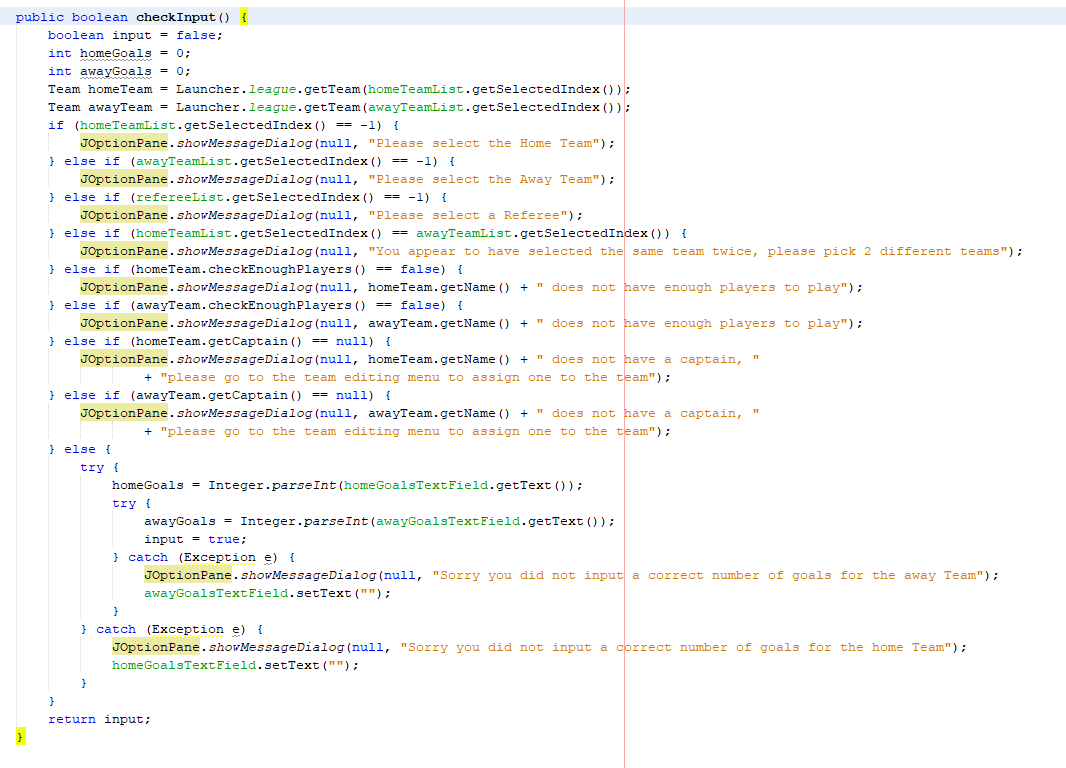


Another example of error handling is when the user tries to create a match, because the scenario states each team will only play each other twice, (once home once away), before creating a match, a check is made to see if both teams have played each other in the same situation, if so the match cannot be created and the user receives feedback on this. If the match can be created, it is then stored into the leagueMatches ArrayList.

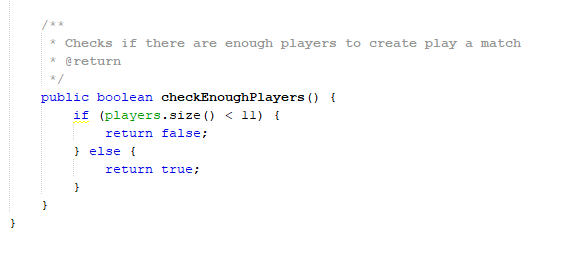




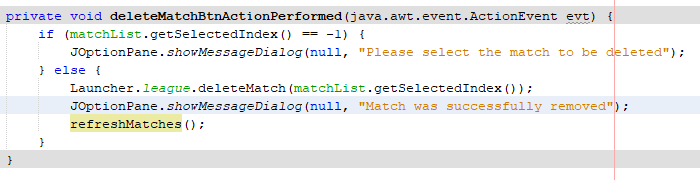
Not only that but other checks are made in the checkInputMethod when the user presses the create a match button. This checks things such as if both teams have a team captain assigned, and enough players to play or even if the user has selected both teams and a referee:

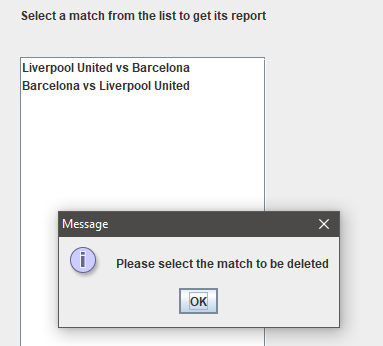


This References the following code in the Team Class:



A check is also performed whenever it is expected for a user to select an item from a list in order to prevent errors, so if nothing was selected, a pop up message will give the user feedback but the rest of the code does not run.



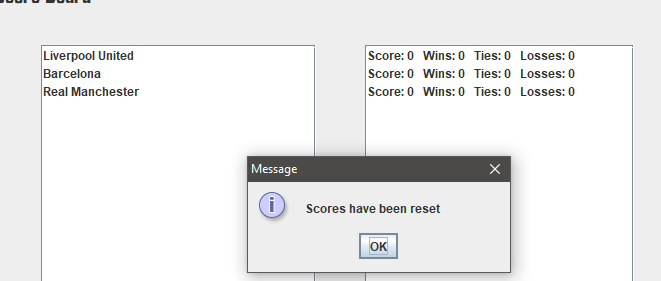


# Extra features

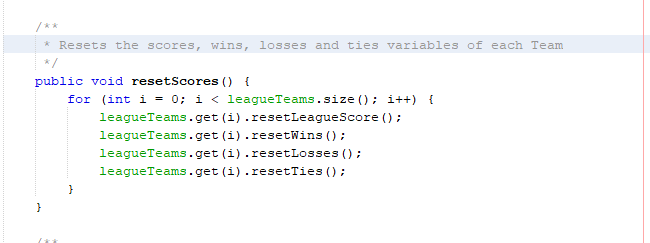
Every feature needs to be mentioned

## Reset Scores

Something additional that was implemented was a button on the Score Board form that the user could press to reset the scores of every team. This is not a finished or refined feature because, it will not delete any matches, and so there will be a conflict between the two. However, it is possible for the user to remove the matches manually so, providing they are careful with its usage it could be useful in certain circumstances.

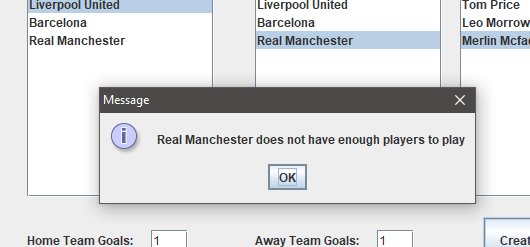


The Score Board form refers to the method in the league class, which in turn refers to the specific methods of each team.

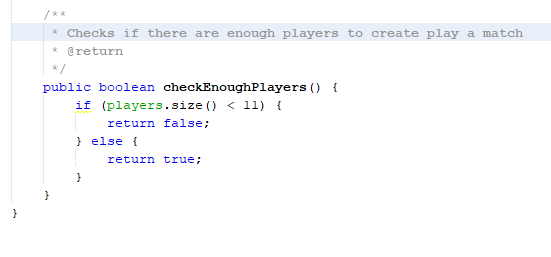


## Sufficient Player Checking

Before a match is created it is first checked if the Team has enough players to play, if not a pop up will give feedback to the user:



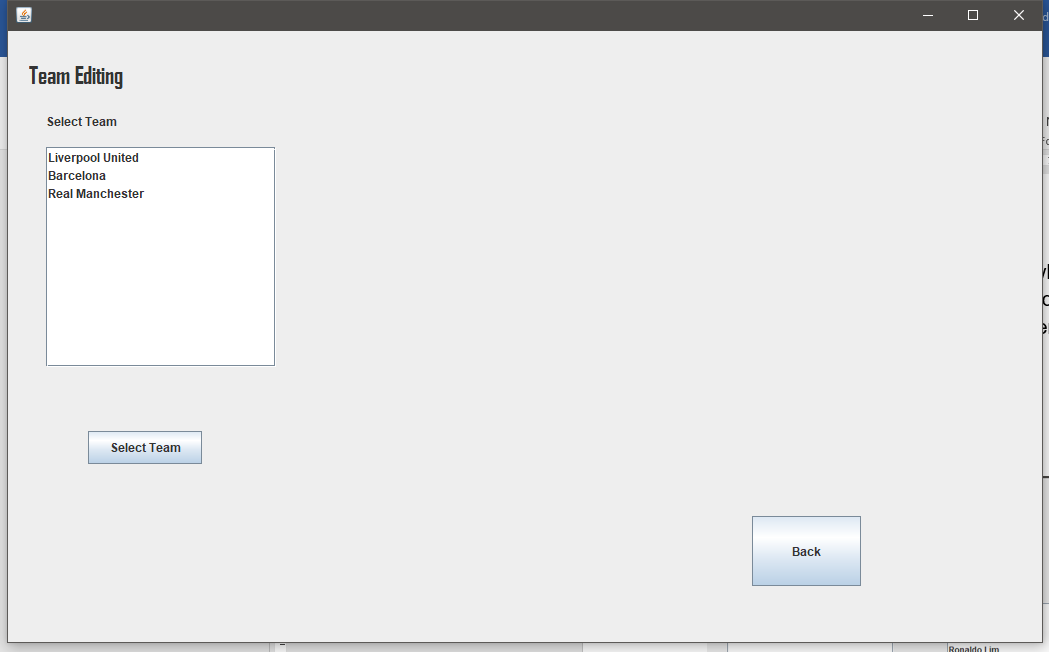
The method that does this check, is found on the Team Class:



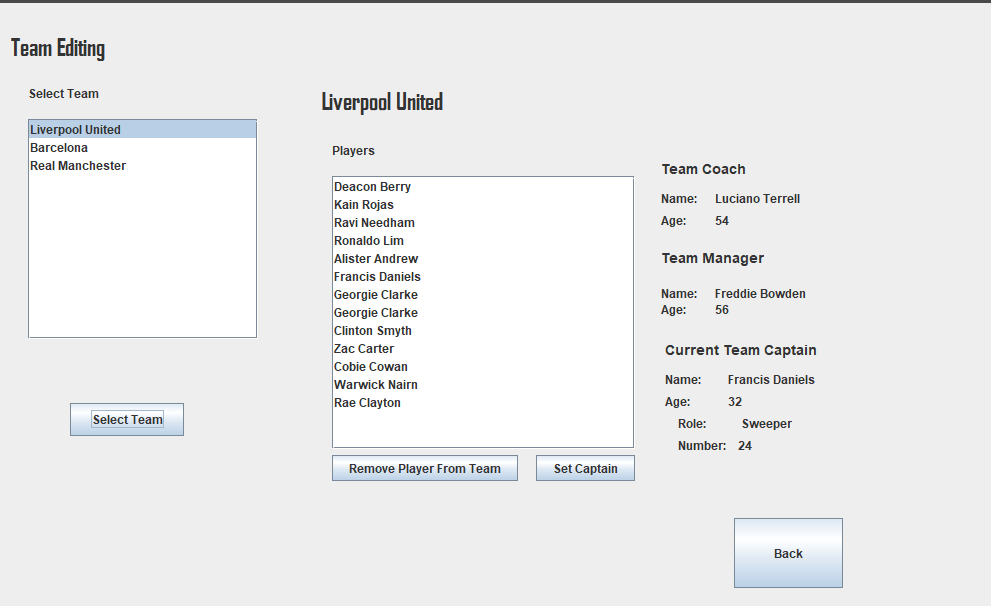
## Team Editing Menu

A Team Editing Menu was created where the user could select a team and them from that team, check its players, and remove a select player from that team, or even set that player as team captain. Other useful information is provided such as the Coach and Manager details

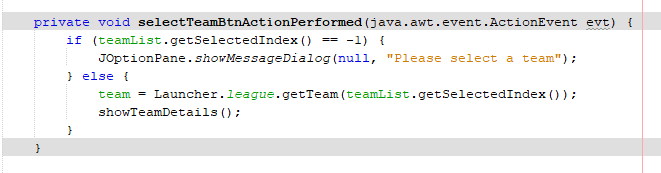
Menu before selecting the team:



After selecting the team:

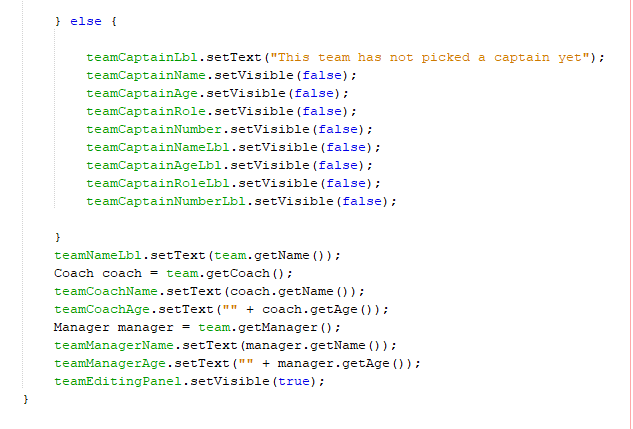


Select Team button code:



This references to a method called showTeamDetails which allows the specific team information to be displayed:





# Program and OO Paradigm

Garrido (2013) defines an object-oriented program as “an implementation of all the collections of objects that were modelled in the analysis”. This is exactly how the program was created. Each Object identified in the design stage was implemented as a class. Designing and implementing the program in this matter made it easier, because it allows the division of a big problem into smaller problems which can then be tackled individually. Instead of having to plan a complete solution from the start it is possible to tackle each sub problem at a time, creating solutions for each of them, each of them being a building block that creates the complete solution in the end.

The use of inheritance was used by making different entities such as Player, Coach, Manager and Referee all extend from a Super Class called Person so all these could access their common attributes such as name and age and behaviours. This allowed the reuse of the same code over and over again making the project more efficient and time saving.

Encapsulation was used by making the attributes of each Class as private to protect its behaviours and attributes from other objects and therefore the only way to access and manipulate such data is through the associated methods. Encapsulating the data makes the classes more secure and reliable according to B.M.Harwani (2014) which further adds that “Encapsulation also simplifies the debugging and program maintenance task.” (B.M.Harwani, 2014:205).

# Reflection

In retrospective league could have been the main method, however, I think from an OO point, and in terms of future proofing I am satisfied that I created the main class separately because in the future it would allow for different league objects at the same time, for example in case the program would need to be upgraded to be able to store different leagues, one for each year, then having the main class separately would make this upgrade a lot easier.

A lot more planning should have been undertaken before implementing the program because the design had to be constantly changed. This resulted in a program that is not really at all refined, with a lot of methods that could have been in better places.

The use of inheritance should also have been more accentuated in order to recycle specific code instead of repeating similar code throughout different entities.

# Bibliography

Garrido, J., 2003. *Object-Oriented Programming: From Problem Solving to Java* [e-book]. Charles River Media. Available from: <https://ebookcentral.proquest.com/lib/edgehill/reader.action?docID=3135658&ppg=7> [Accessed 07 December 2018]

Harwani, B.M., 2014. *Learning Object-Oriented Programming in C# 5. 0* [e-book]. CENGAGE Learning. Available from: <https://ebookcentral.proquest.com/lib/edgehill/reader.action?docID=3136712&ppg=15> [Accessed 07 December 2018]