Notes

Defensive programming has been used extensively throughout the classes, particularly with the LeagueIO, and whilst it is difficult to make a genuine mistake and break the program, it is still straight forward to cause an issues if you attempt to do so. In the input league and match methods I did individual checks for formatting errors as an example of a useful output for a user, but surrounded the whole block to check for null pointers as I ran out of time to individualise it.

‘createFixtures’ was a method I was particulary happy with, it creates the fixture list automatically when the teams are entered from a text file, and ensures no duplication of matches. Manual inputting of fixtures has been kept as an option in the LeagueIO when a new team is created as a demonstration that creating a match manually in the IO works, but to improve the program ‘createFixtures’ would be automated whenever a new team was created. A unique date is set for the match but this only advances by one day so this is something I would improve on. Rand could be used to select from possible dates and produce a realistic fixture list.

Throughout the classes ‘String force – sc.nextLine()’ is used following a ‘nextInt’ call. An error occurs as after the ‘nextInt’ the scanner is not clear, meaning the next nextLine was being skipped. Initially I used ‘String force’ to clear the scanner and allow me to continue with the project and fix at a later date, but after speaking to people it seems there are only 2 options to get around this, the way which I had done it, and the other to convert by parse. Since both required the same amount of code I decided to keep the structure the same.

Everything through the console IO is based on using a number to select teams, matches and players. I had initialised the lists to start from 1 and then subtracted to match the position in the ArrayList. I felt as an interface it looks better to do it this way, but a problem arises when inputting multiple matches from a file. When a match is completed it is immediately moved from the fixture array list to a results array list, and the remaining matches move position in the array list. If the user structures the matches in the correct order then the program runs okay as the minus is increased each time, but if they were to order the matches in the wrong way it would apply the results to the wrong team.

If an issue is found halfway through inputting a league or the results of a match, the user will be given the output message for how to fix the text file and returned to the previous menu. However the league/match will be partially created with the data the buffered reader has already processed. An option to improve the program is to remove the league/match from the holding ArrayList at this point. This was not implemented as it seemed beyond the scope, but removing from an ArrayList is demonstrated in the ‘updateFixture’ method.

I have enjoyed this project and want to thank the teaching and demonstrating staff who have helped me with this crash course in Java. Looking back through the code there are points I would do differently, I did not like to tackle bulky code initially so many methods in Match are seperate for the home and away teams where I now feel I could do both within the same methods. Overall I am happy with the outcome, and think the interface of the console is a high standard.