CT5025: Object Oriented Programming in Java

Matej Kondrot

December 24, 2019

Contents

1 Introduction

The solution implemented consists of a software which tracks football games and their results. The current implementation tracks tournaments, teams and players within it, games played and team standings. The user is able to import and export tournament save data through XML files. Recorded games can also be played back at their original speed. A hardship encountered while developing the solution was in the transition from the text-based version of the software to the current UI implementation. Another hardship encountered was proper implementation of Object-Oriented code: the foundation of the code was written with little Java experience, meaning that as the project moved along it was coded on rocky foundations. Public get() functions were used within their classes instead of private variables and there was too much reliance on static variables. Much of the code had to be systematically reworked eventually. SDLC was followed with the project being planned and implemented using Agile Methods, first starting tracking on a Microsoft Excel spreadsheet before maturing into a Kanban-style Trello board. Progress was tracked and updated periodically with commits to GitHub at https://github.com/KondrotM/footballStats

2 Design of System

2.1 UML

All UML diagrams were created using PlantUML due to its ease of use - no need to drag/drop each diagram element when adding to the diagrams.

2.1.1 Activity Diagram

The main activity diagram depicts a user's proposed activity when using the software. The diagram is split into three partitions to aimed to show what happens where. Although the partitions do not completely fulfill their purpose of showing which class does what (Fowler 2004, p. 122) - mapping each class to a partition proved too convoluted, hard to read, and went against the use of an activity diagram, it is still separated into classes which view, store, and create data.

Forks were implemented in the main diagram to show parallel processing. For example, within the Browse Tournament group, once the user selects the option to browse the tournament, a fork is created with the View Active Tournament action and a connector to the end node. This is done because when 'Browse Tournament' is selected, it is opened in a new window separate from the one used for navigation. The connector is detached from the fork join and is sent to the end node as both tokens are not needed to continue through the activity diagram. The user can start a new activity with the window open and close it whenever they like.

Within each diagram, lines were primarily split using 'splits' rather than decisions and merges. This is was done whenever a decision could be summarised

by 'whatever the user decides to do' rather than being based upon a specific condition. A decision is used for validation within the 'Play Game' group, which checks whether the home and away teams are the same or not. If they are, the user is re-prompted to make a valid team selection.

A subsidiary activity diagram shows the action of 'Begin Game.' The activity forks to allow the user to track the game while it is not finished. The flows of both secondary forks end with a merge rather than a join. This is because first flow to reach the merge will terminate the others as they are no longer needed. [UML Distilled] In the example of the 'End Game' fork, either the user manually ends the game or it is ended after a time signal input after 90 minutes. The latter fork for tracking game data will have the user score a goal, change possession or make a comment. The other flows are then terminated but the activity is looped while the game is not finished, allowing the user to track more of the game's data.

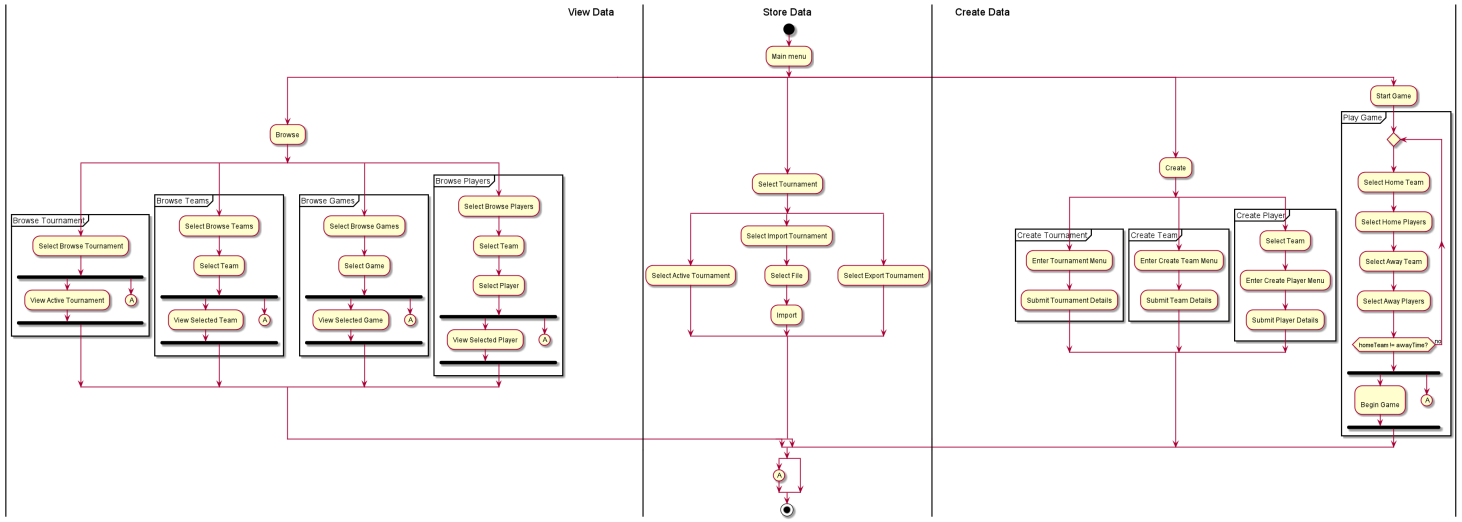

Figure 2.1.1: Main Activity Diagram
Full size can be found here
https://i.imgur.com/AxjxgpN.png

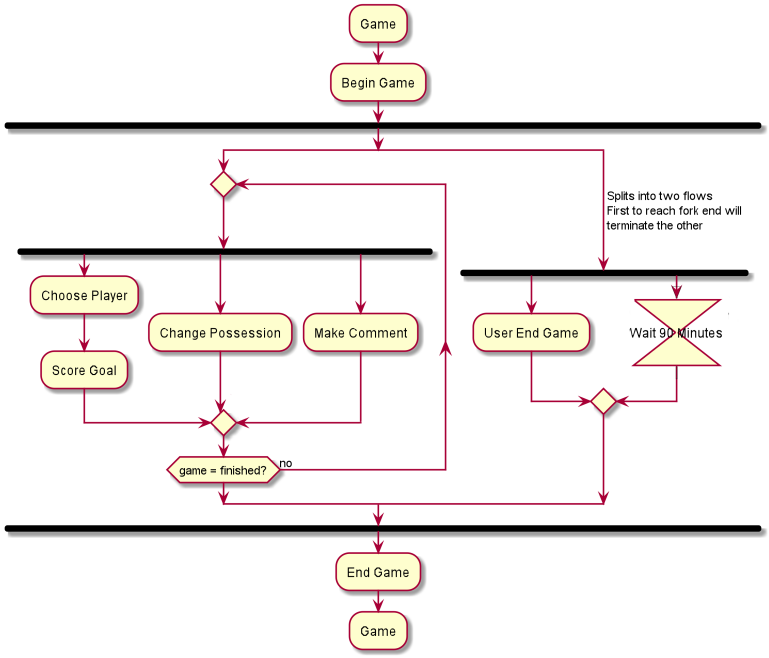

Figure 2.1.2: Subsidiary Game Activity Diagram
Full size can be found here
https://i.imgur.com/Bm4dY15.png

2.2 Class Diagram

The project has a large volume of classes, methods and members. Showing all of them on a single image of a class diagram would prove too confusing and would obfuscate the understanding of the system. Because of this, the class diagram is shown in four figures.

Figure 2.2.2 shows the relationships between the main classes of the program. These classes hold most of the data and form the heart of the program. They are used to run the program regardless of a text or graphical user interface. The diagram shows concrete and abstract classes, association verbs, dependencies and multiplicities. A bidirectional association is formed between the `Team` and `Player` Classes as the Team owns the Player, and the Player plays in the Team (Fowler 2004, p. 41). This is used in the code for adding a player to a team and being able to get the Player's Team class for a specific Player. Figure 2.2.3 shows the methods and members of each of the main classes of the program. Both were implemented with thought given to encapsulation.

Figure 2.2.4 displays how the GUI classes interact, as well as any interaction they have with the main classes of the program. Composition was used to show how the main classes affect the GUI classes, but Aggregation was omitted as it has been referred to by professionals as a 'Modelling Placebo.' (Fowler 2004, p. 67) (Rumbaugh, Jacobson, and Booch 1999) The diagram flow reflects that of the activity diagram at Figure 2.1.1, but the connections are not the same as the UI layout does not represent which classes are responsible for which windows but rather user comfort and ease of use. In Figure 2.2.5, it is visible that the class `PlayGamewindow` holds the a lot of the 'meat' of the program, as this is the class responsible for tracking the actual games being played in real time.

Type	Symbol	Drawing
Extension	<|--	
Composition	*--	
Aggregation	o--	

Character	Icon for field	Icon for method	Visibility
-			private
#			protected
~			package private
+			public

Figure 2.2.1: Class Diagram Key (PlantUML 2015)

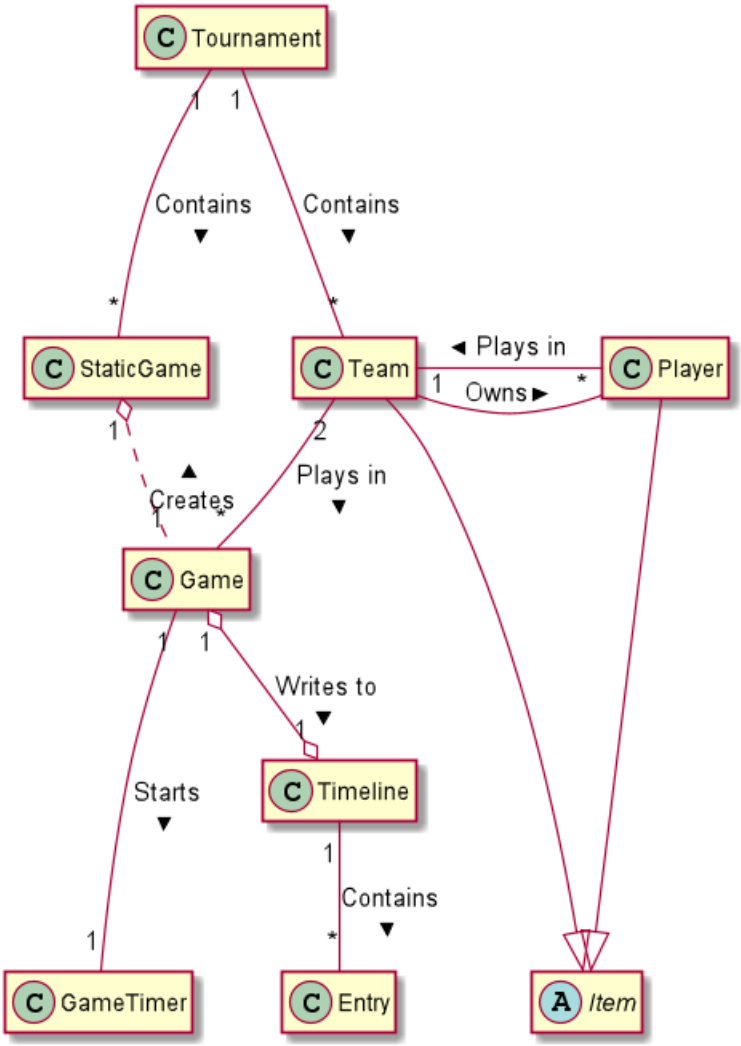

Figure 2.2.2: Main Class Diagram
Full size can be found here
https://i.imgur.com/jTJe2Ip.png

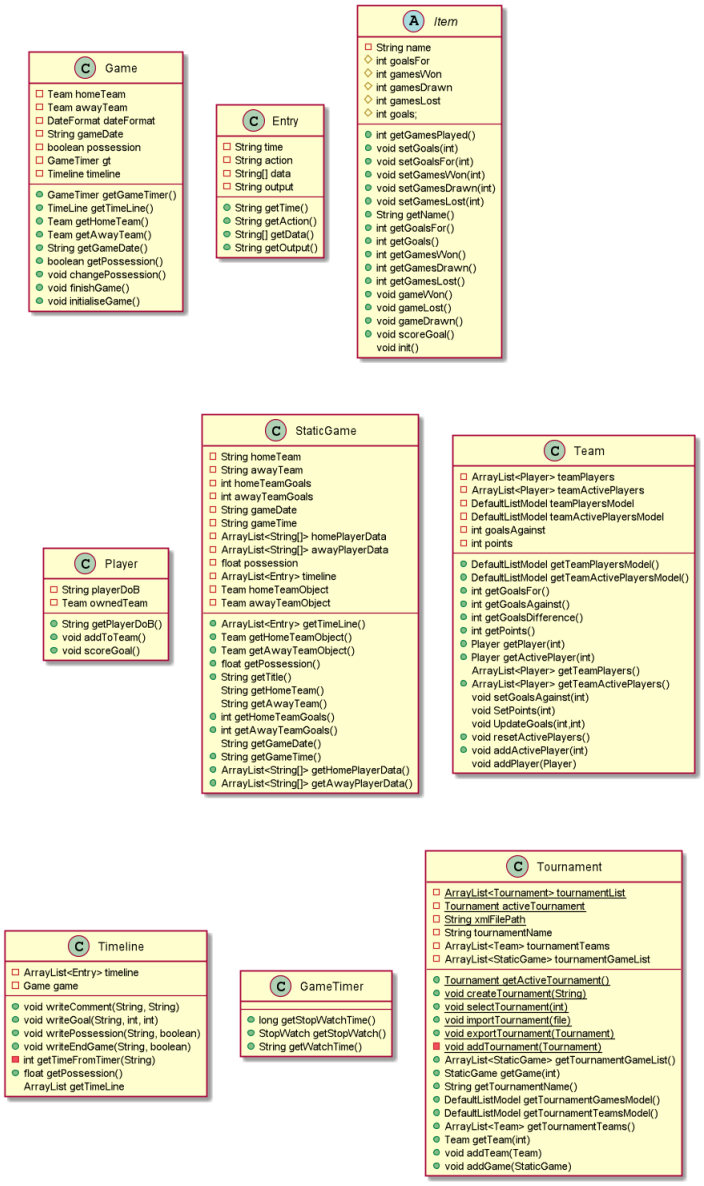

Figure 2.2.3: Methods of Main Class Diagram
Full size can be found here
https://i.imgur.com/8nOtOVE.png

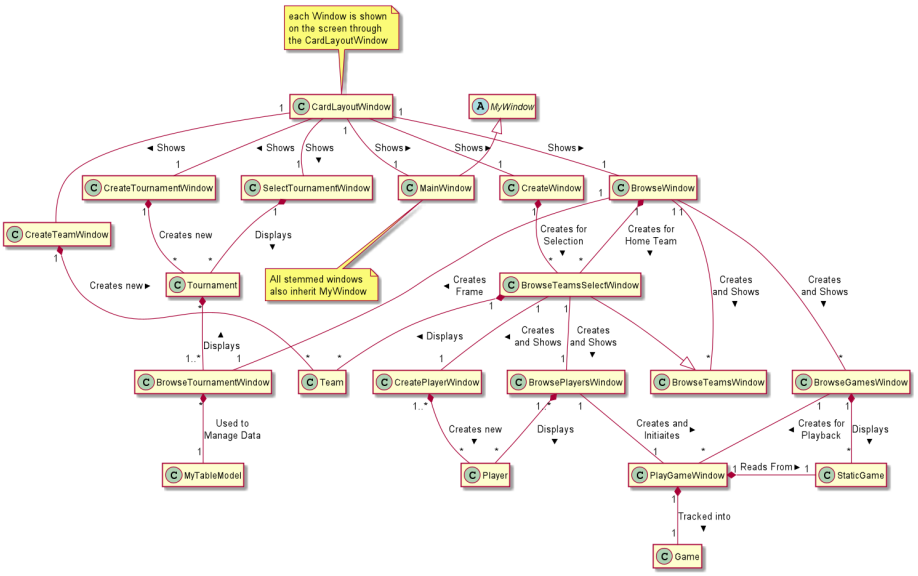

Figure 2.2.4: UI Class Diagram
Full size can be found here
https://i.imgur.com/URWYBV5.png

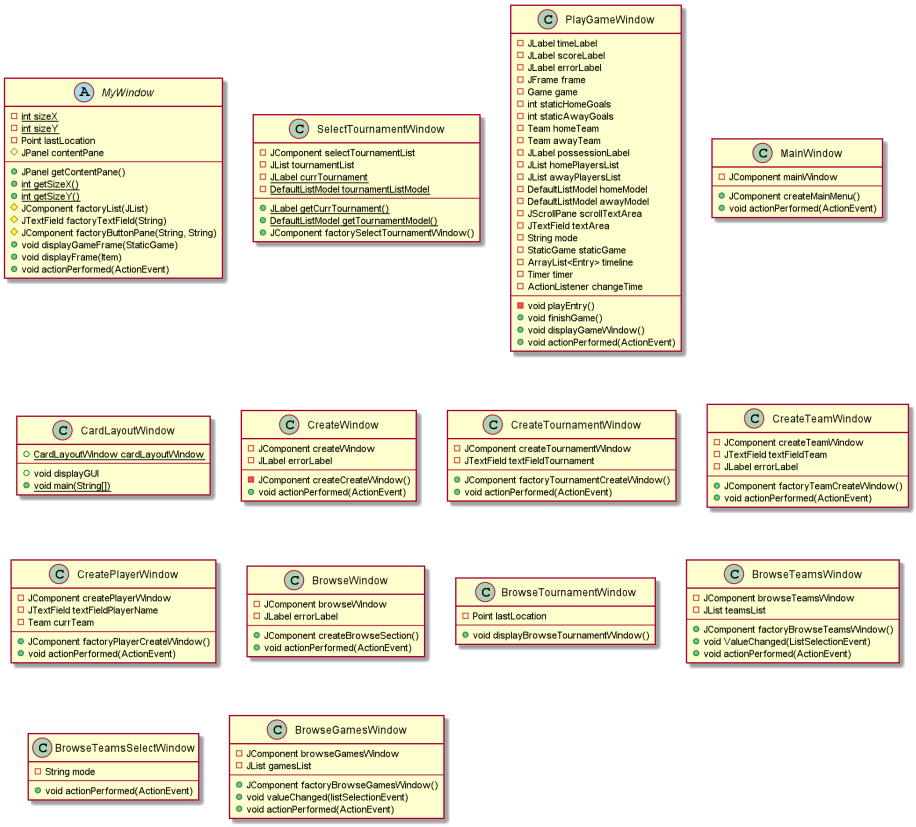

Figure 2.2.5: Methods of UI Class Diagram
Full size can be found here
https://i.imgur.com/EuoMcxH.png

2.3 Sequence Diagrams

Figure 2.3.1 tracks the process of calculating the possession of the home team within the game. The diagram shows iteration through the Timeline Event ArrayList, as well as several alt clauses.

Figure 2.3.2 shows the process of exporting a tournament into an XML file. The sequence loops through all the elements of a tournament and appends them to the xml document. The Document participant is left active throughout the sequence as each element in the code is defined through its methods - `document` is used to create XML elements. The document participant is left active throughout the 'loop' methods as the loop can be left empty if there are no teams, games, or players to append to the tournament with the XML coming out readable for the code.

Figure 2.3.3 demonstrates recursion through visualising how games are played back from a StaticGame file. Each second, an ActionListener calls the `playEntry()` method. This method checks whether the timeline isn't empty before comparing if the most recent entry's timestamp is equal to the current game time. If it is, it performs the action specified within the event, removes the event from the timeline, and then proceeds to call upon itself again. This is done so that multiple events can be played in one second by the program.

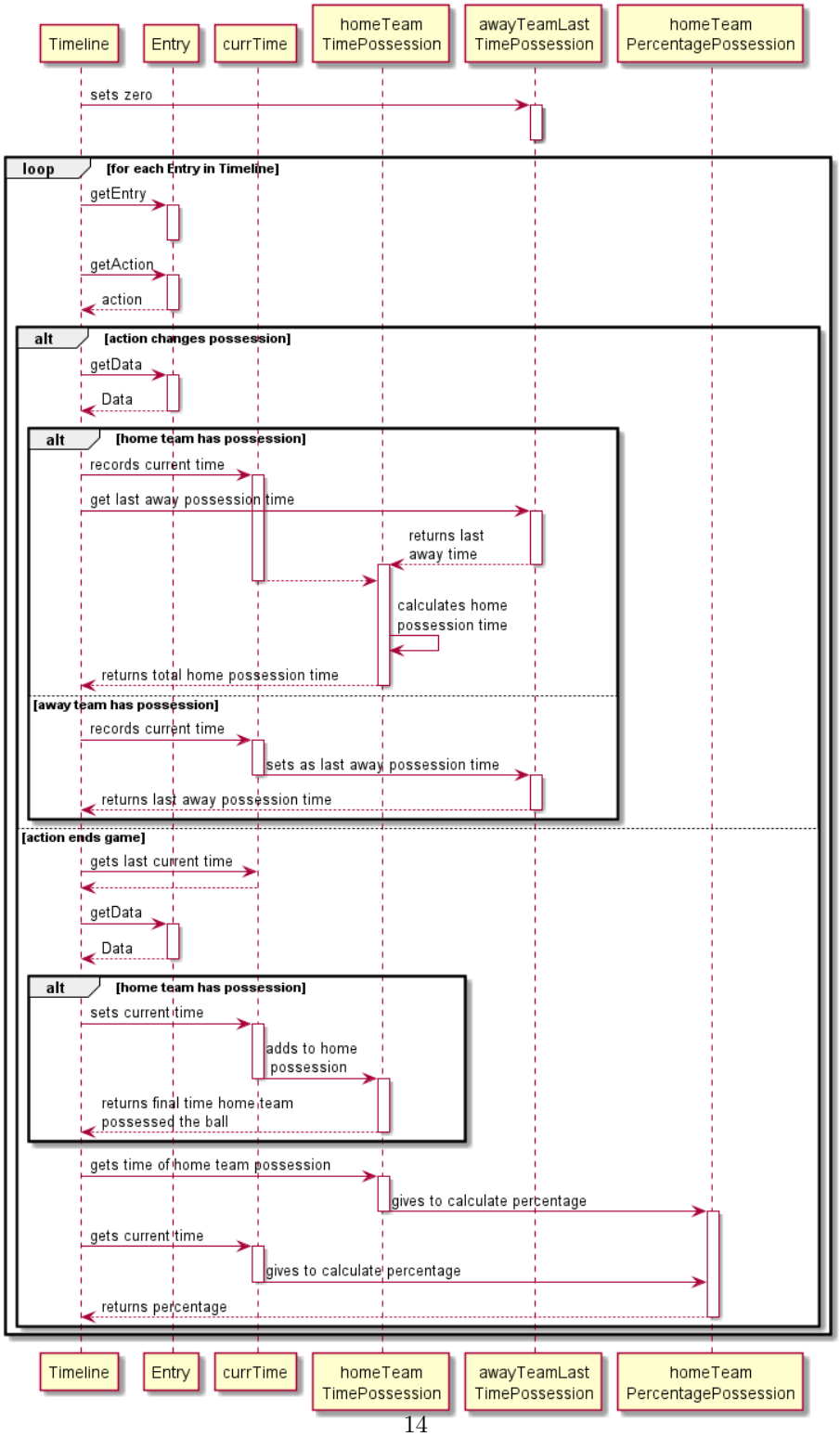

Figure 2.3.1: Sequence Diagram to Show Calculation of Possession
Full size can be found here
https://i.imgur.com/peSeTK0.png

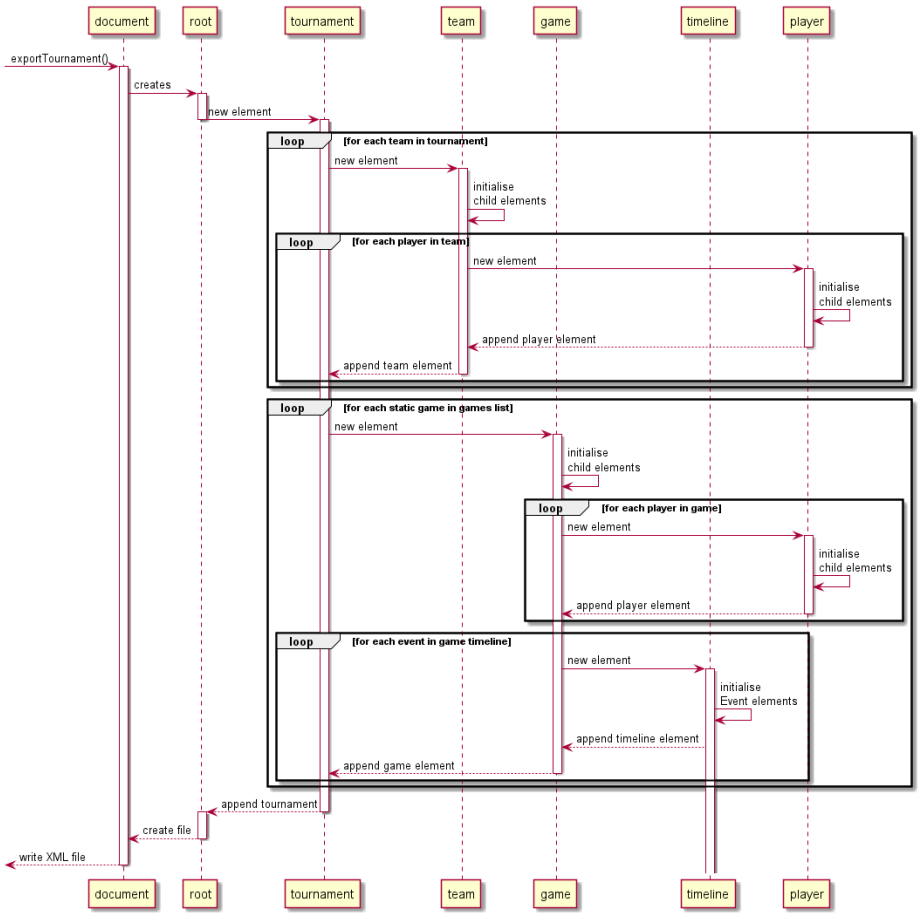

Figure 2.3.2: Sequence Diagram of Tournament Export Function
Full size can be found here
https://i.imgur.com/5WlJYcP.png

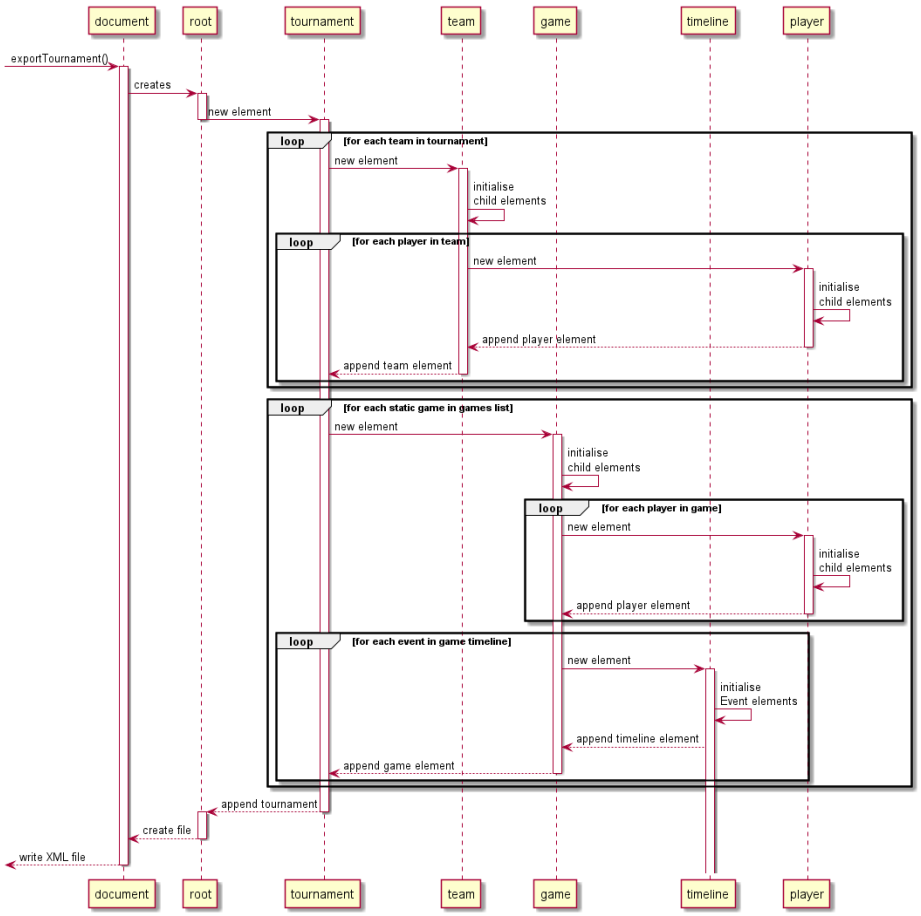

Figure 2.3.3: Sequence Diagram to Show playEntry recursion
Full size can be found here
https://i.imgur.com/7wYjCvu.png

2.4 Object-Oriented Implementation

2.4.1 Encapsulation

Encapsulation was implemented to gain simplicity and locality. It allows the program to access an internal representation of an object without having to refer to each attribute explicitly. (O'Docherty 2005, p. 20) There are no public variables declared in the code as they violate encapsulation. (Lewis and Chase 2004, p. 456) Figure 2.4.1 illustrates how encapsulation was achieved throughout the project: private variables were declared which are manipulated by the methods. Public methods interact with the private methods set, such as how the public method `getPossession()` uses the private method `getTimeFromTimer()` in order to turn the String clock time displayed into an integer for the method to interpret. The former method is more complex and needs to handle the time each team has had the ball. If the way in which the game time is tracked changes, the latter method can be amended with minor or no change to the complex method.

Figure 2.4.2 shows encapsulation in static constants. `tournamentList` and `xmlFilePath` are both static constants. Although implementing a public get method has no current practical difference to declaring the variable public (the compiler removes the getter wrapper [1]), it is still worth following the encapsulation model since, in some theoretical future version, the user could choose their export file path.

2.4.2 Method Overloading

Method overloading is a useful resource for performing similar methods on different types of data. Methods are distinguished by their signatures: the distinct number, type, or order of their parameters. (Lewis and Chase 2004, p. 458) It was implemented in Figure 2.4.4, when declaring a static game. Here, a static game is declared either from an active Game class from which its variables are then defined using `Get()` functions, or they are passed into the declaration when the games are being loaded from an XML file.

2.4.3 Polymorphism

Declaring polymorphic variables denotes the concept of saying one class type is-a different class type, where the declared class type is higher up the respective class hierarchy (O'Docherty 2005, p. 82). The concept of polymorphism was used when implementing the user interface. Buttons within JPanels were created within a `factoryButtonPane` which embedded each button within a JPanel. Since it was possible for this mode of creating to be changed at any point in the project, these buttons were declared as JComponents. Figure 2.4.3 shows the process of creating a `buttonPane`.

[1] https://stackoverflow.com/a/5922512 2019/12/19

```java
public class Timeline {

    private ArrayList<Entry> timeline;
    private Game game;

    public Timeline(Game tempGame){...}

    // method for when the user wants to write their own comment
    public void writeComment(String time, String comment){...}

    // method for when a goal is scored
    public void writeGoal(String time, int teamNo, int playerNo){...}

    // method for when the possession is changed
    public void writePossession(String time, boolean possession){...}

    // method for when the game ends
    public void writeEndGame(String time, boolean possession){...}

    // gets second value from string timer 00:00
    private int getTimeFromTimer(String timer){...}

    // calculates possession
    public float getPossession(){...}
```

Figure 2.4.1: Example of Encapsulation

2.4.4 Abstraction

An abstract class `Item` was implemented within the code. The class is used to define both `Player` and `Team`, since they share variables such as goals scored and games won. An abstract class serves to create concrete classes. No instances can be created of an abstract class. (Barnes and Kölling 2012, p. 346) Using an abstract class here increases code reuse, since both of the sub-classes can refer to superclass for variables and methods. In Figure 2.4.5, an `Item` is passed into a UI component when wanting to display its statistics. The method can be reused for both Players and Teams, and can later be expanded if, say, a Goalkeeper class was implemented which extends `Player`

Another use of abstraction is shown in Figure 2.4.6, where an abstract model is extended into a concrete class. Here, this is done to create a custom table model for showing tournament placements.

```java
public class Tournament {

    // list of tournaments declared and initialised. all tournaments, teams and players are declared in regards to the t
    private final static ArrayList<Tournament> tournamentList = new ArrayList<~>();

    // actively selected tournament
    private static Tournament activeTournament;

    private String tournamentName;
    // saves all teams within tournament
    private ArrayList<Team> tournamentTeams;
    // saves all games within tournament
    private ArrayList<StaticGame> tournamentGameList;
    // xml file path
    private final static String xmlFilePath = "./src/main/java/uk/ac/glos/ct5025/s1804317/footballStats/Tournaments/";

    // displays teams within tournament
    private DefaultListModel tournamentTeamsModel = new DefaultListModel();
```

Figure 2.4.2: Static Constants

```java
protected JComponent factoryButtonPane(String title, String command) {
    JButton button = new JButton(title);
    button.setPreferredSize(new Dimension( width: 150,  height: 25));
    button.setActionCommand(command);
    button.addActionListener( l: this);

    JPanel pane = new JPanel();
    pane.setBorder(BorderFactory.createEmptyBorder( top: 5,  left: 5,  bottom: 5,  right: 5));
    pane.add(button);

    return pane;
}
```

Figure 2.4.3: Polymorphism in a Method

```java
// loads static info from game object
public StaticGame(Game game){...}


// loads static info from XML
public StaticGame(String tempHomeTeam, String tempAwayTeam, int tempHomeTeamGoals, int tempAwayTeamGoals,
                  String tempGameDate, String tempGameTime, ArrayList<String[]> tempHomePlayerData,
                  ArrayList<String[]> tempAwayPlayerData, float tempPossession, int tempHomeTeamIndex,
                  int tempAwayTeamIndex, ArrayList<Entry> tempTimeline){...}
```

Figure 2.4.4: Overloading a Method

```java
public void displayFrame (Item item){
    JFrame frame = new JFrame(item.getName());
    frame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);

    //Set window location.
    if (lastLocation != null) {
        //Move the window over and down 40 pixels.
        lastLocation.translate( dx: 40,  dy: 40);
        if ((lastLocation.x > sizeX) || (lastLocation.y > sizeY)) {
            lastLocation.setLocation( x: 0,  y: 0);
        }
        frame.setLocation(lastLocation);
    } else {
        lastLocation = frame.getLocation();
    }
```

Figure 2.4.5: Passing an Item

```java
public class MyTableModel extends AbstractTableModel {


    // Creates a String Array of column names
    private String[] columnNames = {"Team Name",
            "GP","W","L","D","GF","GA","GD","Pts"};

    // Creates a multidimensional array, each containing a team's data.
    private ArrayList<ArrayList> data(){
        ArrayList<ArrayList> data = new ArrayList<~>();
```

Figure 2.4.6: Extending an Abstract Model

3 Discussing Logic

3.1 Inner Classes

Inner classes are useful to declare ActionEvents and other UI components within a class or method, rather than clumping them all to one series of ActionEvents at the end of the class. (Barnes and Kölling 2012, p. 380) In Figure 3.1.1 an inner class was used to declare an ActionListener which waits one second between each increment of the timer. It also calls a function to end the game when it has gone longer than 90 minutes, and attempts to play a new entry if a game is being played back rather than recorded.

```
private ActionListener changeTime = new ActionListener() {
    public void actionPerformed(ActionEvent env) {
        timeLabel.setText(game.getGameTimer().getWatchTime());
        if (mode.equals("PLAYBACK")){
            playEntry();
        }
        if( game.getGameTimer().getStopWatchTime() > 5400 ){
            finishGame();
        }
    }
};
```

Figure 3.1.1: Inner Class Used as a Timer

3.2 indexOf

When searching an ArrayList for an integer value from an object, the keyword `indexOf` is used (McCormack 2002, p. 188). Figure 3.3.1 shows the method which returns an index from a team object, while Figure 3.3.2 shows its implementation within the code. When storing teams within games into an XML file, it isn't practical to store a whole Team object within the Game - mainly since Teams within the Tournament are stored futher up the XML file. Therefore, the team's index within the `tournamentTeams` list is used as the team id. This can then later be used to `get()` the team.

3.3 Graphical User Interface

3.3.1 CardLayout

To interact with the system, the user uses the GUI. The layout manager chosen to display Panels was CardLayoutManager. This is because it allows for an easy

way to `show()` panels. Also, CardLayout only shows one component at a time. (Zukowski 1997, p. 264) Given that each menu is held in a JPanel component, this is ideal for most panels in the given scenario. Static panels which don't change components, such as `MainMenuWindow`, work well with CardLayout as they only need to be painted once. Windows such as `CreatePlayerWindow` rely on variables to be passed into them such as the current Team selected. This means that they cannot be declared upon compilation but be declared when the user selects the 'Create Player' button with a Team selected. These windows are then appended to the CardLayout, shown, and destroyed once the user navigates away from them. Figure 3.3.3 shows instances of this.

3.3.2 GridBagLayout

To place components within a JPanel, GridBagLayout was implemented. Figure 3.3.4 shows a Window constructed using GridBagLayout. The window demonstrates x/y positioning, changing insets (5px on game, 0px for text boxes/submit button) as well as horizontal and vertical fill. Figure 3.3.5 shows how the co-ordinates of the GridBagLayout were incremented in a loop in order to show an indefinite number of players in the game in the window.

```
// gets team index in regards to the teamList
private int getTeamIndex(Team team){
    ArrayList<Team> teamsList = tournamentTeams;
    int index = teamsList.indexOf(team);
    return index;
}
```

Figure 3.3.1: getTeamIndex Method

```
Element homeTeam = document.createElement( s: "hometeam");
homeTeam.setAttribute( s: "id",Integer.toString(tournament.getTeamIndex(currGame.getHomeTeamObject())));
game.appendChild(homeTeam);
```

Figure 3.3.2: Usage of getTeamIndex

```
    cardLayout.show(contentPane, command);
} else if (command.contains("MKE_TEAMS")){
    CardLayout cardLayout = (CardLayout) contentPane.getLayout();
    // Creates new window to show teams for current
    BrowseTeamsWindow browseTeamsWindow;
    if(command.contains("SELECT")){
        browseTeamsWindow = new BrowseTeamsSelectWindow(contentPane,CardLayoutWindow.cardLayoutWindow, switchMode: "BROWSE");
    } else { // if(command.contains("BROWSE")){
        browseTeamsWindow = new BrowseTeamsWindow(contentPane,CardLayoutWindow.cardLayoutWindow);
    }
    contentPane.add(browseTeamsWindow,command);
    cardLayout.show(contentPane,command);
} else if (command.contains("MKE_BROWSE_GAMES")){
    CardLayout cardLayout = (CardLayout) contentPane.getLayout();
    BrowseGamesWindow browseGamesWindow = new BrowseGamesWindow(contentPane, CardLayoutWindow.cardLayoutWindow);
    contentPane.add(browseGamesWindow,command);
    cardLayout.show(contentPane,command);
}
```

Figure 3.3.3: Declaring Windows Within the CardLayout Layout Manager

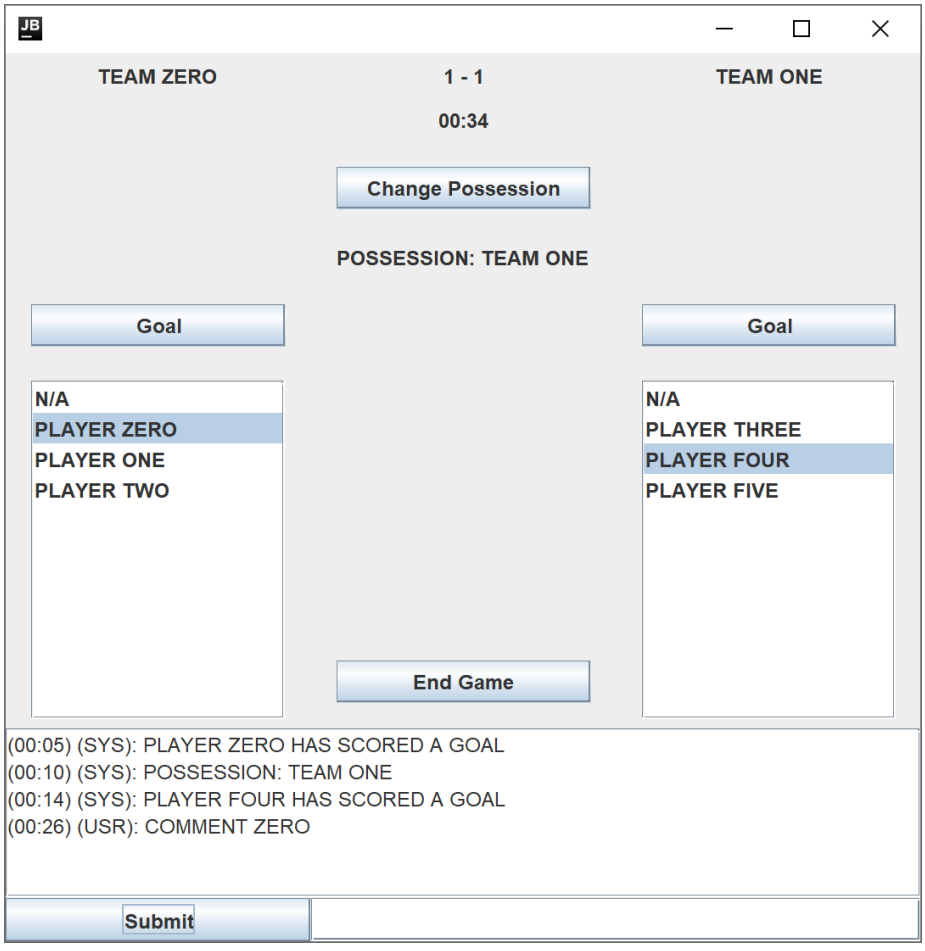

Figure 3.3.4: Game Window Constructed Using GridBagLayout

```
for (int i=0; i < game.getHomePlayerData().size(); i++){
    String[] player = game.getHomePlayerData().get(i);
    panel.add(new JLabel(player[0]),gbc);
    gbc.gridx++;
    panel.add(new JLabel(player[1]),gbc);
    gbc.gridx--;
    gbc.gridy++;
}
```

Figure 3.3.5: For Loop within GridBagLayout

4 Testing

Unit testing was done on all non-ui components within the class. When testing components, most getter/setter functions which simply return a given value were omitted unless they weren't called indirectly throughout testing. [2] (Tarlinder 2016, p. 81). Figure 4.0.1 shows the usage of JUnit `BeforeEach` and `AfterEach` methods. The former is used to declare objects which are used for either most or all tests within the test unit while the latter is used to clean up the testing process. [3]

Figure 4.0.2 shows a more complex test which involves calling multiple methods: `writePossession` methods are called to create data showing possession is being changed between the teams; `writeEndGame` is used to declare the game being over; and `calculatePossession` is used alongside the JUnit assertEquals method to show that the possession is correctly calculated. Because the value returned is a float, a delta has to be declared within the assertEquals call to allow for margin of error.

Figure 4.0.3 shows a try-catch method within a unit test. The game timer is tested by starting the timer, waiting three seconds, and then seeing whether the time is equal to the time waited. This proccess can be interrupted by a `InterruptedException`. If this happens, the exception is caught, the test is failed, and the stack trace is printed.

Figure 4.0.4 refers to a potential problem while unit-testing the code. Certain methods were unable to be tested on their own as they were non-public methods, and it was better to test them implicitly through public methods rather than declaring them public for the sake of testing.

[2] http://developertesting.rocks/common-questions/should-i-test-getters-and-setters/, 2019-12-22

[3] https://examples.javacodegeeks.com/core-java/junit/junit-setup-teardown-example/, 2019-12-22

```java
@org.junit.jupiter.api.BeforeEach
void setUp(){
    team = new Team( tempName: "Hull");
    player = new Player( name: "Harry", tempDoB: "19/02/1993",team);
    tournament = new Tournament( tempName: "League Tournament");
    player.addToTeam();

}

@org.junit.jupiter.api.AfterEach
void tearDown() {

}
```

Figure 4.0.1: Before-each and Tear-down Within Test Suite

```java
29      @Test
30      void possessionWithAwayEnd(){
31          // adds timeline events
32          timeline.writePossession( time: "00:00",  possession: true);
33          timeline.writePossession( time: "00:10",  possession: false);
34          timeline.writePossession( time: "00:25",  possession: true);
35          timeline.writePossession( time: "00:30",  possession: false);
36          // ends game
37          timeline.writeEndGame( time: "00:40", possession: false);
38          // 15/40 == 0.375
39          assertEquals( expected: 0.375f,timeline.calculatePossession(), delta: 0.0002);
40      }
41
```

Figure 4.0.2: Adding multiple values to be tested within a complex calculation

```java
    @Test
    void testGameTimer() {
        gameTimer = new GameTimer();
        try {
            TimeUnit.SECONDS.sleep( timeout: 3);
            assertEquals( expected: 3, gameTimer.getStopWatchTime(),  delta: 0.1);
        } catch (InterruptedException e){
            fail("Test Interrupted");
            e.printStackTrace();
        }
    }
```

Figure 4.0.3: Using a Try-Catch statement in a test suite

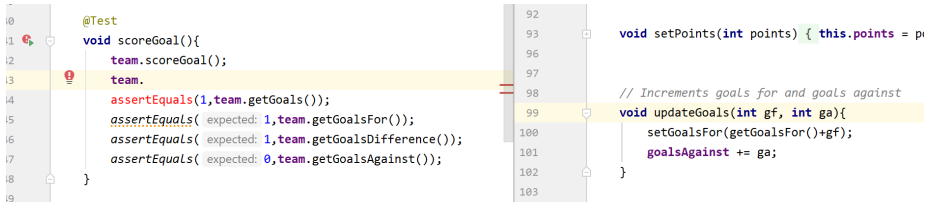

Figure 4.0.4: Problems encountered during testing

5 Conclusion

Creating this software was a learning experience. In the future, self-governance should be greater priority. Too much reliance has been placed within accessor functions, which violate encapsulation. (Lewis and Chase 2004, p. 455) (Holub 2003) Object Oriented systems should have a focus on data abstraction: hiding the way in which an object implements a message handler from the rest of the program. (Holub 2003) The variable `tournamentList` could have been implemented as a Singleton class as only one of these is needed and it must be present throughout the code. Inner classes and Anonymous inner classes should have been used when implementing menu buttons, as they allow for less storing of variables, improved cohesion, and a better class specialisation for particular tasks. (Barnes and Kölling 2012, pp. 378–380) Integration testing should have been implemented to display the way in which classes work together. (Wu 2010, p. 25) Any non-academic references were segmented to footnotes in the document to allow for purely academic references. Overall, the project did well to implement functionality shown in the module, as well as expanding on Java implementation through academic research.

References

Barnes, David J and Kölling, Michael (2012). *Objects first with Java : a practical introduction using BlueJ*. Pearson. ISBN: 9780132492669.

Fowler, Martin (2004). *UML Distilled: a Brief Guide to the Standard Object Modeling Language*. Addison-Wesley.

Holub, Allen (2003). *Why getter and setter methods are evil*. URL: `https://www.javaworld.com/article/2073723/why-getter-and-setter-methods-are-evil.html` (visited on 12/17/2019).

Lewis, John and Chase, Joseph (2004). *Java software structures : designing and using data structures*. Addison-Wesley. ISBN: 0321225317.

McCormack, Colin (2002). *Java: getting down to business*. Basingstoke. ISBN: 0333791851.

O'Docherty, Mike (2005). *Object-Oriented Analysis and Design: Understanding System Development with UML*. Wiley. ISBN: 0470092408.

PlantUML (2015). *PlantUML Documentation*. URL: `https://plantuml.com/class-diagram` (visited on 12/22/2019).

Rumbaugh, James, Jacobson, Ivar, and Booch, Grady (1999). *The unified modeling language reference manual*. Addison-Wesley. ISBN: 020130998x.

Tarlinder, Alexander (2016). *Developer Testing: Building Quality into Software*. Addison-Wesley. ISBN: 0134291069.

Wu, C.Thomas (2010). *An Introduction to Object-Oriented Programming with Java*. McGraw-Hill. ISBN: 9780073523309.

Zukowski, John (1997). *Java AWT Reference*. O'Reilly Media. ISBN: 9781565922402.

6 Appendix 1

- The JavaDoc zip is located under ./footBallStats/javadoc
- The .jar file for the software is located under ./footBallStats/classes/artifacts/footballstats_jar
- The user can load a tournament XML file from ./footBallStats/Tournaments
- Within the software, the user is able to
  - Create a tournament (Create > Tournament)
  - Create a team (Create > Team)
  - Create a player (Create > Player)
  - Play a game (Start Game)
  - View tournament stats (Browse > Tournament)
  - View team stats (Browse > Teams)
  - View player stats (Browse > Players)
  - View game stats (Browse > Games ¿ Select)
  - Replay a game (Browse > Games > Play game)
  - Import a tournament (Select tournament > Import tournament)
  - Export a tournament (Select tournament > Export tournament)
  - Select active tournament (Select tournament > Select)

7 Appendix 2

Java Code

7.1 Unit Classes

7.1.1 Entry.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

public class Entry {

    private String time;
    private String action;
    // String[] as multiple data needs to be sent
       sometimes
    private String[] data;
    private String output;

    /**
```

```
     * Entry class used to write entries within the
       Timeline
     * @param time time of entry
     * @param action action performed
     * @param data data stores
     * @param output output shown to user
     */
    public Entry(String time, String action,
    String[] data, String output) {
        this.time = time;
        this.action = action;
        this.data = data;
        this.output = output;
    }

    public String getTime() {
        return time;
    }

    public String getAction() {
        return action;
    }

    public String[] getData() {
        return data;
    }

    public String getOutput() {
        return output;
    }
}
```

7.1.2 Game.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Game {
    // defines home and away team objects
    private Team homeTeam;
    private Team awayTeam;
    // game date based on date format
```

```
private DateFormat dateFormat;
private String gameDate;
// tracks which team has possession of the ball
private boolean possession;
// increments time each second
private GameTimer gt;
// documents each entry in the game
private Timeline timeLine;

/**
 * Game which is played between home and away team
 * @param tempHomeTeam home team object selected from
     UI
 * @param tempAwayTeam away team object selected from
     UI
 */
public Game (Team tempHomeTeam, Team tempAwayTeam){
    homeTeam = tempHomeTeam;
    awayTeam = tempAwayTeam;
    possession = true;
    gt = new GameTimer();
    dateFormat = new SimpleDateFormat("yyyy/MM/dd_HH:
        mm:ss");

    // initialises a new timeline and passes this
        game as a parameter
    timeLine = new Timeline(this);

    // sets gameDate to be current date.
    // written as a string but ordered as Y/M/D H:M:S
         so that it can still be ordered
    gameDate = dateFormat.format(new Date());
}

public GameTimer getGameTimer(){
    return gt;
}

public Timeline getTimeLine(){
    return timeLine;
}

Team getHomeTeam(){ return homeTeam; }

Team getAwayTeam(){
    return awayTeam;
```

```
}

String getGameDate(){
    return gameDate;
}

public boolean getPossession(){
    return possession;
}

public void changePossession(){
    // inverses which team has possession
    possession = !possession;
}

public void finishGame(){
    // compares win/lose/draw
    if (homeTeam.getGoals() > awayTeam.getGoals()){
        homeTeam.gameWon();
        awayTeam.gameLost();
    } else if (awayTeam.getGoals() > homeTeam.
        getGoals()) {
        awayTeam.gameWon();
        homeTeam.gameLost();
    } else {
        awayTeam.gameDrawn();
        homeTeam.gameDrawn();
    }
    // sets each team's goals for and goals against
    homeTeam.updateGoals(homeTeam.getGoals(),awayTeam
        .getGoals());
    awayTeam.updateGoals(awayTeam.getGoals(),
        homeTeam.getGoals());

    // wipes active players
    homeTeam.resetActivePlayers();
    awayTeam.resetActivePlayers();
}

public void initialiseGame(){
    // Flushes out local player data
    Player player;
    homeTeam.init();
    for (int i = 0; i < homeTeam.getActivePlayers().
        size(); i++){
        player = homeTeam.getActivePlayers().get(i);
```

```
            player.init();
        }
        awayTeam.init();
        for (int i = 0; i < awayTeam.getActivePlayers().
            size(); i++){
            player = awayTeam.getActivePlayers().get(i);
            player.init();
        }

    }

}
```

7.1.3 GameTimer.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

import org.apache.commons.lang3.time.StopWatch;

import java.util.concurrent.TimeUnit;

public class GameTimer {
    StopWatch stopWatch = new StopWatch();

    /**
     * Declares and begins game timer
     */
    public GameTimer(){
        stopWatch.start();
    }

    public long getStopWatchTime() {
        return stopWatch.getTime(TimeUnit.SECONDS);
    }

    public StopWatch getStopWatch(){
        return stopWatch;
    }

    public String getWatchTime(){
        long time = stopWatch.getTime(TimeUnit.SECONDS);
        long mintime = time/60;
        long sectime = time%60;
        mintime = Math.round(mintime);
        String mintimeStr = Long.toString(mintime);
```

```
        String sectimeStr = Long.toString(sectime);
        if (mintimeStr.length()==1){
            mintimeStr = "0"+mintimeStr;
        } if (sectimeStr.length()==1){
            sectimeStr = "0"+sectimeStr;
        }


        String outputTime = mintimeStr + ":" + sectimeStr
            ;

        return outputTime;
    }
}
```

7.1.4 Item.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

import java.util.ArrayList;

// Item is used to extend team and player classes
public abstract class Item {
    private String name;

    // item stats
    private int goalsFor;
    private int gamesWon;
    private int gamesDrawn;
    private int gamesLost;
    // goals refers to goals local to a game while
       goalsFor carries over for the whole tournament
    private int goals;

    /**
     * Abstract item class used to define Player and Team
          classes
     * @param tempName name given to the instance of the
         class
     */
    public Item(String tempName) {
        name = tempName;
        goals = 0;
        gamesWon = 0;
        gamesDrawn = 0;
```

```
        gamesLost = 0;
    }

    public void incrementGoalsFor(){
        goalsFor++;
    }

    public void incrementGamesWon(){
        gamesWon++;
    }

    public void incrementGamesLost(){
        gamesLost++;
    }

    public void incrementGamesDrawn(){
        gamesDrawn++;
    }

    public void incrementGoals(){
        goals++;
    }

    public int getGamesPlayed(){
        // sums all games played
        int gamesPlayed = gamesWon + gamesDrawn +
            gamesLost;
        return gamesPlayed;
    }


    // setters
    public void setGoals(int goals) {
        this.goals = goals;
    }

    // package-private as it is only used by classes
        which inherit this

    void setGoalsFor(int goalsFor) {
        this.goalsFor = goalsFor;
    }

    void setGamesWon(int gamesWon) {
        this.gamesWon = gamesWon;
    }
```

```
void setGamesDrawn(int gamesDrawn) {
    this.gamesDrawn = gamesDrawn;
}

void setGamesLost(int gamesLost) {
    this.gamesLost = gamesLost;
}

// getters

public String getName() { return name; }

public int getGoalsFor() {
    return goalsFor;
}

public int getGoals() { return goals; }

public int getGamesWon(){
    return gamesWon;
}

public int getGamesDrawn() {
    return gamesDrawn;
}

public int getGamesLost() {
    return gamesLost;
}

// increments variables
public void gameWon() {
    gamesWon++;
}

public void gameDrawn(){
    gamesDrawn++;
}

public void gameLost(){
    gamesLost++;
}

public void scoreGoal() {
    goals++;
```

```
    }

    // clears local goals
    void init(){
        goals = 0;
    }

}
```

7.1.5 Player.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

import java.util.ArrayList;

public class Player extends Item {

    private String playerDoB;
    // shows which team player belongs to
    private Team ownedTeam;

    /**
     *  Creates a Player object which can then be used to
        assign new players
     * @param name name super'd into Item parent class
     * @param tempDoB player date of birth
     * @param team Team class to which the player belongs
        to
     */
    public Player(String name, String tempDoB, Team team)
        {
        super(name);
        playerDoB = tempDoB;
        ownedTeam = team;
    }

    public void addToTeam(){
        ownedTeam.addPlayer(this);
    }

    public String getPlayerDoB(){
        return playerDoB;
    }
```

```
    // Overrides item method to also score for the player
        's team
    @Override
    public void scoreGoal(){
        ownedTeam.scoreGoal();
        incrementGoals();
        incrementGoalsFor();
    }

    }


package uk.ac.glos.ct5025.s1804317.footballStats;

import java.util.ArrayList;

// StaticGame is a static instance of a game, which can
    be used for storing local games.
public class StaticGame {
    // names of teams
    private String homeTeam;
    private String awayTeam;
    // team goals
    private int homeTeamGoals;
    private int awayTeamGoals;
    // date of game and how long it lasted
    private String gameDate;
    private String gameTime;
    // players and their respective goals scored
    private ArrayList<String[]> homePlayerData;
    private ArrayList<String[]> awayPlayerData;
    // percentage of home team possession
    private float possession;
    // timeline of entries
    private ArrayList<Entry> timeline;
    // team objects for playback
    private Team homeTeamObject;
    private Team awayTeamObject;

    /**
     * loads static info from game object
     * @param game game which has just been played,
         contains all necessary staticgame variables
     */
    public StaticGame(Game game){
        homeTeamObject = game.getHomeTeam();
```

```
        awayTeamObject = game.getAwayTeam();

        homeTeam = game.getHomeTeam().getName();
        awayTeam = game.getAwayTeam().getName();
        homeTeamGoals = game.getHomeTeam().getGoals();
        awayTeamGoals = game.getHomeTeam().getGoals();
        gameDate = game.getGameDate();
        gameTime = game.getGameTimer().getWatchTime();
        possession = game.getTimeLine().
            calculatePossession();
        timeline = game.getTimeLine().getTimeline();
        // initialises new arraylists and appends player
            data
        // each player's data is a String list, imitating
             a tuple
        homePlayerData = new ArrayList<String[]>();
        awayPlayerData = new ArrayList<String[]>();
        for (Player player : game.getHomeTeam().
            getActivePlayers()){
            String[] playerTuple = new String[]{player.
                getName(),Integer.toString(player.getGoals
                ())};
            homePlayerData.add(playerTuple);
        }
        for (Player player : game.getAwayTeam().
            getActivePlayers()){
            String[] playerTuple = new String[]{player.
                getName(),Integer.toString(player.getGoals
                ())};
            awayPlayerData.add(playerTuple);
        }

    }



    /**
     * loads static info from XML
     * @param tempHomeTeam home team name
     * @param tempAwayTeam away team name
     * @param tempHomeTeamGoals home team goals integer
     * @param tempAwayTeamGoals away team goals integer
     * @param tempGameDate date of the game
     * @param tempGameTime time taken for the game to be
         played
     * @param tempHomePlayerData data with players in the
```

```
     home team and how many goals they have scored
 * @param tempAwayPlayerData data with players in the
     away team and how many goals they have scored
 * @param tempPossession possession value
 * @param tempHomeTeamIndex team index to get team
     object
 * @param tempAwayTeamIndex team index to get team
     object
 * @param tempTimeline timeline of events which
     occurred in the game
 */
public StaticGame(String tempHomeTeam, String
    tempAwayTeam, int tempHomeTeamGoals, int
    tempAwayTeamGoals,
                    String tempGameDate, String
                        tempGameTime, ArrayList<String
                        []> tempHomePlayerData,
                    ArrayList<String[]>
                        tempAwayPlayerData, float
                        tempPossession, int
                        tempHomeTeamIndex,
                    int tempAwayTeamIndex, ArrayList<
                        Entry> tempTimeline){
    homeTeam = tempHomeTeam;
    awayTeam = tempAwayTeam;
    // gets team object from team list index
    homeTeamObject = Tournament.getActiveTournament()
        .getTournamentTeams().get(tempHomeTeamIndex);
    awayTeamObject = Tournament.getActiveTournament()
        .getTournamentTeams().get(tempAwayTeamIndex);
    homeTeamGoals = tempHomeTeamGoals;
    awayTeamGoals = tempAwayTeamGoals;
    gameDate = tempGameDate;
    gameTime = tempGameTime;
    homePlayerData = tempHomePlayerData;
    awayPlayerData =  tempAwayPlayerData;
    possession = tempPossession;
    timeline = tempTimeline;
}

// getters

public ArrayList<Entry> getTimeline(){
    return timeline;
}
```

```
public Team getHomeTeamObject(){
    return homeTeamObject;
}

public Team getAwayTeamObject(){
    return awayTeamObject;
}

public float getPossession(){
    return possession;
}

public String getTitle(){
    return gameDate + " - " + homeTeam + " vs " +
        awayTeam;
}

public String getHomeTeam() {
    return homeTeam;
}

public String getAwayTeam() {
    return awayTeam;
}

public int getHomeTeamGoals() {
    return homeTeamGoals;
}

public int getAwayTeamGoals() {
    return awayTeamGoals;
}

public String getGameDate() {
    return gameDate;
}

public String getGameTime() {
    return gameTime;
}

public ArrayList<String[]> getHomePlayerData() {
    return homePlayerData;
}

public ArrayList<String[]> getAwayPlayerData() {
```

```
        return awayPlayerData;
    }
}
```

7.1.6 Team.java

```
//Add playerID so that there aren't duplicates

package uk.ac.glos.ct5025.s1804317.footballStats;

import uk.ac.glos.ct5025.s1804317.footballStats.UI.Create
    .CreateTeamWindow;

import javax.swing.*;
import java.util.ArrayList;

public class Team extends Item {
    // players on the team
    private ArrayList<Player> teamPlayers;
    // players in a specified game
    private ArrayList<Player> activePlayers;

    // Holds all players to show within UI
    private DefaultListModel teamPlayersModel;
    private DefaultListModel teamActivePlayersModel;

    // goals against and points are stored separate from
       Item, as they are team-specific and not player
    private int goalsAgainst;
    private int points;

    /**
     * Team which is declared within a Tournament and
        holds Player objects
     * @param tempName team name super'd to Item parent
        class
     */
    public Team(String tempName) {
        // sends name to Item constructor
        super(tempName);

        // initialises list model
        teamPlayersModel = new DefaultListModel();
        teamActivePlayersModel = new DefaultListModel();
```

```
        //declares an Array to store the team's players
        teamPlayers = new ArrayList();
        // declares an Array to store active players for
            future games
        activePlayers = new ArrayList();

        setGoalsFor(0);
        goalsAgainst = 0;
        points = 0;
    }

    public DefaultListModel getTeamPlayersModel() {
        return teamPlayersModel;
    }

    public DefaultListModel getTeamActivePlayersModel() {
        return teamActivePlayersModel;
    }

    // getters

    public int getGoalsAgainst() {
        return goalsAgainst;
    }

    public int getGoalsDifference() {
        int goalsDifference = getGoalsFor() -
            goalsAgainst;
        return goalsDifference;
    }

    public int getPoints() {
        return points;
    }

    // gets player from playerList
    public Player getPlayer(int playerNo){
        Player currPlayer = (Player) teamPlayers.get(
            playerNo);
        return currPlayer;
    }

    public Player getActivePlayer(int playerNo){
        Player currPlayer = (Player) activePlayers.get(
            playerNo);
        return currPlayer;
```

```
}

// Returns array of team player names
public ArrayList<Player> getTeamPlayers() {
    return teamPlayers;
}

public ArrayList<Player> getActivePlayers(){
    return activePlayers;
}


// setters

void setGoalsAgainst(int goalsAgainst) {
    this.goalsAgainst = goalsAgainst;
}

void setPoints(int points) {
    this.points = points;
}


// Increments goals for and goals against
void updateGoals(int gf, int ga){
    setGoalsFor(getGoalsFor()+gf);
    goalsAgainst += ga;
}

// clears active players list from local game
public void resetActivePlayers(){
    activePlayers = new ArrayList();
}

// add active player to activePlayers
public void addActivePlayer(int playerNo){
    Player player = getPlayer(playerNo);
    activePlayers.add(player);
}

// adds player to team
void addPlayer(Player player) {
    teamPlayers.add(player);
    // updates teamplayersmodel
    teamPlayersModel.add(teamPlayersModel.getSize(),
        player.getName());
```

```
    }

    // Override functions as team wins need to refer
       increment points also
    @Override
    public void gameWon(){
        incrementGamesWon();
        points += 3;
        Player currPlayer;
        for (int i = 0; i < getActivePlayers().size(); i
           ++){
            currPlayer = (Player) getActivePlayers().get(
               i);
            // calls Item gameWon function
            currPlayer.gameWon();
        }
    }

    @Override
    public void gameDrawn(){
        incrementGamesDrawn();
        points ++;
        Player currPlayer;
        for (int i = 0; i < getActivePlayers().size(); i
           ++){
            currPlayer = (Player) getActivePlayers().get(
               i);
            currPlayer.gameDrawn();
        }
    }

    @Override
    public void gameLost(){
        incrementGamesLost();
        Player currPlayer;
        for (int i = 0; i < getActivePlayers().size(); i
           ++){
            currPlayer = (Player) getActivePlayers().get(
               i);
            currPlayer.gameLost();
        }
    }

}
```

7.1.7 Timeline.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

import java.util.ArrayList;

public class Timeline {

    private ArrayList<Entry> timeline;
    private Game game;

    /**
     * timeline which tracks the details of a game
     * @param tempGame game to which timeline is attached
     */
    public Timeline(Game tempGame){
        timeline = new ArrayList<Entry>();
        game = tempGame;
    }

    // method for when the user wants to write their own
        comment
    public void writeComment(String time, String comment)
        {
        String output = comment.toUpperCase();
        String action = "COMMENT";
        // creates new entry with no data, only player
            input as output
        Entry entry = new Entry(time,action,new String[]{
            ""},output);
        timeline.add(entry);
    }

    // method for when a goal is scored
    public void writeGoal(String time, int teamNo, int
        playerNo){
        int[] data = new int[]{teamNo,playerNo};
        String action = "SCR_GOAL";
        String output;

        if (teamNo == 0) {
            if (playerNo == -1) {
                // Gets team name if no player scored the
                    goal
                output = game.getHomeTeam().getName() + "
                    _SCORED_A_GOAL!";
```

```
            }
            else {
                output = game.getHomeTeam().
                    getActivePlayer(playerNo).getName() +
                    " SCORED A GOAL!";
            }
        } else if (teamNo == 1) {
            if (playerNo == -1) {
                // Gets team name if no player scored the
                    goal
                output = game.getAwayTeam().getName() + "
                    SCORED A GOAL!";
            }
            else {
                output = game.getAwayTeam().
                    getActivePlayer(playerNo).getName() +
                    " SCORED A GOAL!";
            }
        } else {
            // there are only two teams, 0 or 1. if the
                user manages to break this somehow then
                the code keeps running but lets the user
                know something is wrong
            output = "INCORRECT TIMELINE PARSE";
        }

        Integer.toString(data[0]);

        // saves team number (0 home / 1 away) and player
            index
        String[] sData = new String[]{Integer.toString(
            data[0]),Integer.toString(data[1])};

        Entry entry = new Entry(time,action,sData,output)
            ;

        timeline.add(entry);
    }

    // method for when the possession is changed
    public void writePossession(String time, boolean
        possession){
        String action = "CGE_POSSESSION";
        String output;
        if (possession){
```

```
            output = game.getHomeTeam().getName() + " HAS
                POSSESSION";
        } else {
            output = game.getAwayTeam().getName() + " HAS
                POSSESSION";
        }

        // saves boolean possession
        String[] strPossession = new String[]{Boolean.
            toString(possession)};

        Entry entry = new Entry(time,action,strPossession
            ,output);

        timeline.add(entry);
    }

    // method for when the game ends
    public void writeEndGame(String time, boolean
        possession){
        String action = "END_GAME";
        String output = "GAME_OVER";

        Entry entry = new Entry(time,action,new String[]{
            Boolean.toString(possession)},output);

        timeline.add(entry);
    }

    // gets second value from string timer 00:00
    private int getTimeFromTimer(String timer){
        int minutes = Integer.parseInt(timer.substring
            (0,2));
        int seconds = Integer.parseInt(timer.substring
            (3,5));
        int time = (minutes*60)+seconds;
        return time;
    }

    // calculates possession
    public float calculatePossession(){
        float homeTeamTimePossession = 0;
        float awayTeamLastTimePossession = 0;
        float homeTeamPossession = 0;
        float currTime = 0;
```

```
    for (int i = 0; i < timeline.size(); i++) {
        Entry entry = timeline.get(i);
        if (entry.getAction().equals("CGE_POSSESSION"
            )){
            if(!Boolean.valueOf(entry.getData()[0])){
                // calculates time the home team has
                    had the ball based from
                // the current time and last time the
                     enemy team had possession
                currTime = getTimeFromTimer(entry.
                    getTime());
                homeTeamTimePossession += currTime -
                    awayTeamLastTimePossession;
            } else {
                // saves the last time the away team
                    had the ball
                currTime = getTimeFromTimer(entry.
                    getTime());
                awayTeamLastTimePossession = currTime
                    ;
            }
        } if(entry.getAction().equals("END_GAME")) {
            // saves the last instance of the current
                 time
            if(Boolean.valueOf(entry.getData()[0])){
                // calculates time the home team has
                    had the ball based from
                // the current time and last time the
                     enemy team had possession
                homeTeamTimePossession +=
                    getTimeFromTimer(entry.getTime())
                    - currTime;
            }
            currTime = getTimeFromTimer(entry.getTime
                ());
            if (currTime != 0) {
                // divides time home team has had the
                     ball over the total time
                homeTeamPossession = (
                    homeTeamTimePossession / currTime)
                    ;
            }
        }
    }
    return homeTeamPossession;
}
```

```
    // returns timeline
    ArrayList getTimeline(){
        return timeline;
    }

}
```

7.1.8 Tournament.java

```
package uk.ac.glos.ct5025.s1804317.footballStats;

import org.w3c.dom.*;
import org.xml.sax.SAXException;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    SelectTournamentWindow;
import javax.swing.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

public class Tournament {

    // list of tournaments declared and initialised. all
        tournaments, teams and players are declared in
        regards to the tournament list
    private final static ArrayList<Tournament>
        tournamentList = new ArrayList<Tournament>();

    // actively selected tournament
    private static Tournament activeTournament;

    private String tournamentName;
    // saves all teams within tournament
    private ArrayList<Team> tournamentTeams;
    // saves all games within tournament
    private ArrayList<StaticGame> tournamentGameList;
```

```
// xml file path
private final static String xmlFilePath = "./src/main
    /java/uk/ac/glos/ct5025/s1804317/footballStats/
    Tournaments/";

// displays teams within tournament
private DefaultListModel tournamentTeamsModel = new
    DefaultListModel();

// displays games within tournament
private DefaultListModel tournamentGamesModel = new
    DefaultListModel();

/**
 * Tournament which holds all Team and StaticGame
     classes, held within the tournamentTeams
     ArrayList
 * @param tempName name given to the tournament
 */
public Tournament(String tempName){
    tournamentName = tempName;
    tournamentTeams = new ArrayList();
    tournamentGameList = new ArrayList();
}

// getters

public static Tournament getActiveTournament(){
    return activeTournament;
}

private ArrayList<StaticGame> getTournamentGameList()
    {
    return tournamentGameList;
}

public StaticGame getGame(int i) {
    StaticGame game = tournamentGameList.get(i);
    return game;
}

public String getTournamentName(){
    try{
        return tournamentName;
    } catch (NullPointerException e) {
        return "No tournament selected";
```

```
    }
}

public DefaultListModel getTournamentGamesModel(){
    return tournamentGamesModel;
}

public DefaultListModel getTournamentTeamsModel(){
    return tournamentTeamsModel;
}

public static ArrayList<Tournament> getTournamentList
    (){
    return tournamentList;
}

public ArrayList<Team> getTournamentTeams(){
    return tournamentTeams;
}

// gets team
public Team getTeam(int teamNo){
    Team currTeam = tournamentTeams.get(teamNo);
    return currTeam;
}

// gets team index in regards to the teamList
private int getTeamIndex(Team team){
    ArrayList<Team> teamsList = tournamentTeams;
    int index = teamsList.indexOf(team);
    return index;
}


public void addTeam(Team team){
    // adds team to tournament
    tournamentTeams.add(team);
    // adds team to display
    tournamentTeamsModel.add(tournamentTeamsModel.
        size(), team.getName());
}

public void addGame(StaticGame game) {
    // adds game to tournament
    tournamentGameList.add(game);
    // adds game to display
```

```
        tournamentGamesModel.add(tournamentGamesModel.
            size(), game.getTitle());
    }

    // creates new tournament and appends it to
        tournament list
    public static void createTournament(String
        tournamentName){
        Tournament tournament = null;
        // checks if name is not empty
        if (tournamentName != ""){
            // creates new tournament
            tournament = new Tournament(tournamentName);
            // appends
            addTournament(tournament);
        }
    }

    private static void addTournament(Tournament
        tournament){
        // gets tournament list model and appends new
            tournament to it
        DefaultListModel model = SelectTournamentWindow.
            getTournamentModel();
        model.add(model.getSize(), tournament.
            getTournamentName());

        // adds tournament to tournament list
        tournamentList.add(tournament);
        // if there is only one tournament, it is set as
            the active one
        if(tournamentList.size()==1) {
            selectTournament(0);
        }
    }

    // selects active tournament
    public static void selectTournament(int
        tournamentNumber){
        activeTournament = tournamentList.get(
            tournamentNumber);
        // updates UI text
        SelectTournamentWindow.getCurrTournament().
            setText("ACTIVE TOURNAMENT: " +
            activeTournament.getTournamentName());
```

```
        SelectTournamentWindow.getCurrTournament().
            repaint();
        SelectTournamentWindow.getCurrTournament().
            revalidate();

    }

    public static void importTournament(File file){
        try {
            // creates a new instance of documentBuilder,
                the thing which builds xml elements
            DocumentBuilderFactory documentFactory =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder documentBuilder =
                documentFactory.newDocumentBuilder();
            Document document = documentBuilder.parse(
                file);

            // Normalises document so it's easier for the
                computer to read
            document.getDocumentElement().normalize();

            // new tournament from xml name
            Tournament tournament = new Tournament(
                document.getDocumentElement().getAttribute
                ("name"));

            // adds tournament to list
            addTournament(tournament);

            // gets teams element
            Element teamsList = (Element) document.
                getElementsByTagName("teams").item(0);
            // gets a NodeList of all the teams
            NodeList teamsNodeList = teamsList.
                getElementsByTagName("team");

            for(int i = 0; i < teamsNodeList.getLength();
                i++){
                // turns current team node into element
                    which can be manipulated better
                Element teamElement = (Element)
                    teamsNodeList.item(i);

                // new team from team name
```

```
Team team = new Team(teamElement.
    getElementsByTagName("name").item(0).
    getTextContent());

// sets team attributes from xml
team.setGamesWon(Integer.parseInt(
    teamElement.getElementsByTagName("
    gameswon").item(0).getTextContent()));
team.setGamesLost(Integer.parseInt(
    teamElement.getElementsByTagName("
    gameslost").item(0).getTextContent()))
    ;
team.setGamesDrawn(Integer.parseInt(
    teamElement.getElementsByTagName("
    gamesdrawn").item(0).getTextContent())
    );
team.setGoalsFor(Integer.parseInt(
    teamElement.getElementsByTagName("
    goalsfor").item(0).getTextContent()));
team.setGoalsAgainst(Integer.parseInt(
    teamElement.getElementsByTagName("
    goalsagainst").item(0).getTextContent
    ()));
team.setPoints(Integer.parseInt(
    teamElement.getElementsByTagName("
    teampoints").item(0).getTextContent())
    );

// gets players element
Element playersList = (Element)
    teamElement.getElementsByTagName("
    players").item(0);
// gets a nodelist of all the players
NodeList playersNodeList = playersList.
    getElementsByTagName("player");
for(int j = 0; j < playersNodeList.
    getLength(); j++){
     Element playerElement = (Element)
         playersNodeList.item(j);

     Player player = new Player(
         playerElement.getElementsByTagName
         ("name").item(0).getTextContent(),
         "",team);
     player.setGamesWon(Integer.parseInt(
         playerElement.getElementsByTagName
```

```
                ("gameswon").item(0).
                    getTextContent()));
            player.setGamesLost(Integer.parseInt(
                playerElement.getElementsByTagName
                ("gameslost").item(0).
                    getTextContent()));
            player.setGamesDrawn(Integer.parseInt
                (playerElement.
                getElementsByTagName("gamesdrawn")
                .item(0).getTextContent()));
            player.setGoalsFor(Integer.parseInt(
                playerElement.getElementsByTagName
                ("goalsfor").item(0).
                    getTextContent()));

            // adds player to team
            team.addPlayer(player);
        }

        // adds team to tournament
        tournament.addTeam(team);

    }

    // gets games
    Element gamesList = (Element) document.
        getElementsByTagName("games").item(0);
    NodeList gameNodeList = gamesList.
        getElementsByTagName("game");

    for(int i = 0; i < gameNodeList.getLength();
        i++) {
        Element gameElement = (Element)
            gameNodeList.item(i);

        // gets date, time, home team possession
        String gameDate = gameElement.
            getElementsByTagName("date").item(0).
            getTextContent();
        String gameTime = gameElement.
            getElementsByTagName("time").item(0).
            getTextContent();
        float gamePossession = Float.parseFloat(
            gameElement.getElementsByTagName("
            possession").item(0).getTextContent())
            ;
```

```
// gets timeline
Element timeLineElement = (Element)
    gameElement.getElementsByTagName("
    timeline").item(0);

NodeList entryNodeList = timeLineElement.
    getElementsByTagName("entry");

// creates new timeline
ArrayList<Entry> timeline = new ArrayList
    <Entry>();

// adds entries to timeline
for(int j = 0; j < entryNodeList.
    getLength(); j++){
     Element entryElement = (Element)
         entryNodeList.item(j);

     String entryTime = entryElement.
         getElementsByTagName("time").item
         (0).getTextContent();
     String entryAction = entryElement.
         getElementsByTagName("action").
         item(0).getTextContent();
     String entryOutput = entryElement.
         getElementsByTagName("output").
         item(0).getTextContent();
     String[] entryData = new String[]{
              entryElement.
                  getElementsByTagName("data
                  ").item(0).getTextContent
                  (),"0"
     };


     Entry entry = new Entry(entryTime,
         entryAction,entryData,entryOutput)
         ;

     timeline.add(entry);
}

// gets home team attributes
Element homeTeamElement = (Element)
    gameElement.getElementsByTagName("
```

```
    hometeam").item(0);

String homeTeamName = homeTeamElement.
    getElementsByTagName("name").item(0).
    getTextContent();
int homeTeamIndex = Integer.parseInt(
    homeTeamElement.getAttribute("id"));
int homeTeamGoals = Integer.parseInt(
    homeTeamElement.getElementsByTagName("
    goals").item(0).getTextContent());

Element homeTeamPlayersElement = (Element
    ) gameElement.getElementsByTagName("
    players").item(0);
NodeList homeTeamPlayersNodeList =
    homeTeamPlayersElement.
    getElementsByTagName("player");

ArrayList<String[]> homePlayerData = new
    ArrayList<String[]>();

for(int j = 0; j <
    homeTeamPlayersNodeList.getLength(); j
    ++ ){
     Element playerElement = (Element)
         homeTeamPlayersNodeList.item(j);

     String[] playerData = new String[]{
              playerElement.
                  getElementsByTagName("name
                  ").item(0).getTextContent
                  (),
              playerElement.
                  getElementsByTagName("
                  goals").item(0).
                  getTextContent()
     };

     homePlayerData.add(playerData);
}

// gets away team attributes
Element awayTeamElement = (Element)
    gameElement.getElementsByTagName("
    awayteam").item(0);
```

```
int awayTeamIndex = Integer.parseInt(
    awayTeamElement.getAttribute("id"));
String awayTeamName = awayTeamElement.
    getElementsByTagName("name").item(0).
    getTextContent();
int awayTeamGoals = Integer.parseInt(
    awayTeamElement.getElementsByTagName("
    goals").item(0).getTextContent());


Element awayTeamPlayersElement = (Element
    ) awayTeamElement.getElementsByTagName
    ("players").item(0);
NodeList awayTeamPlayersNodeList =
    awayTeamPlayersElement.
    getElementsByTagName("player");

ArrayList<String[]> awayPlayerData = new
    ArrayList<String[]>();

for(int j = 0; j <
    awayTeamPlayersNodeList.getLength(); j
    ++ ){
     Element playerElement = (Element)
         awayTeamPlayersNodeList.item(j);

     String[] playerData = new String[]{
              playerElement.
                  getElementsByTagName("name
                  ").item(0).getTextContent
                  (),
              playerElement.
                  getElementsByTagName("
                  goals").item(0).
                  getTextContent()
     };

     awayPlayerData.add(playerData);
}

// initialises static game from all the
    imported data
StaticGame game = new StaticGame(
    homeTeamName,awayTeamName,
    homeTeamGoals,awayTeamGoals,gameDate,
```

```
                        gameTime, homePlayerData,
                            awayPlayerData, gamePossession,
                            homeTeamIndex, awayTeamIndex,
                            timeline);

                    // appends game to tournament
                    tournament.addGame(game);

                }

                // catches errors
            } catch (ParserConfigurationException e) {
                e.printStackTrace();

            } catch (SAXException e) {
                e.printStackTrace();
                // Cannot parse file
            } catch (IOException e) {
                e.printStackTrace();
            }


    }

    public static void exportTournament(Tournament
        tournament){
        String path = xmlFilePath + tournament.
            getTournamentName() + ".xml";
        try {
            // creates document which is then used for
                element manipulation
            DocumentBuilderFactory documentFactory =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder documentBuilder =
                documentFactory.newDocumentBuilder();
            Document document = documentBuilder.
                newDocument();

            // creates root element
            Element root = document.createElement("
                tournament");
            // appends tournament to root
            document.appendChild(root);
            // create's the tournament's attribute: name
            root.setAttribute("name",tournament.
                getTournamentName());
```

```
// appends teams to tournament
Element teams = document.createElement("teams
    ");
teams.setAttribute("id","teams");
root.appendChild(teams);

Element team;
int teamID = 0;

// Constructs XML element for each team with
    corresponding data and appends it to the
    teams element
for (Team currTeam : tournament.
    getTournamentTeams()){
     team = document.createElement("team");
     teams.appendChild(team);

     team.setAttribute("id",Integer.toString(
         teamID));

     Element name = document.createElement("
         name");
     name.appendChild(document.createTextNode(
         currTeam.getName()));
     team.appendChild(name);

     Element gamesWon = document.createElement
         ("gameswon");
     gamesWon.appendChild(document.
         createTextNode(Integer.toString(
         currTeam.getGamesWon())));
     team.appendChild(gamesWon);

     Element gamesLost = document.
         createElement("gameslost");
     gamesLost.appendChild(document.
         createTextNode(Integer.toString(
         currTeam.getGamesLost())));
     team.appendChild(gamesLost);

     Element gamesDrawn = document.
         createElement("gamesdrawn");
     gamesDrawn.appendChild(document.
         createTextNode(Integer.toString(
         currTeam.getGamesLost())));
```

```
team.appendChild(gamesDrawn);

Element goalsFor = document.createElement
    ("goalsfor");
goalsFor.appendChild(document.
    createTextNode(Integer.toString(
    currTeam.getGoalsFor())));
team.appendChild(goalsFor);

Element goalsAgainst = document.
    createElement("goalsagainst");
goalsAgainst.appendChild(document.
    createTextNode(Integer.toString(
    currTeam.getGoalsAgainst())));
team.appendChild(goalsAgainst);

Element teamPoints = document.
    createElement("teampoints");
teamPoints.appendChild(document.
    createTextNode(Integer.toString(
    currTeam.getPoints())));
team.appendChild(teamPoints);

Element teamPlayers = document.
    createElement("players");
team.appendChild(teamPlayers);

int playerID = 0;
Element player;

// Constructs XML format for each player
    within a team with corresponding data
    and appends to current team
for (Player currPlayer : currTeam.
    getTeamPlayers()){
     player = document.createElement("
         player");
     teamPlayers.appendChild(player);

     player.setAttribute("id",Integer.
         toString(playerID));

     name = document.createElement("name")
         ;
     name.appendChild(document.
         createTextNode(currPlayer.getName
```

```
                ()));
            player.appendChild(name);

            gamesWon = document.createElement("
                gameswon");
            gamesWon.appendChild(document.
                createTextNode(Integer.toString(
                currPlayer.getGamesWon())));
            player.appendChild(gamesWon);

            gamesLost = document.createElement("
                gameslost");
            gamesLost.appendChild(document.
                createTextNode(Integer.toString(
                currPlayer.getGamesLost())));
            player.appendChild(gamesLost);

            gamesDrawn = document.createElement("
                gamesdrawn");
            gamesDrawn.appendChild(document.
                createTextNode(Integer.toString(
                currPlayer.getGamesLost())));
            player.appendChild(gamesDrawn);

            goalsFor = document.createElement("
                goalsfor");
            goalsFor.appendChild(document.
                createTextNode(Integer.toString(
                currPlayer.getGoalsFor())));
            player.appendChild(goalsFor);

            playerID++;
        }

        teamID++;
    }

    // creates list of played games
    Element games = document.createElement("games
        ");
    root.appendChild(games);

    Element game;
    int gameID = 0;
```

```
// Constructs XML format for each game within
     a tournament with corresponding data and
    appends to
// current tournament

for (StaticGame currGame : tournament.
    getTournamentGameList()){
     game = document.createElement("game");
     games.appendChild(game);

     game.setAttribute("id",Integer.toString(
         gameID));

     Element name = document.createElement("
         title");
     name.appendChild(document.createTextNode(
         currGame.getTitle()));
     game.appendChild(name);

     Element gameDate = document.createElement
         ("date");
     gameDate.appendChild(document.
         createTextNode(currGame.getGameDate())
         );
     game.appendChild(gameDate);

     Element gameTime = document.createElement
         ("time");
     gameTime.appendChild(document.
         createTextNode(currGame.getGameTime())
         );
     game.appendChild(gameTime);

     Element gamePossession = document.
         createElement("possession");
     gamePossession.appendChild(document.
         createTextNode(Float.toString(currGame
         .getPossession())));
     game.appendChild(gamePossession);

     Element homeTeam = document.createElement
         ("hometeam");
     homeTeam.setAttribute("id",Integer.
         toString(tournament.getTeamIndex(
         currGame.getHomeTeamObject())));
     game.appendChild(homeTeam);
```

```
Element homeTeamName = document.
    createElement("name");
homeTeamName.appendChild(document.
    createTextNode(currGame.getHomeTeam())
    );
homeTeam.appendChild(homeTeamName);

Element awayTeam = document.createElement
    ("awayteam");
awayTeam.setAttribute("id",Integer.
    toString(tournament.getTeamIndex(
    currGame.getAwayTeamObject()))));
game.appendChild(awayTeam);


Element awayTeamName = document.
    createElement("name");
awayTeamName.appendChild(document.
    createTextNode(currGame.getAwayTeam())
    );
awayTeam.appendChild(awayTeamName);

Element homeTeamGoals = document.
    createElement("goals");
homeTeamGoals.appendChild(document.
    createTextNode(Integer.toString(
    currGame.getHomeTeamGoals())));
homeTeam.appendChild(homeTeamGoals);

Element awayTeamGoals = document.
    createElement("goals");
awayTeamGoals.appendChild(document.
    createTextNode(Integer.toString(
    currGame.getAwayTeamGoals())));
awayTeam.appendChild(awayTeamGoals);

Element homeTeamPlayers = document.
    createElement("players");
homeTeam.appendChild(homeTeamPlayers);

Element awayTeamPlayers = document.
    createElement("players");
awayTeam.appendChild(awayTeamPlayers);
```

```
Element timeline = document.createElement
    ("timeline");
game.appendChild(timeline);

int entryID = 0;
// Constructs XML format for each entry
    within the game's timeline with
    corresponding data and appends
// to current game
for (Entry arrayEntry : currGame.
    getTimeline()) {
    Element entry = document.
        createElement("entry");
    entry.setAttribute("id",Integer.
        toString(entryID));
    timeline.appendChild(entry);

    Element entryTime = document.
        createElement("time");
    entryTime.appendChild(document.
        createTextNode(arrayEntry.getTime
        ()));
    entry.appendChild(entryTime);

    Element entryAction = document.
        createElement("action");
    entryAction.appendChild(document.
        createTextNode(arrayEntry.
        getAction()));
    entry.appendChild(entryAction);

    if(arrayEntry.getData().equals("
        SCR_GOAL")){
        String[] data = arrayEntry.
            getData();

        Element entryData = document.
            createElement("data");
        entryData.appendChild(document.
            createTextNode(data[0]+data
            [1]));
        entry.appendChild(entryData);
    } else {
        Element entryData = document.
            createElement("data");
```

```
            entryData.appendChild(document.
                createTextNode(arrayEntry.
                getData()[0]));
            entry.appendChild(entryData);
        }
        Element entryOutput = document.
            createElement("output");
        entryOutput.appendChild(document.
            createTextNode(arrayEntry.
            getOutput()));
        entry.appendChild(entryOutput);

        entryID++;
    }


    int playerID = 0;
    // appends home players within the game
    for (String[] playerTuple : currGame.
        getHomePlayerData()){
        Element player = document.
            createElement("player");
        player.setAttribute("id",Integer.
            toString(playerID));
        homeTeamPlayers.appendChild(player);

        Element playerName = document.
            createElement("name");
        playerName.appendChild(document.
            createTextNode(playerTuple[0]));
        player.appendChild(playerName);

        Element playerGoals = document.
            createElement("goals");
        playerGoals.appendChild(document.
            createTextNode(playerTuple[1]));
        player.appendChild(playerGoals);
        playerID++;
    }

    playerID = 0;
    // appends away players within the game
    for (String[] playerTuple : currGame.
        getAwayPlayerData()){
        Element player = document.
            createElement("player");
```

```
                    player.setAttribute("id",Integer.
                        toString(playerID));
                    awayTeamPlayers.appendChild(player);

                    Element playerName = document.
                        createElement("name");
                    playerName.appendChild(document.
                        createTextNode(playerTuple[0]));
                    player.appendChild(playerName);

                    Element playerGoals = document.
                        createElement("goals");
                    playerGoals.appendChild(document.
                        createTextNode(playerTuple[1]));
                    player.appendChild(playerGoals);
                    playerID++;
                }


                gameID++;
            }

            // creates transformer to create
                corresponding xml code from document
                instance
            TransformerFactory transformerFactory =
            TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.
                newTransformer();
            DOMSource domSource = new DOMSource(document)
                ;
            StreamResult streamResult = new StreamResult(
                new File(path));

            transformer.transform(domSource,streamResult)
                ;

            // catch errors
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (TransformerException tfe) {
            tfe.printStackTrace();
        }
    }
}
```

7.2 UI Main

7.2.1 CardLayoutWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI;

import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.Browse
    .BrowseTeamsWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.Browse
    .BrowseWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.Create
    .CreateTeamWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.Create
    .CreateTournamentWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.Create
    .CreateWindow;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;
import java.awt.*;
import java.util.ArrayList;

public class CardLayoutWindow {
    public static CardLayoutWindow cardLayoutWindow = new
        CardLayoutWindow();

    public void displayGUI() {
        JFrame frame = new JFrame("Football Stats");
        frame.setDefaultCloseOperation(JFrame.
            EXIT_ON_CLOSE);

        GridBagLayout gridBagLayout = new GridBagLayout()
            ;
        frame.setLayout(gridBagLayout);

        JPanel contentPane = new JPanel();
        contentPane.setBorder(BorderFactory.
            createEmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new CardLayout());

        BrowseWindow browseWindow = new BrowseWindow(
            contentPane, cardLayoutWindow);
```

```
        MainMenuWindow mainWindow = new MainMenuWindow(
            contentPane, cardLayoutWindow);
        CreateWindow createWindow = new CreateWindow(
            contentPane, cardLayoutWindow);
        CreateTournamentWindow createTournamentWindow =
            new CreateTournamentWindow(contentPane,
            cardLayoutWindow);
        SelectTournamentWindow selectTournamentWindow =
            new SelectTournamentWindow(contentPane,
            cardLayoutWindow);
        CreateTeamWindow createTeamWindow = new
            CreateTeamWindow(contentPane, cardLayoutWindow
            );

        contentPane.add(mainWindow,"NAV_MAIN");
        contentPane.add(browseWindow,"NAV_BROWSE");
        contentPane.add(createWindow, "NAV_CREATE");
        contentPane.add(createTournamentWindow,"
            NAV_CREATE_TOURNAMENT");
        contentPane.add(selectTournamentWindow,"
            NAV_SELECT_TOURNAMENT");
        contentPane.add(createTeamWindow,"NAV_CREATE_TEAM
            ");

        frame.getContentPane().add(contentPane);
        frame.setSize(MyWindow.getSizeX(),MyWindow.
            getSizeY());
        frame.setLocationByPlatform(true);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread
            :
        //creating and showing this application's GUI.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new CardLayoutWindow().displayGUI();
            }
        });
    }

    public static class MyTableModel extends
        AbstractTableModel {
```

```
// Creates a String Array of column names
private String[] columnNames = {"Team Name",
        "GP","W","L","D","GF","GA","GD","Pts"};

// Creates a multidimensional array, each
    containing a team's data.
private ArrayList<ArrayList> data(){
    ArrayList<ArrayList> data = new ArrayList<
        ArrayList>();
    for (int i = 0; i < Tournament.
        getActiveTournament().getTournamentTeams()
        .size(); i++){
        Team currTeam = Tournament.
            getActiveTournament().getTeam(i);
        ArrayList teamData = new ArrayList();
        teamData.add(currTeam.getName());
        teamData.add(currTeam.getGamesPlayed());
        teamData.add(currTeam.getGamesWon());
        teamData.add(currTeam.getGamesLost());
        teamData.add(currTeam.getGamesDrawn());
        teamData.add(currTeam.getGoalsFor());
        teamData.add(currTeam.getGoalsAgainst());
        teamData.add(currTeam.getGoalsDifference
            ());
        teamData.add(currTeam.getPoints());
        data.add(teamData);
    }
    return data;
}

//    @Override is not allowed when implementing
    interface method
public int getRowCount() {
    return data().size();
}

//    @Override is not allowed when implementing
    interface method
public int getColumnCount() {
    return columnNames.length;
}

// Gets value at specified cell co-ordinates
//    @Override is not allowed when implementing
    interface method
public Object getValueAt(int row, int col) {
```

```
            // Gets column
            ArrayList colData = data().get(row);
            // Gets data
            Object data = colData.get(col);
            return data;
        }

        @Override
        public Class getColumnClass(int c) {
            return getValueAt(0, c).getClass();
        }

        @Override
        public String getColumnName(int col) {
            return columnNames[col];
        }

        // Denies cells being editable
        public boolean isCellEditable(int row, int col) {
            return false;
        }
    }
}
```

7.2.2 MainMenuWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI;

import uk.ac.glos.ct5025.s1804317.footballStats.UI.Browse
    .BrowseTeamsSelectWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MainMenuWindow extends MyWindow implements
    ActionListener {

    private JComponent mainWindow;

    /**
     * Main menu window to browse the rest of the program
     * @param panel panel which will be displayed
```

```
     * @param clw layout to which the panel will be
         appended to
     */
    public MainMenuWindow(JPanel panel, CardLayoutWindow
        clw) {
        super(panel, clw);
        contentPane = panel;
        setOpaque(true);

        mainWindow = createMainMenu();
        add(mainWindow);
    }

    public JComponent createMainMenu(){
        JLabel menuLabel = new JLabel("FOOTBALL STATS");

        JComponent buttonSelectTournament =
            factoryButtonPane("Select Tournament", "
            NAV_SELECT_TOURNAMENT");
        JComponent buttonCreate = factoryButtonPane("
            Create..", "NAV_CREATE");
        JComponent buttonBrowse = factoryButtonPane("
            Browse..", "NAV_BROWSE");
        JComponent buttonStartGame = factoryButtonPane("
            Start Game","MKE_SELECT_HOME");

        JPanel panel = new JPanel();

        // GridBagLayout is not needed for a simple menu,
             but it was used for consistency throughout
            the program
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        // Sets the padding
        gbc.insets = new Insets(3,3,3,3);

        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(menuLabel,gbc);

        gbc.gridy++;
        panel.add(buttonCreate,gbc);

        gbc.gridy++;
        panel.add(buttonBrowse,gbc);
```

```
        gbc.gridy++;
        panel.add(buttonSelectTournament,gbc);

        gbc.gridy++;
        panel.add(buttonStartGame,gbc);

        return panel;
    }

    // Handle action events from all the buttons
    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command.contains("NAV_")) {
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            cardLayout.show(contentPane, command);
        } else if(command.equals("MKE_SELECT_HOME")){
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            BrowseTeamsSelectWindow
                browseTeamsSelectWindow = new
                BrowseTeamsSelectWindow(contentPane,
                CardLayoutWindow.cardLayoutWindow,"HOME");
            contentPane.add(browseTeamsSelectWindow,
                command);
            cardLayout.show(contentPane,command);
        }
    }
}
```

7.2.3 MyWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI;

import uk.ac.glos.ct5025.s1804317.footballStats.Item;
import uk.ac.glos.ct5025.s1804317.footballStats.
    StaticGame;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
public abstract class MyWindow extends JPanel implements
    ActionListener {
    private static int sizeX;
    private static int sizeY;
    private static JLabel errorMessage = new JLabel();

    // Last location to allow new windows not to overlap
    private Point lastLocation = null;

    protected JPanel contentPane;

    public MyWindow(JPanel panel, CardLayoutWindow clw) {
        contentPane = panel;

        Dimension screenSize = Toolkit.getDefaultToolkit
            ().getScreenSize();
        sizeX = (int) Math.round(screenSize.width * .75);
        sizeY = (int) Math.round(screenSize.height * .75)
            ;
    }

    public JPanel getContentPane(){
        //contentpane is the panel with cardLayout
        return contentPane;
    }

    public static int getSizeX() {
        return sizeX;
    }

    public static JLabel getErrorMessage(){
        return errorMessage;
    }

    public static int getSizeY() {
        return sizeY;
    }

    protected JComponent factoryList(JList list){
        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setPreferredSize(new Dimension
            (150,200));
        scrollPane.setViewportView(list);
        return scrollPane;
    }
```

```
protected JTextField factoryTextField(String title){
    JTextField textField = new JTextField();
    new TextPrompt(title, textField, TextPrompt.Show.
        FOCUS_LOST);
    textField.setPreferredSize(new Dimension(150,25))
        ;
    textField.setBorder(BorderFactory.
        createEmptyBorder(5,5,5,5));
    return textField;
}

protected JComponent factoryButtonPane(String title,
    String command) {
    JButton button = new JButton(title);
    button.setPreferredSize(new Dimension(150, 25));
    button.setActionCommand(command);
    button.addActionListener(this);

    JPanel pane = new JPanel();
    pane.setBorder(BorderFactory.createEmptyBorder(5,
        5, 5, 5));
    pane.add(button);

    return pane;
}

public void displayGameFrame (StaticGame game) {
    JFrame frame = new JFrame(game.getTitle());
    frame.setDefaultCloseOperation(WindowConstants.
        DISPOSE_ON_CLOSE);

    //Set window location.
    if (lastLocation != null) {
        //Move the window over and down 40 pixels.
        lastLocation.translate(40, 40);
        if ((lastLocation.x > sizeX) || (lastLocation
            .y > sizeY)) {
            lastLocation.setLocation(0, 0);
        }
        frame.setLocation(lastLocation);
    } else {
        lastLocation = frame.getLocation();
    }

    JPanel panel = new JPanel();
    panel.setLayout(new GridBagLayout());
```

```
GridBagConstraints gbc = new GridBagConstraints()
    ;

gbc.insets = new Insets(3,3,3,3);
gbc.anchor = GridBagConstraints.FIRST_LINE_START;

gbc.gridx = 2; gbc.gridy = 0;
panel.add(new JLabel(game.getHomeTeamGoals() + "␣
    ␣" + game.getHomeTeam() + "␣vs␣" + game.
    getAwayTeam() + "␣␣" + game.getAwayTeamGoals()
    ),gbc);
gbc.gridx = 0;

////////// Add Home Players

gbc.gridy++;
panel.add(new JLabel(game.getHomeTeam()),gbc);

gbc.gridy++;
panel.add(new JLabel("Player␣name"),gbc);

gbc.gridx++;
panel.add(new JLabel("Goals"),gbc);

gbc.gridy--;
panel.add(new JLabel(String.valueOf(Math.round(
    game.getPossession()*100))+ "%"),gbc);
gbc.gridy++;

gbc.gridx--;

gbc.gridy++;

// adds each player's data from the game
for (int i=0; i < game.getHomePlayerData().size()
    ; i++){
     String[] player = game.getHomePlayerData().
         get(i);
     panel.add(new JLabel(player[0]),gbc);
     gbc.gridx++;
     panel.add(new JLabel(player[1]),gbc);
     gbc.gridx--;
     gbc.gridy++;
}

////////// Add Away Players
```

```
gbc.gridy = 1;

gbc.gridx = 5;

panel.add(new JLabel(game.getAwayTeam()),gbc);

gbc.gridy++;
panel.add(new JLabel("Player Name"),gbc);

gbc.gridx--;
panel.add(new JLabel("Goals"),gbc);

gbc.gridy--;
panel.add(new JLabel(String.valueOf(Math.round((1
    - game.getPossession())*100)) + "%"),gbc);
gbc.gridy++;

gbc.gridy++;


for (int i=0; i < game.getAwayPlayerData().size()
    ; i++){
    String[] player = game.getAwayPlayerData().
        get(i);
    panel.add(new JLabel(player[1]),gbc);
    gbc.gridx++;
    panel.add(new JLabel(player[0]),gbc);
    gbc.gridx--;
    gbc.gridy++;
}

////////// Add Game Time and Details
gbc.gridy = 1; gbc.gridx = 2;

panel.add(new JLabel(game.getGameDate(),
    SwingConstants.CENTER),gbc);
gbc.gridy++;
panel.add(new JLabel(game.getGameTime(),
    SwingConstants.CENTER),gbc);

////////// Add Contents

frame.setLayout(new GridBagLayout());
GridBagConstraints gbc2 = new GridBagConstraints
    ();
```

```
        gbc2.anchor = GridBagConstraints.FIRST_LINE_START
            ;

        frame.add(panel,gbc2);


        frame.pack();
//          frame.setSize(new Dimension(350,200));
        frame.setVisible(true);
    }

    public void displayFrame (Item item){
        JFrame frame = new JFrame(item.getName());
        frame.setDefaultCloseOperation(WindowConstants.
            DISPOSE_ON_CLOSE);

        //Set window location.
        if (lastLocation != null) {
            //Move the window over and down 40 pixels.
            lastLocation.translate(40, 40);
            if ((lastLocation.x > sizeX) || (lastLocation
                .y > sizeY)) {
                lastLocation.setLocation(0, 0);
            }
            frame.setLocation(lastLocation);
        } else {
            lastLocation = frame.getLocation();
        }

        JPanel panel = new JPanel();
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        gbc.insets = new Insets(3,3,3,3);
        gbc.anchor = GridBagConstraints.FIRST_LINE_START;

        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(new JLabel(item.getName()),gbc);

        gbc.gridx++;
        panel.add(new JLabel("Games Won: " + item.
            getGamesWon()),gbc);

        gbc.gridy++;
```

```
            panel.add(new JLabel("Games Lost: " + item.
                getGamesLost()),gbc);

            gbc.gridy++;
            panel.add(new JLabel("Games Drawn: " + item.
                getGamesDrawn()),gbc);

            gbc.gridy++;
            panel.add(new JLabel("Goals Scored: " + item.
                getGoalsFor()),gbc);


            frame.setLayout(new GridBagLayout());
            GridBagConstraints gbc2 = new GridBagConstraints
                ();
            gbc2.anchor = GridBagConstraints.FIRST_LINE_START
                ;

            frame.add(panel,gbc2);


            frame.pack();
//          frame.setSize(new Dimension(350,200));
            frame.setVisible(true);

        }

        //    @Override is not allowed when implementing
            interface method
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            CardLayout cardLayout = (CardLayout) contentPane.
                getLayout();
            cardLayout.show(contentPane, command);
        }
}
```

7.2.4 PlayGameWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI;

import uk.ac.glos.ct5025.s1804317.footballStats.*;

import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Arrays;

public class PlayGameWindow extends MyWindow {


    private JLabel timeLabel = new JLabel("00:00");

    private JFrame frame;

    private Game game;

    private JLabel scoreLabel;

    private JLabel errorMsg = new JLabel();



    private int staticHomeGoals = 0;
    private int staticAwayGoals = 0;
    private JLabel possessionLabel;
    private JList homePlayersList;
    private JList awayPlayersList;
    private DefaultListModel homeModel;
    private DefaultListModel awayModel;
    private Point lastLocation = null;
    private JScrollPane scrollTextArea;
    private JTextField textField;
    private Team homeTeam;
    private Team awayTeam;
    private JTextArea textArea;
    private String mode;
    private StaticGame staticGame = null;
    private ArrayList<Entry> timeline;
    private Timer timer;

    private ActionListener changeTime = new
        ActionListener() {
        public void actionPerformed(ActionEvent env) {
            timeLabel.setText(game.getGameTimer().
                getWatchTime());
            if (mode.equals("PLAYBACK")){
                playEntry();
```

```
            }
            if( game.getGameTimer().getStopWatchTime() >
                5400 ){
                finishGame();
            }
        }
};

/**
 * Loads active game
 * @param panel panel which will be displayed
 * @param clw layout to which the panel will be
     appended to
 * @param homeTeamTemp home team
 * @param awayTeamTemp away team
 * @param tempGame active game
 * @param tempMode mode which is being played
 */
public PlayGameWindow(JPanel panel, CardLayoutWindow
    clw, Team homeTeamTemp, Team awayTeamTemp, Game
    tempGame, String tempMode) {
    super(panel, clw);
    homeTeam = homeTeamTemp;
    awayTeam = awayTeamTemp;
    homeModel = homeTeam.getTeamActivePlayersModel();
    awayModel = awayTeam.getTeamActivePlayersModel();
    game = tempGame;
    mode = tempMode;
}

/**
 * Loads static game
 * @param panel panel which will be displayed
 * @param clw layout to which the panel will be
     appended to
 * @param tempStaticGame static game contains home
     and away team as well as game details
 * @param tempMode mode which is being played
 */
public PlayGameWindow(JPanel panel, CardLayoutWindow
    clw, StaticGame tempStaticGame, String tempMode){
    super(panel,clw);
    staticGame = tempStaticGame;
    homeTeam = staticGame.getHomeTeamObject();
    awayTeam = staticGame.getAwayTeamObject();
    game = new Game(homeTeam,awayTeam);
```

```
        homeModel = new DefaultListModel();
        awayModel = new DefaultListModel();
        mode = tempMode;
        timeline = (ArrayList<Entry>) staticGame.
            getTimeline().clone();
    }

    private void playEntry(){
        String s;
        if (timeline.size()!=0) {
            if (timeline.get(0).getTime().equals(game.
                getGameTimer().getWatchTime())) {
                if (timeline.get(0).getAction().equals("
                    SCR_GOAL")) {
                    if (timeline.get(0).getData()[0].
                        equals("0")) {
                        staticHomeGoals++;
                    } else {
                        staticAwayGoals++;
                    }
                    scoreLabel.setText(staticHomeGoals +
                        " - " + staticAwayGoals);
                    s = " (SYS): ";
                } else if (timeline.get(0).getAction().
                    equals("CGE_POSSESSION")) {
                    String s1;
                    if (timeline.get(0).getData()[0].
                        equals("true")) {
                        s1 = ("POSSESSION: " + homeTeam.
                            getName());
                    } else {
                        s1 = ("POSSESSION: " + awayTeam.
                            getName());
                    }
                    possessionLabel.setText(s1);
                    s = " (SYS): ";
                } else if (timeline.get(0).getAction().
                    equals("COMMENT")) {
                    s = " (USR): ";


                } else { // timeline.get(0).getAction().
                    equals("END_GAME")
                    s = " (SYS): ";
                    timer.stop();
                    frame.dispose();
```

```
                }
                textArea.append("(" + game.getGameTimer()
                    .getWatchTime() + ")" + s + timeline.
                    get(0).getOutput() + "\n");
                timeline.remove(0);
                playEntry();
            }
        }
    }

    public void finishGame(){
        game.getGameTimer().getStopWatch().stop();
        game.getTimeLine().writeEndGame(game.getGameTimer
            ().getWatchTime(),game.getPossession());
        game.getTimeLine().calculatePossession();
        StaticGame currGame = new StaticGame(game);
        Tournament.getActiveTournament().addGame(currGame
            );
        game.finishGame();
        homeModel.clear();
        awayModel.clear();
        frame.dispose();
    }

    public void displayGameWindow(){

        timer = new Timer(1000,changeTime);
        timer.start();

        JComponent endGameButton;
        JComponent scoreButtonAway;
        JComponent scoreButtonHome;
        JComponent possessionButton;
        JButton submitButton;

        if (mode.equals("RECORD")) {
            Player currPlayer;
            for (int i = 0; i < homeTeam.getActivePlayers
                ().size(); i++) {
                currPlayer = homeTeam.getActivePlayer(i);
                homeModel.add(homeModel.size(),
                    currPlayer.getName());
            }
            for (int i = 0; i < awayTeam.getActivePlayers
                ().size(); i++) {
```

```
            currPlayer = awayTeam.getActivePlayer(i);
            awayModel.add(awayModel.size(),
                currPlayer.getName());
        }

        homeModel.add(0, "N/A");
        awayModel.add(0, "N/A");

        homePlayersList = new JList(homeTeam.
            getTeamActivePlayersModel());
        awayPlayersList = new JList(awayTeam.
            getTeamActivePlayersModel());

        textField = new JTextField();

        submitButton = new JButton("Submit");
        submitButton.setActionCommand("ACT_SUBMIT");
        submitButton.addActionListener(this);

        possessionButton = factoryButtonPane("Change
            Possession","ACT_CHANGE_POSSESSION");
        scoreButtonHome = factoryButtonPane("Goal","
            SCR_GOAL_HOME");
        scoreButtonAway = factoryButtonPane("Goal","
            SCR_GOAL_AWAY");
        endGameButton = factoryButtonPane("End_Game",
            "ACT_END_GAME");

    } else { // if mode.equals("PLAYBACK") {
        String[] currPlayer;
        for (int i = 0; i < staticGame.
            getHomePlayerData().size(); i++){
            currPlayer = staticGame.getHomePlayerData
                ().get(i);
            homeModel.add(homeModel.size(),currPlayer
                [0]);
        }
        for (int i = 0; i < staticGame.
            getAwayPlayerData().size(); i++){
            currPlayer = staticGame.getAwayPlayerData
                ().get(i);
            awayModel.add(awayModel.size(),currPlayer
                [0]);
        }

        homeModel.add(0, "N/A");
```

```
        awayModel.add(0, "N/A");

        homePlayersList = new JList(homeModel);
        awayPlayersList = new JList(awayModel);

        textField = new JTextField();

        submitButton = new JButton("Submit");

        possessionButton = factoryButtonPane("Change
            Possession","..");
        scoreButtonHome = factoryButtonPane("Goal","
            ..");
        scoreButtonAway = factoryButtonPane("Goal","
            ..");
        endGameButton = factoryButtonPane("End Game",
            "ACT_END_STATIC_GAME");
    }

    JComponent homeTeamScrollList = factoryList(
        homePlayersList);
    JComponent awayTeamScrollList = factoryList(
        awayPlayersList);

    frame = new JFrame();
    frame.setDefaultCloseOperation(WindowConstants.
        DISPOSE_ON_CLOSE);

    //Set window location.
    if (lastLocation != null) {
        //Move the window over and down 40 pixels.
        lastLocation.translate(40, 40);
        if ((lastLocation.x > getSizeX() || (
            lastLocation.y > getSizeY()))) {
             lastLocation.setLocation(0, 0);
        }
        frame.setLocation(lastLocation);
    } else {
        lastLocation = frame.getLocation();
    }


    JLabel homeTeamName = new JLabel(homeTeam.getName
        ());
    JLabel awayTeamName = new JLabel(awayTeam.getName
        ());
```

```
scoreLabel = new JLabel(homeTeam.getGoals() + " -
    " + awayTeam.getGoals());

possessionLabel = new JLabel("Possession: "+
    homeTeam.getName());

textArea = new JTextArea();
// makes user unable to edit area
textArea.setEditable(false);
// makes text wrap around instead of going in an
    infinite line
textArea.setLineWrap(true);
scrollTextArea = new JScrollPane(textArea);
scrollTextArea.setPreferredSize(new Dimension
    (100,100));



JPanel panel = new JPanel();
panel.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints()
    ;

Insets defaultInsets = new Insets(5,5,5,5);

gbc.insets = defaultInsets;

gbc.gridx = 0; gbc.gridy = 0;
panel.add(homeTeamName,gbc);

gbc.gridx++;
panel.add(scoreLabel,gbc);

gbc.gridx++;
panel.add(awayTeamName,gbc);

gbc.gridx = 1; gbc.gridy = 0;
panel.add(scoreLabel,gbc);

gbc.gridy++;
panel.add(timeLabel,gbc);

gbc.gridy++;
panel.add(possessionButton,gbc);
```

```
        gbc.gridy++;
        panel.add(possessionLabel,gbc);

        gbc.gridx = 0; gbc.gridy++;
        panel.add(scoreButtonHome,gbc);

        gbc.gridy++;
        panel.add(homeTeamScrollList,gbc);

        gbc.gridy--; gbc.gridx = 2;
        panel.add(scoreButtonAway,gbc);

        gbc.gridy++;
        panel.add(awayTeamScrollList,gbc);

        gbc.gridx--;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.anchor = GridBagConstraints.SOUTH;

        panel.add(endGameButton,gbc);

        gbc.insets = new Insets(0,0,0,0);

        gbc.gridy++; gbc.gridx = 0;
        gbc.gridwidth=3;
        gbc.fill = GridBagConstraints.BOTH;
        panel.add(scrollTextArea,gbc);

        gbc.gridy++; gbc.gridwidth = 1;
        panel.add(submitButton,gbc);

        gbc.gridx++; gbc.gridwidth = 2;
        panel.add(textField,gbc);

        frame.add(panel);

        frame.pack();
        frame.setVisible(true);


}


@Override
```

```
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.contains("SCR_GOAL")) {
        boolean isPlayer = false;
        String s;
        try {
            Player currPlayer = null;
            if (command.contains("HOME")) {
                // checks if no player was selected
                if (homePlayersList.getSelectedIndex
                    () == 0) {
                    s = homeTeam.getName() + "_HAS_
                        SCORED_A_GOAL!";
                    // scores goal and updates
                        timeline and text
                    homeTeam.scoreGoal();
                    game.getTimeLine().writeGoal(game
                        .getGameTimer().getWatchTime()
                        ,0,-1);
                } else {
                    isPlayer = true;
                    s = homePlayersList.
                        getSelectedValue().toString()
                        + "_HAS_SCORED_A_GOAL";
                    game.getTimeLine().writeGoal(game
                        .getGameTimer().getWatchTime()
                        ,0,homePlayersList.
                        getSelectedIndex() - 1);
                    currPlayer = homeTeam.
                        getActivePlayer(
                        homePlayersList.
                        getSelectedIndex() - 1);
                }
                // logs action to timeline
                // sets who the current player is for
                    later in the code
            } else { // if (command.contains("AWAY"))
                {
                if (awayPlayersList.getSelectedIndex
                    () == 0) {
                    s = awayTeam.getName() + "_HAS_
                        SCORED_A_GOAL!";
                    awayTeam.scoreGoal();
                    game.getTimeLine().writeGoal(game
                        .getGameTimer().getWatchTime()
                        ,1,-1);
```

```
                    errorMsg.setText("");
                } else {
                    isPlayer = true;
                    s = awayPlayersList.
                        getSelectedValue().toString()
                        + " HAS SCORED A GOAL";
                    game.getTimeLine().writeGoal(game
                        .getGameTimer().getWatchTime()
                        ,1,awayPlayersList.
                        getSelectedIndex() - 1);
                    currPlayer = awayTeam.
                        getActivePlayer(
                        awayPlayersList.
                        getSelectedIndex() - 1);
                }
            }
            if (isPlayer){
                currPlayer.scoreGoal();
            }
            scoreLabel.setText(homeTeam.getGoals() +
                " - " + awayTeam.getGoals());
            textArea.append("(" + game.getGameTimer()
                .getWatchTime() + ")" + " (SYS): " + s
                +"\n");
            // clears potential error messages
            errorMsg.setText("");
        } catch (NullPointerException ex) {
            errorMsg.setText("Select player");
        }
    }
    if( command.contains("ACT_END_GAME") ){
        finishGame();
    }
    if (command.equals("ACT_CHANGE_POSSESSION")){
        game.changePossession();
        game.getTimeLine().writePossession(game.
            getGameTimer().getWatchTime(),game.
            getPossession());
        String s;
        if (game.getPossession()) {
            s = ("POSSESSION: " + homeTeam.getName())
                ;
        } else {
            s = ("POSSESSION: " + awayTeam.getName())
                ;
        }
```

```
                possessionLabel.setText(s);
                textArea.append("(" + game.getGameTimer().
                    getWatchTime() + ")" + "␣(SYS):␣" + s+"\n"
                    );
            }
            if (command.equals("ACT_SUBMIT")){
                game.getTimeLine().writeComment(game.
                    getGameTimer().getWatchTime(),textField.
                    getText());
                textArea.append("(" + game.getGameTimer().
                    getWatchTime() + ")" + "␣(USR):␣" +
                    textField.getText().toUpperCase()+"\n");
                textField.setText("");
            } if (command.equals("ACT_END_STATIC_GAME")){
                frame.dispose();
            }
            textArea.validate();
            JScrollBar vertical = scrollTextArea.
                getVerticalScrollBar();
            vertical.setValue( vertical.getMaximum() );
        }
}
```

7.2.5 SelectTournamentWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI;

import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.filechooser.FileSystemView;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

public class SelectTournamentWindow extends MyWindow
    implements ActionListener, ListSelectionListener {

    private JComponent selectTournamentWindow;
    private JList tournamentList;
```

```
// Static variables are referenced when creating new
    tournament
private static JLabel currTournament = new JLabel("
    ACTIVE TOURNAMENT: NONE");
private static DefaultListModel tournamentModel = new
     DefaultListModel();

/**
 * Declares the window, super()s variables, embeds it
      in a panel and appends it.
 * @param panel panel which will be displayed
 * @param clw layout to which the panel will be
     appended to
 */
public SelectTournamentWindow(JPanel panel,
    CardLayoutWindow clw) {
    // Panel and Window inherited
    super(panel, clw);
    contentPane = panel;
    setOpaque(true);

    selectTournamentWindow =
        factorySelectTournamentWindow();
    add(selectTournamentWindow);
}

public static JLabel getCurrTournament(){
    return currTournament;
}

public static DefaultListModel getTournamentModel(){
    return tournamentModel;
}



public JComponent factorySelectTournamentWindow(){
    JLabel menuLabel = new JLabel("SELECT TOURNAMENT"
        );

    // Initialises tournamentList with
        tournamentModel
    tournamentList = new JList(tournamentModel);

    // Makes tournamentList scrollable
```

```
JComponent tournamentScrollList = factoryList(
    tournamentList);

// Creates buttons within panes
JComponent buttonBack = factoryButtonPane("..","
    NAV_MAIN");
JComponent buttonSelectTournament =
    factoryButtonPane("Select","
    ACT_SELECT_TOURNAMENT");
JComponent buttonExportTournament =
    factoryButtonPane("Export Tournament","
    ACT_EXPORT_TOURNAMENT");
JComponent buttonImportTournament =
    factoryButtonPane("Import Tournament", "
    ACT_IMPORT_TOURNAMENT");


// Creates GridBagLayout panel
JPanel panel = new JPanel();
panel.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints()
    ;

// Sets the padding
gbc.insets = new Insets(3,3,3,3);

// Adds components
gbc.gridx = 0; gbc.gridy = 0;
panel.add(menuLabel,gbc);

// Increments Y grid
gbc.gridy++;
panel.add(buttonBack,gbc);

gbc.gridy++;
panel.add(currTournament,gbc);

gbc.gridy++;
panel.add(tournamentScrollList,gbc);

gbc.gridy++;
panel.add(buttonSelectTournament,gbc);

gbc.gridy++;
panel.add(buttonExportTournament,gbc);
```

```
        gbc.gridy++;
        panel.add(buttonImportTournament,gbc);

        return panel;
    }

    //    @Override is not allowed when implementing
        interface method
    public void valueChanged(ListSelectionEvent
        listSelectionEvent) {

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command.contains("NAV_")){
            // Navigates to requested page
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            cardLayout.show(contentPane, command);
        } else if (command.equals("ACT_SELECT_TOURNAMENT"
            )){
            // Calls static function to set the selected
                tournament as active
            Tournament.selectTournament(tournamentList.
                getSelectedIndex());
        } else if (command.equals("ACT_EXPORT_TOURNAMENT"
            )){
            Tournament tournament = Tournament.
                getTournamentList().get(tournamentList.
                getSelectedIndex());
            Tournament.exportTournament(tournament);
        } else if (command.equals("ACT_IMPORT_TOURNAMENT"
            )){

            // Initialises a new file chooser
            JFileChooser jfc = new JFileChooser(
                FileSystemView.getFileSystemView().
                getDefaultDirectory());
            jfc.setDialogTitle("CHOOSE_TOURNAMENT");

            // Only lets the user select XML files
            jfc.setAcceptAllFileFilterUsed(false);
```

```
                FileNameExtensionFilter filter = new
                    FileNameExtensionFilter("XML Files","xml")
                    ;
                jfc.addChoosableFileFilter(filter);

                int returnValue = jfc.showOpenDialog(null);
                if (returnValue == JFileChooser.
                    APPROVE_OPTION) {
                    File selectedFile = jfc.getSelectedFile();
                    Tournament.importTournament(selectedFile);
                }
            }

        }
}
```

7.2.6 TextPrompt.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI;

// TextPrompt class written by Rob Camick
// https://tips4java.wordpress.com/2009/11/29/text-prompt
    /

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import javax.swing.text.*;

/**
 *  The TextPrompt class will display a prompt over top
    of a text component when
 *  the Document of the text field is empty. The Show
    property is used to
 *  determine the visibility of the prompt.
 *
 *  The Font and foreground Color of the prompt will
    default to those properties
 *  of the parent text component. You are free to change
    the properties after
 *  class construction.
 */
public class TextPrompt extends JLabel
```

```
        implements FocusListener, DocumentListener
{
    public enum Show
    {
        ALWAYS,
        FOCUS_GAINED,
        FOCUS_LOST;
    }

    private JTextComponent component;
    private Document document;

    private Show show;
    private boolean showPromptOnce;
    private int focusLost;

    public TextPrompt(String text, JTextComponent
        component)
    {
        this(text, component, Show.ALWAYS);
    }

    public TextPrompt(String text, JTextComponent
        component, Show show)
    {
        this.component = component;
        setShow( show );
        document = component.getDocument();

        setText( text );
        setFont( component.getFont() );
        setForeground( component.getForeground() );
        setBorder( new EmptyBorder(component.getInsets())
            );
        setHorizontalAlignment(JLabel.LEADING);

        component.addFocusListener( this );
        document.addDocumentListener( this );

        component.setLayout( new BorderLayout() );
        component.add( this );
        checkForPrompt();
    }

    /**
```

```
     *  Convenience method to change the alpha value of
        the current foreground
     *  Color to the specifice value.
     *
     *  @param alpha value in the range of 0 - 1.0.
     */
    public void changeAlpha(long alpha)
    {
        changeAlpha( (int)(alpha * 255) );
    }

    /**
     *  Convenience method to change the alpha value of
        the current foreground
     *  Color to the specifice value.
     *
     *  @param alpha value in the range of 0 - 255.
     */
    public void changeAlpha(int alpha)
    {
        alpha = alpha > 255 ? 255 : alpha < 0 ? 0 : alpha
            ;

        Color foreground = getForeground();
        int red = foreground.getRed();
        int green = foreground.getGreen();
        int blue = foreground.getBlue();

        Color withAlpha = new Color(red, green, blue,
            alpha);
        super.setForeground( withAlpha );
    }

    /**
     *  Convenience method to change the style of the
        current Font. The style
     *  values are found in the Font class. Common values
         might be:
     *  Font.BOLD, Font.ITALIC and Font.BOLD + Font.
        ITALIC.
     *
     *  @param style value representing the the new style
         of the Font.
     */
    public void changeStyle(int style)
    {
```

```
    setFont( getFont().deriveFont( style ) );
}

/**
 *  Get the Show property
 *
 *  @return the Show property.
 */
public Show getShow()
{
    return show;
}

/**
 *  Set the prompt Show property to control when the
    promt is shown.
 *  Valid values are:
 *
 *  Show.AWLAYS (default) - always show the prompt
 *  Show.Focus_GAINED - show the prompt when the
    component gains focus
 *      (and hide the prompt when focus is lost)
 *  Show.Focus_LOST - show the prompt when the
    component loses focus
 *      (and hide the prompt when focus is gained)
 *
 *  @param show a valid Show enum
 */
public void setShow(Show show)
{
    this.show = show;
}

/**
 *  Get the showPromptOnce property
 *
 *  @return the showPromptOnce property.
 */
public boolean getShowPromptOnce()
{
    return showPromptOnce;
}

/**
 *  Show the prompt once. Once the component has
    gained/lost focus
```

```
     *  once, the prompt will not be shown again.
     *
     *  @param showPromptOnce  when true the prompt will
         only be shown once,
     *                         otherwise it will be shown
          repeatedly.
     */
    public void setShowPromptOnce(boolean showPromptOnce)
    {
        this.showPromptOnce = showPromptOnce;
    }

    /**
     *  Check whether the prompt should be visible or not
         . The visibility
     *  will change on updates to the Document and on
         focus changes.
     */
    private void checkForPrompt()
    {
        //  Text has been entered, remove the prompt

        if (document.getLength() > 0)
        {
            setVisible( false );
            return;
        }

        //  Prompt has already been shown once, remove it

        if (showPromptOnce && focusLost > 0)
        {
            setVisible(false);
            return;
        }

        //  Check the Show property and component focus
            to determine if the
        //  prompt should be displayed.

        if (component.hasFocus())
        {
            if (show == Show.ALWAYS
                    ||  show == Show.FOCUS_GAINED)
                setVisible( true );
            else
```

```
                setVisible( false );
        }
        else
        {
            if (show == Show.ALWAYS
                    || show == Show.FOCUS_LOST)
                setVisible( true );
            else
                setVisible( false );
        }
    }

//  Implement FocusListener

    public void focusGained(FocusEvent e)
    {
        checkForPrompt();
    }

    public void focusLost(FocusEvent e)
    {
        focusLost++;
        checkForPrompt();
    }

//  Implement DocumentListener

    public void insertUpdate(DocumentEvent e)
    {
        checkForPrompt();
    }

    public void removeUpdate(DocumentEvent e)
    {
        checkForPrompt();
    }

    public void changedUpdate(DocumentEvent e) {}
}
```

7.3 UI Browse

7.3.1 BrowseGamesWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Browse;

import uk.ac.glos.ct5025.s1804317.footballStats.
    StaticGame;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    PlayGameWindow;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BrowseGamesWindow extends MyWindow
    implements ActionListener, ListSelectionListener {

    private JComponent browseGamesWindow;
    private JList gamesList;

    /**
     * Window to show past played games within the
         tournament
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * */
    public BrowseGamesWindow(JPanel panel,
        CardLayoutWindow clw) {
        super(panel, clw);
        contentPane = panel;

        browseGamesWindow = factoryBrowseGamesWindow();
        add(browseGamesWindow);
    }

    public JComponent factoryBrowseGamesWindow() {
        JLabel menuLabel = new JLabel("SELECT_GAME");
```

```
        gamesList = new JList(Tournament.
            getActiveTournament().getTournamentGamesModel
            ());

        JComponent gamesScrollList = factoryList(
            gamesList);

        JComponent buttonBack = factoryButtonPane("..", "
            NAV_BACK");
        JComponent buttonSelectGame = factoryButtonPane("
            Select", "ACT_SELECT_GAME");
        JComponent buttonPlayGame = factoryButtonPane("
            PLAY_GAME","MKE_PLAY_GAME");

        // Creates GridBagLayout panel
        JPanel panel = new JPanel();
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        // Sets the padding
        gbc.insets = new Insets(3, 3, 3, 3);

        // Adds components
        gbc.gridx = 0;
        gbc.gridy = 0;
        panel.add(menuLabel, gbc);

        // Increments Y grid
        gbc.gridy++;
        panel.add(buttonBack, gbc);

        gbc.gridy++;
        panel.add(gamesScrollList, gbc);

        gbc.gridy++;
        panel.add(buttonSelectGame, gbc);

        gbc.gridy++;
        panel.add(buttonPlayGame,gbc);

        return panel;
    }

    @Override
```

```
    public void valueChanged(ListSelectionEvent
        listSelectionEvent) {

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        CardLayout cardLayout = (CardLayout) contentPane.
            getLayout();
        if (command.equals("NAV_BACK")) {
            // removes this panel
            contentPane.remove(contentPane.getComponents
                ().length-1);
            cardLayout.show(contentPane, "NAV_BROWSE");
        } else if (command.equals("ACT_SELECT_GAME")) {
            // displays new JFrame with game information
            StaticGame game = Tournament.
                getActiveTournament().getGame(gamesList.
                getSelectedIndex());
            displayGameFrame(game);
        } else if (command.equals("MKE_PLAY_GAME")){

            StaticGame game = Tournament.
                getActiveTournament().getGame(gamesList.
                getSelectedIndex());
            PlayGameWindow playGameWindow = new
                PlayGameWindow(contentPane,
                CardLayoutWindow.cardLayoutWindow,game,"
                PLAYBACK");
            playGameWindow.displayGameWindow();
        }
    }
}
```

7.3.2 BrowsePlayersWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Browse;

import uk.ac.glos.ct5025.s1804317.footballStats.Game;
import uk.ac.glos.ct5025.s1804317.footballStats.Player;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
```

```
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    PlayGameWindow;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class BrowsePlayersWindow extends MyWindow
    implements ActionListener, ListSelectionListener {

    private JComponent browsePlayersWindow;
    private JList playersList;
    private JList activePlayersList;
    private Team currTeam;
    private String mode;
    private static Team homeTeam;

    public static Team getHomeTeam(){
        return homeTeam;
    }

    /**
     * Window to show players to select within a team
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * @param team team for which active players are
         shown
     * @param switchMode mode for what to do with the
         player selection within the window
     */
    public BrowsePlayersWindow(JPanel panel,
        CardLayoutWindow clw, Team team, String switchMode
        ) {
        super(panel, clw);
        contentPane = panel;

        // Gets which team to show the players of
        currTeam = team;
        mode = switchMode;
```

```
        browsePlayersWindow =
            factoryBrowseTournamentWindow();
        add(browsePlayersWindow);

    }

    public JComponent factoryBrowseTournamentWindow(){
        JLabel menuLabel = new JLabel("SELECT_PLAYER");

        JLabel currTeamLabel = new JLabel("TEAM:_" +
            currTeam.getName());

        // Initialises playerList with with model of
            players within the team
        playersList = new JList(currTeam.
            getTeamPlayersModel());

        if (mode.equals("HOME") || mode.equals("AWAY")){
            playersList.setSelectionMode(
                ListSelectionModel.
                MULTIPLE_INTERVAL_SELECTION);
        }

        activePlayersList = new JList(currTeam.
            getTeamActivePlayersModel());


        JButton button1 = new JButton("neat");
        button1.setActionCommand("....");
        button1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

            }
        });

        // Makes playerList scrollable
        JComponent playerScrollList = factoryList(
            playersList);

        // select up to 11 players if mode is home or
            away
        JComponent buttonBack = factoryButtonPane("..","
            NAV_CLOSE");
        JComponent buttonSelectPlayer = factoryButtonPane
            ("Select","ACT_SELECT_PLAYER");
```

```
        // Creates GridBagLayout panel
        JPanel panel = new JPanel();
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        // Sets the padding
        gbc.insets = new Insets(3,3,3,3);

        // Adds components
        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(menuLabel,gbc);

        // Increments Y grid
        gbc.gridy++;
        panel.add(buttonBack,gbc);

        gbc.gridy++;
        panel.add(currTeamLabel,gbc);

        gbc.gridy++;
        panel.add(playerScrollList,gbc);

        gbc.gridy++;
        panel.add(buttonSelectPlayer,gbc);

        return panel;


    }

    //    @Override is not allowed when implementing
        interface method
    public void valueChanged(ListSelectionEvent
        listSelectionEvent) {

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        CardLayout cardLayout = (CardLayout) contentPane.
            getLayout();
```

```
if (command.equals("NAV_CLOSE")){
    if (mode.equals("AWAY")){
        homeTeam.resetActivePlayers();
    }
    contentPane.remove(contentPane.getComponents
        ().length-1);
    cardLayout.show(contentPane,"MKE_TEAMS_SELECT
        ");


} else if (command.equals("ACT_SELECT_PLAYER")){
    if(mode.equals("BROWSE")){
        Player player = currTeam.getPlayer(
            playersList.getSelectedIndex());
        displayFrame(player);
    } else if(mode.equals("HOME")){
        int[] players = playersList.
            getSelectedIndices();
        for (int i : players){
            currTeam.addActivePlayer(i);
        }
        homeTeam = currTeam;
        BrowseTeamsSelectWindow
            browseTeamsSelectWindow = new
            BrowseTeamsSelectWindow(contentPane,
            CardLayoutWindow.cardLayoutWindow,"
            AWAY");
        contentPane.add(browseTeamsSelectWindow,
            command);
        cardLayout.show(contentPane,command);

    } else if(mode.equals("AWAY")){
        int[] players = playersList.
            getSelectedIndices();
        for (int i : players){
            currTeam.addActivePlayer(i);
        }
        Game game = new Game(BrowsePlayersWindow.
            getHomeTeam(),currTeam);
        game.initialiseGame();
        PlayGameWindow playGameWindow = new
            PlayGameWindow(contentPane,
            CardLayoutWindow.cardLayoutWindow,
            BrowsePlayersWindow.getHomeTeam(),
            currTeam,game,"RECORD");
        playGameWindow.displayGameWindow();
```

```
                contentPane.remove(contentPane.
                    getComponents().length-1);
                cardLayout.show(contentPane,"NAV_MAIN");
            }

        }

    }

}
```

7.3.3 BrowseTeamsSelectWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Browse;

import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.Create
    .CreatePlayerWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BrowseTeamsSelectWindow extends
    BrowseTeamsWindow implements ActionListener {

    private String mode;

    /**
     * Window to select team from active tournament for
         other purposes than showing team deatails
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * @param switchMode chooses the purpose of the
         creation of the window
     */
    public BrowseTeamsSelectWindow(JPanel panel,
        CardLayoutWindow clw, String switchMode) {
```

```
        super(panel, clw);
        mode = switchMode;


    }

    @Override
    public void actionPerformed(ActionEvent e) {
        boolean error = false;
        String command = e.getActionCommand();

        if (command.equals("NAV_BROWSE")){
            // Removes content pane to minimise memory
                usage, user goes back to 'Browse' menu
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            contentPane.remove(contentPane.getComponents
                ().length-1);
            cardLayout.show(contentPane,command);
        }
        else if (command.contains("NAV_")){
            // Navigates to requested page
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            cardLayout.show(contentPane, command);
        } else if (command.equals("MKE_SELECT")){
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            int teamIndex = getTeamsList().
                getSelectedIndex();
            Team team = Tournament.getActiveTournament().
                getTeam(teamIndex);
            if (mode.equals("BROWSE")) {
                BrowsePlayersWindow browsePlayersWindow =
                     new BrowsePlayersWindow(contentPane,
                    CardLayoutWindow.cardLayoutWindow,
                    team,"BROWSE");
                contentPane.add(browsePlayersWindow, "
                    MKE_BROWSE_PLAYERS");
                cardLayout.show(contentPane, "
                    MKE_BROWSE_PLAYERS");
            } else if(mode.equals("CREATE")){
                CreatePlayerWindow createPlayerWindow =
                    new CreatePlayerWindow(contentPane,
                    CardLayoutWindow.cardLayoutWindow,
                    team);
```

```
                contentPane.add(createPlayerWindow,"
                    MKE_CREATE_PLAYER");
                cardLayout.show(contentPane, "
                    MKE_CREATE_PLAYER");
            } else if(mode.equals("HOME")){
                BrowsePlayersWindow browsePlayersWindow =
                     new BrowsePlayersWindow(contentPane,
                    CardLayoutWindow.cardLayoutWindow,team
                    ,"HOME");
                contentPane.add(browsePlayersWindow,"
                    MKE_BROWSE_PLAYERS");
                cardLayout.show(contentPane,"
                    MKE_BROWSE_PLAYERS");
            } else if(mode.equals("AWAY")){
                if (BrowsePlayersWindow.getHomeTeam() ==
                    team){
                    getErrorLabel().setText("Home Team
                        and Away Team Can't Be the Same");
                    error = true;
                } else {
                    BrowsePlayersWindow
                        browsePlayersWindow = new
                        BrowsePlayersWindow(contentPane,
                        CardLayoutWindow.cardLayoutWindow,
                         team, "AWAY");
                    contentPane.add(browsePlayersWindow,
                        "MKE_BROWSE_PLAYERS");
                    cardLayout.show(contentPane, "
                        MKE_BROWSE_PLAYERS");
                }
            }
            if (!error){
                getErrorLabel().setText("");
            }

        }

    }
}
```

7.3.4 BrowseTeamsSelectWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Browse;
```

```
import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BrowseTeamsWindow extends MyWindow
    implements ActionListener, ListSelectionListener {

    private JComponent browseTeamsWindow;
    private JList teamsList;
    private JLabel errorLabel;

    /**
     * Window to select a team from the active tournament
         and view its stats
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * */
    public BrowseTeamsWindow(JPanel panel,
        CardLayoutWindow clw) {
        super(panel, clw);
        contentPane = panel;
        setOpaque(true);

        browseTeamsWindow = factoryBrowseTeamsWindow();
        add(browseTeamsWindow);
    }

    public JLabel getErrorLabel(){
        return errorLabel;
    }

    public JComponent factoryBrowseTeamsWindow(){
        JLabel menuLabel = new JLabel("SELECT TEAM");
```

```
        errorLabel = new JLabel("");

        teamsList = new JList(Tournament.
            getActiveTournament().getTournamentTeamsModel
            ());

        JComponent teamsScrollList = factoryList(
            teamsList);

        JComponent buttonBack = factoryButtonPane("..","
            NAV_BROWSE");
        JComponent buttonSelectTeam = factoryButtonPane("
            Select","MKE_SELECT");

        // Creates GridBagLayout panel
        JPanel panel = new JPanel();
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        // Sets the padding
        gbc.insets = new Insets(3,3,3,3);

        // Adds components
        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(menuLabel,gbc);

        // Increments Y grid
        gbc.gridy++;
        panel.add(buttonBack,gbc);

        gbc.gridy++;
        panel.add(teamsScrollList,gbc);

        gbc.gridy++;
        panel.add(buttonSelectTeam,gbc);

        gbc.gridy++;
        panel.add(errorLabel,gbc);

        return panel;
    }

//    @Override is not allowed when implementing
    interface method
```

```
    public void valueChanged(ListSelectionEvent
        listSelectionEvent) {

    }

    public JList getTeamsList(){
        return teamsList;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command.equals("NAV_BROWSE")){
            // Removes content pane to minimise memory
                usage, user goes back to 'Browse' menu
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            contentPane.remove(contentPane.getComponents
                ().length-1);
            cardLayout.show(contentPane,command);
        }
        else if (command.contains("NAV_")){
            // Navigates to requested page
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            cardLayout.show(contentPane, command);
        } else if (command.equals("MKE_SELECT")){
//              CardLayout cardLayout = (CardLayout)
    contentPane.getLayout();
            Team team = Tournament.getActiveTournament().
                getTeam(teamsList.getSelectedIndex());
            displayFrame(team);
//              BrowsePlayersWindow browsePlayersWindow =
    new BrowsePlayersWindow(contentPane, CardLayoutWindow.
    cardLayoutWindow, team);
//              contentPane.add(browsePlayersWindow,"
    MKE_BROWSE_PLAYERS");
//              cardLayout.show(contentPane,"
    MKE_BROWSE_PLAYERS");
            }

    }
}
```

7.3.5 BrowseTournamentWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Browse;

import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class BrowseTournamentWindow extends MyWindow
    implements ActionListener {

    private Point lastLocation = null;

    /**
     * Window used to select active tournament, import
        and export tournament
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
        appended to
     * */
    public BrowseTournamentWindow(JPanel panel,
        CardLayoutWindow clw) {
        super(panel, clw);
    }

    public void displayBrowseTournamentWindow(){
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(WindowConstants.
            DISPOSE_ON_CLOSE);

        //Set window location.
        if (lastLocation != null) {
            //Move the window over and down 40 pixels.
            lastLocation.translate(40, 40);
            if ((lastLocation.x > MyWindow.getSizeX() ||
                (lastLocation.y > MyWindow.getSizeY()))) {
```

```
            lastLocation.setLocation(0, 0);
        }
        frame.setLocation(lastLocation);
    } else {
        lastLocation = frame.getLocation();
    }

    JLabel menuLabel = new JLabel(Tournament.
        getActiveTournament().getTournamentName());

    JComponent buttonBack = factoryButtonPane("..", "
        NAV_CLOSE");

    JTable tournamentTable = new JTable(new
        CardLayoutWindow.MyTableModel());
    tournamentTable.setAutoCreateRowSorter(true);

    ArrayList<Team> teamList = Tournament.
        getActiveTournament().getTournamentTeams();
    Object rowData[] = new Object[9];

    tournamentTable.
        setPreferredScrollableViewportSize(new
        Dimension(700, 150));
    tournamentTable.setFillsViewportHeight(true);

    //Create the scroll pane and add the table to it.
    JScrollPane scrollPane = new JScrollPane(
        tournamentTable);

    // Creates GridBagLayout panel
    JPanel panel = new JPanel();
    panel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints()
        ;

    // Sets the padding
    gbc.insets = new Insets(3,3,3,3);

    // Adds components
    gbc.gridx = 0; gbc.gridy = 0;
    panel.add(menuLabel,gbc);

    // Increments Y grid
    gbc.gridy++;
    panel.add(buttonBack,gbc);
```

```
            gbc.gridy = 0; gbc.gridx++;
            gbc.gridheight = 2;
            gbc.fill = GridBagConstraints.HORIZONTAL;
            panel.add(scrollPane,gbc);

            frame.add(panel);

            frame.pack();
            frame.setVisible(true);
        }
}
```

7.3.6 BrowseWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Browse;

import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BrowseWindow extends MyWindow implements
    ActionListener {

    private JComponent browseWindow;
    private JLabel errorLabel;

    /**
     * Window to show which items the user can browse,
         navigated to from the main menu
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * */
    public BrowseWindow(JPanel panel, CardLayoutWindow
        clw) {
        super(panel, clw);
        contentPane = panel;
```

```
        setOpaque(true);

        browseWindow = createBrowseSection();
        add(browseWindow);
    }

    public JComponent createBrowseSection() {
        JLabel menuLabel = new JLabel("BROWSE");
        errorLabel = new JLabel("");

        JComponent buttonBack = factoryButtonPane("..", "
            NAV_MAIN");
        JComponent buttonBrowseGames = factoryButtonPane(
            "Games", "MKE_BROWSE_GAMES");
        JComponent buttonBrowsePlayers =
            factoryButtonPane("Players","MKE_TEAMS_SELECT"
            );
        JComponent buttonBrowseTeams = factoryButtonPane(
            "Teams", "MKE_TEAMS_BROWSE");
        JComponent buttonBrowseTournament =
            factoryButtonPane("Tournament", "
            MKE_BROWSE_TOURNAMENT");



        JPanel panel = new JPanel();

        // GridBagLayout is not needed for a simple menu,
             but it was used for consistency throughout
            the program
        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        // Sets the padding
        gbc.insets = new Insets(3,3,3,3);

        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(menuLabel,gbc);

        gbc.gridy++;
        panel.add(buttonBack,gbc);

        gbc.gridy++;
        panel.add(buttonBrowseTournament,gbc);
```

```
        gbc.gridy++;
        panel.add(buttonBrowseTeams,gbc);

        gbc.gridy++;
        panel.add(buttonBrowsePlayers,gbc);

        gbc.gridy++;
        panel.add(buttonBrowseGames,gbc);

        gbc.gridy++;
        panel.add(errorLabel,gbc);

        return panel;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        try {
            if (command.contains("NAV_")) {
                CardLayout cardLayout = (CardLayout)
                    contentPane.getLayout();
                cardLayout.show(contentPane, command);
            } else if (command.contains("MKE_TEAMS")) {
                CardLayout cardLayout = (CardLayout)
                    contentPane.getLayout();
                // Creates new window to show teams for
                    current
                BrowseTeamsWindow browseTeamsWindow;
                if (command.contains("SELECT")) {
                    browseTeamsWindow = new
                        BrowseTeamsSelectWindow(
                        contentPane, CardLayoutWindow.
                        cardLayoutWindow, "BROWSE");
                } else { // if(command.contains("BROWSE")
                    ){
                    browseTeamsWindow = new
                        BrowseTeamsWindow(contentPane,
                        CardLayoutWindow.cardLayoutWindow)
                        ;
                }
                contentPane.add(browseTeamsWindow,
                    command);
                cardLayout.show(contentPane, command);
            } else if (command.contains("MKE_BROWSE_GAMES
                ")) {
```

```
                CardLayout cardLayout = (CardLayout)
                    contentPane.getLayout();
                BrowseGamesWindow browseGamesWindow = new
                     BrowseGamesWindow(contentPane,
                    CardLayoutWindow.cardLayoutWindow);
                contentPane.add(browseGamesWindow,
                    command);
                cardLayout.show(contentPane, command);
            } else if (command.equals("
                MKE_BROWSE_TOURNAMENT")) {
                BrowseTournamentWindow
                    browseTournamentWindow = new
                    BrowseTournamentWindow(contentPane,
                    CardLayoutWindow.cardLayoutWindow);
                browseTournamentWindow.
                    displayBrowseTournamentWindow();
            }
            errorLabel.setText("");
        } catch (NullPointerException ex) {
            errorLabel.setText("Please␣Create␣a␣
                Tournament");
        }
    }

}
```

7.4 Create UI

7.4.1 CreatePlayerWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Create;

import uk.ac.glos.ct5025.s1804317.footballStats.Player;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
public class CreatePlayerWindow extends MyWindow
    implements ActionListener {

    private JComponent createPlayerWindow;
    private JTextField textFieldPlayerName;
    private Team currTeam;

    /**
     * Window used to create player within a team
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * @param team team to which the new player will be
         appended to
     */
    public CreatePlayerWindow(JPanel panel,
        CardLayoutWindow clw, Team team) {
        super(panel, clw);

        setOpaque(true);

        currTeam = team;

        createPlayerWindow = factoryPlayerCreateWindow();
        add(createPlayerWindow);
    }

    public JComponent factoryPlayerCreateWindow(){
        JLabel menuLabel = new JLabel("CREATE_PLAYER");

        JComponent buttonBack = factoryButtonPane("..", "
            NAV_CREATE");
        JComponent buttonCreate = factoryButtonPane("
            Create", "ACT_CREATE_TEAM");
        textFieldPlayerName = factoryTextField("Player_
            Name");

        JPanel panel = new JPanel();

        panel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
            ;

        gbc.insets = new Insets(3, 3, 3, 3);

        gbc.gridx = 0;
```

```
        gbc.gridy = 0;
        panel.add(menuLabel, gbc);

        gbc.gridy++;
        panel.add(buttonBack, gbc);

        gbc.gridy++;
        panel.add(textFieldPlayerName, gbc);

        gbc.gridy++;
        panel.add(buttonCreate, gbc);

        gbc.gridy++;
        panel.add(getErrorMessage(), gbc);

        return panel;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command.contains("NAV_")){
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            cardLayout.show(contentPane, command);
        } else if (command.equals("ACT_CREATE_TEAM")){
            if(!textFieldPlayerName.getText().equals(""))
                {
                Player player = new Player(
                    textFieldPlayerName.getText().
                    toUpperCase(),"",currTeam);
                player.addToTeam();
                textFieldPlayerName.setText(null);
            }
        }

    }
}
```

7.4.2 CreateTeamWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Create;
```

```
import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CreateTeamWindow extends MyWindow implements
     ActionListener {

    private JComponent createTeamWindow;
    private JTextField textFieldTeam;
    private JLabel errorLabel;

    /**
     * Window used to create new team within a tournament
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * */
    public CreateTeamWindow(JPanel panel,
        CardLayoutWindow clw) {
        super(panel, clw);
        setOpaque(true);

        createTeamWindow = factoryTeamCreateWindow();
        add(createTeamWindow);
    }

    public JComponent factoryTeamCreateWindow(){
        JLabel menuLabel = new JLabel("CREATE_TEAM");

        errorLabel = new JLabel("");

        JComponent buttonBack = factoryButtonPane("..", "
            NAV_CREATE");
        JComponent buttonCreate = factoryButtonPane("
            Create", "ACT_CREATE_TEAM");
        textFieldTeam = factoryTextField("Team_Name");
```

```
    JPanel panel = new JPanel();

    panel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints()
        ;

    gbc.insets = new Insets(3, 3, 3, 3);

    gbc.gridx = 0;
    gbc.gridy = 0;
    panel.add(menuLabel, gbc);

    gbc.gridy++;
    panel.add(buttonBack, gbc);

    gbc.gridy++;
    panel.add(textFieldTeam, gbc);

    gbc.gridy++;
    panel.add(buttonCreate, gbc);

    gbc.gridy++;
    panel.add(getErrorMessage(), gbc);

    gbc.gridy++;
    panel.add(errorLabel,gbc);

    return panel;
}

@Override
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if (command.contains("NAV_")){
        CardLayout cardLayout = (CardLayout)
            contentPane.getLayout();
        cardLayout.show(contentPane, command);
        errorLabel.setText("");
    } else if (command.equals("ACT_CREATE_TEAM")){
        if(!textFieldTeam.getText().equals("")) {
            try {
                Tournament.getActiveTournament().
                    addTeam(new Team(textFieldTeam.
                    getText().toUpperCase()));
                textFieldTeam.setText(null);
```

```
                    errorLabel.setText("");
                } catch (NullPointerException ex){
                    errorLabel.setText("Please Create a
                        Tournament");
                }
            } else {
                errorLabel.setText("Please Enter a Name")
                    ;
            }
        }

    }
}
```

7.4.3 CreateTournamentWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Create;

import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    SelectTournamentWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CreateTournamentWindow extends MyWindow
    implements ActionListener {

    private JComponent createTournamentWindow;
    private JTextField textFieldTournament;

    /**
     * Window to create new tournament
     * @param panel panel which will be displayed
     * @param clw layout to which the panel will be
         appended to
     * */
```

```
public CreateTournamentWindow(JPanel panel,
    CardLayoutWindow clw) {
    super(panel, clw);
    contentPane = panel;
    setOpaque(true);

    createTournamentWindow =
        factoryTournamentCreateWindow();
    add(createTournamentWindow);
}

public JComponent factoryTournamentCreateWindow() {
    JLabel menuLabel = new JLabel("CREATE_TOURNAMENT"
        );

    JComponent buttonBack = factoryButtonPane("..", "
        NAV_CREATE");
    JComponent buttonCreate = factoryButtonPane("
        Create", "ACT_CREATE_TOURNAMENT");
    textFieldTournament = factoryTextField("
        Tournament_Name");

    JPanel panel = new JPanel();

    panel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints()
        ;

    gbc.insets = new Insets(3, 3, 3, 3);

    gbc.gridx = 0;
    gbc.gridy = 0;
    panel.add(menuLabel, gbc);

    gbc.gridy++;
    panel.add(buttonBack, gbc);

    gbc.gridy++;
    panel.add(textFieldTournament, gbc);

    gbc.gridy++;
    panel.add(buttonCreate, gbc);

    return panel;
}
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command.contains("NAV_")){
            CardLayout cardLayout = (CardLayout)
                contentPane.getLayout();
            cardLayout.show(contentPane, command);
        } else if (command.equals("ACT_CREATE_TOURNAMENT"
            )){
            if(!textFieldTournament.getText().equals(""))
                {
                Tournament.createTournament(
                    textFieldTournament.getText().
                    toUpperCase());
                textFieldTournament.setText(null);
            }
        }

    }
}
```

7.4.4 CreateWindow.java

```
package uk.ac.glos.ct5025.s1804317.footballStats.UI.
    Create;

import uk.ac.glos.ct5025.s1804317.footballStats.UI.Browse
    .BrowseTeamsSelectWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    CardLayoutWindow;
import uk.ac.glos.ct5025.s1804317.footballStats.UI.
    MyWindow;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CreateWindow extends MyWindow implements
    ActionListener {

    private JComponent createWindow;
    private JLabel errorLabel;
```

```
/**
 * Window browsing the 'Create' section, navigated to
     from the main menu
 * @param panel panel which will be displayed
 * @param clw layout to which the panel will be
    appended to
 * */
public CreateWindow(JPanel panel, CardLayoutWindow
    clw) {
    super(panel, clw);
    contentPane = panel;
    setOpaque(true);

    createWindow = createCreateWindow();
    add(createWindow);
}

private JComponent createCreateWindow() {

    // Title label
    JLabel menuLabel = new JLabel("CREATE");

    errorLabel = new JLabel("");

    // Creates identical buttons with a title and a
       command to be executed when the button is
       clicked
    JComponent buttonBack = factoryButtonPane("..","
       NAV_MAIN");
    JComponent buttonCreateTournament =
       factoryButtonPane("Tournament", "
       NAV_CREATE_TOURNAMENT");
    JComponent buttonCreateTeam = factoryButtonPane("
       Team","NAV_CREATE_TEAM");
    JComponent buttonCreatePlayer = factoryButtonPane
       ("Player","MKE_PLAYER_SELECT_TEAM");

    // Creates a new panel to append buttons to
    JPanel panel = new JPanel();

    // GridBagLayout is not needed for a simple menu,
        but it was used for consistency throughout
       the program
    panel.setLayout(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints()
       ;
```

```
        // Sets the padding
        gbc.insets = new Insets(3,3,3,3);

        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(menuLabel,gbc);

        gbc.gridy++;
        panel.add(buttonBack,gbc);

        gbc.gridy++;
        panel.add(buttonCreateTournament,gbc);

        gbc.gridy++;
        panel.add(buttonCreateTeam,gbc);

        gbc.gridy++;
        panel.add(buttonCreatePlayer,gbc);

        gbc.gridy++;
        panel.add(errorLabel,gbc);

        return panel;
    }

    @Override
    public void actionPerformed(ActionEvent e){
        String command = e.getActionCommand();
        CardLayout cardLayout = (CardLayout) contentPane.
            getLayout();
        try {

            if (command.contains("NAV_")) {
                cardLayout.show(contentPane, command);
            } else if (command.equals("
                MKE_PLAYER_SELECT_TEAM")) {
                BrowseTeamsSelectWindow
                    browseTeamsSelectWindow = new
                    BrowseTeamsSelectWindow(contentPane,
                    CardLayoutWindow.cardLayoutWindow,"
                    CREATE");
                contentPane.add(browseTeamsSelectWindow,
                    command);
                cardLayout.show(contentPane,command);
            }
            errorLabel.setText("");
```

```
        } catch (NullPointerException ex){
            errorLabel.setText("Please Create a
                Tournament");
        }
    }
}
```

8 Appendix 3

Test suites

8.0.1 EntryTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.Entry;

import static org.junit.Assert.assertEquals;

public class EntryTest {

    @Test
    void testEntry(){
        String[] data = new String[]{"user comment"};
        Entry entry = new Entry("15:50","COMMENT",data,"
            USER_COMMENT");
        assertEquals("15:50",entry.getTime());
        assertEquals("COMMENT",entry.getAction());
        assertEquals(data,entry.getData());
        assertEquals("USER_COMMENT",entry.getOutput());
    }

}
```

8.0.2 GameTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.Game;
import uk.ac.glos.ct5025.s1804317.footballStats.Player;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;

import static org.junit.Assert.assertEquals;

public class GameTest {
```

```
Team team0;
Team team1;
Game game;

@org.junit.jupiter.api.BeforeEach
void setUp() {
    team0 = new Team("Team Zero");
    team1 = new Team("Team One");
}

@org.junit.jupiter.api.AfterEach
void tearDown() {

}

@Test
void possessionChange(){
    game = new Game(team0, team1);
    game.changePossession();
    assertEquals(false,game.getPossession());
}

@Test
void multipleGamesPlayed(){
    game = new Game(team0, team1);
    team0.scoreGoal();
    team0.scoreGoal();
    team1.scoreGoal();
    // team0 scores two goals
    assertEquals(2,team0.getGoals());
    game.finishGame();
    game.initialiseGame();
    team0.scoreGoal();
    // team0 scores one goal since initialised
    assertEquals(1,team0.getGoals());
    team1.scoreGoal();
    game.finishGame();
    // team0 scored three goals over two games
    assertEquals(3,team0.getGoalsFor());
    // goal difference of one
    assertEquals(1,team0.getGoalsDifference());
    // game won (+3 pts) and game drawn (+1 pts) = 4
        pts
    assertEquals(4,team0.getPoints());
    // team1 has lost one game
    assertEquals(1,team1.getGamesLost());
```

```
    }

    @Test
    void activePlayerRetention(){
        Player player = new Player("Player","",team0);
        player.addToTeam();

        team0.addActivePlayer(0);

        game = new Game(team0, team1);
        game.finishGame();
        // active players reset
        assertEquals(0,team0.getActivePlayers().size());
    }

}
```

8.0.3 GameTimerTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.Game;
import uk.ac.glos.ct5025.s1804317.footballStats.GameTimer
    ;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;

import java.util.concurrent.TimeUnit;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

public class GameTimerTest {

    GameTimer gameTimer;

    @org.junit.jupiter.api.AfterEach
    void tearDown() {

    }

    @Test
    void testGameTimer() {
        gameTimer = new GameTimer();
        try {
            TimeUnit.SECONDS.sleep(3);
```

```
            assertEquals(3, gameTimer.getStopWatchTime(),
                0.1);
        } catch (InterruptedException e){
            fail("Test Interrupted");
            e.printStackTrace();
        }
    }


}
```

8.0.4 ItemTest.java

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.
    assertEquals;

public class ItemTest {


    ItemMock item;
    @org.junit.jupiter.api.BeforeEach
    void setUp(){
        item = new ItemMock("Item");

    }

    @Test
    void getName(){
        assertEquals("Item",item.getName());
    }

    @Test
    void increments(){
        item.incrementGoalsFor();
        item.incrementGoals();
        item.incrementGamesWon();
        item.incrementGamesLost();
        item.incrementGamesDrawn();
        assertEquals(1,item.getGoalsFor());
        assertEquals(1,item.getGoals());
        assertEquals(1,item.getGamesWon());
        assertEquals(1,item.getGamesLost());
        assertEquals(1,item.getGamesDrawn());
```

```
    }

}
```

8.0.5 PlayerTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.Player;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;

import static org.junit.jupiter.api.Assertions.*;

public class PlayerTest {
    Player player;
    Team team;

    @org.junit.jupiter.api.BeforeEach
    void setUp(){
        team = new Team("Hull");
        player = new Player("Harry","19/02/1993",team);
    }

    @org.junit.jupiter.api.AfterEach
    void tearDown(){

    }

    @Test
    void getGoals(){
        assertEquals(0,player.getGoals());
    }

    @Test
    void getDoB(){
        assertEquals("19/02/1993",player.getPlayerDoB());
    }

    @Test
    void scoreGoal(){
        player.scoreGoal();
        assertEquals(1,player.getGoals());
        assertEquals(1,player.getGoalsFor());
        assertEquals(1,team.getGoals());
    }
```

```
    @Test
    void addToTeam(){
        player.addToTeam();
        assertEquals(1,team.getTeamPlayers().size());
    }
}
```

8.0.6 TeamTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.Player;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.
    Tournament;

import static org.junit.jupiter.api.Assertions.
    assertEquals;

public class TeamTest {

    Player player;
    Team team;
    Tournament tournament;

    @org.junit.jupiter.api.BeforeEach
    void setUp(){
        team = new Team("Hull");
        player = new Player("Harry","19/02/1993",team);
        tournament = new Tournament("League Tournament");
        player.addToTeam();

    }

    @org.junit.jupiter.api.AfterEach
    void tearDown() {

    }


    @Test
    void getPlayer(){
        // player name is in list model
        assertEquals(1,team.getTeamPlayersModel().size())
            ;
```

```
        // player object is in team players
        assertEquals(1,team.getTeamPlayers().size());

    }

    @Test
    void getActivePlayer(){
        // player is listed as an active player
        team.addActivePlayer(0);
        assertEquals(1,team.getActivePlayers().size());
        assertEquals(1, team.getActivePlayers().size());
        team.resetActivePlayers();
        assertEquals(0,team.getActivePlayers().size());
    }

    @Test
    void scoreGoal(){
        team.scoreGoal();
        assertEquals(1,team.getGoals());
        // further tests aren't done here as they do not
            include public functions
    }

    @Test
    void gamesPlayed(){
        team.gameWon();
        team.gameLost();
        team.gameDrawn();
        assertEquals(1,team.getGamesWon());
        assertEquals(1,team.getGamesLost());
        assertEquals(1,team.getGamesDrawn());
        // tournament points, 3 from win 1 from draw
        assertEquals(4,team.getPoints());

    }

}
```

8.0.7 TimelineTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.Game;
import uk.ac.glos.ct5025.s1804317.footballStats.Player;
import uk.ac.glos.ct5025.s1804317.footballStats.Team;
import uk.ac.glos.ct5025.s1804317.footballStats.Timeline;
```

```
import static org.junit.Assert.assertEquals;

public class TimelineTest {
    Team team0;
    Team team1;
    Game game;
    Timeline timeline;

    @org.junit.jupiter.api.BeforeEach
    void setUp(){
        team0 = new Team("Team Zero");
        team1 = new Team("Team One");
        game = new Game(team0,team1);
        // timeline initiated during game constructor
        timeline = game.getTimeLine();
    }

    @org.junit.jupiter.api.AfterEach
    void tearDown(){

    }

    @Test
    void possessionWithAwayEnd(){
        // adds timeline events
        timeline.writePossession("00:00", true);
        timeline.writePossession("00:10", false);
        timeline.writePossession("00:25", true);
        timeline.writePossession("00:30", false);
        // ends game
        timeline.writeEndGame("00:40",false);
        // 15/40 == 0.375
        assertEquals(0.375f,timeline.calculatePossession
            (),0.0002);
    }

    @Test
    void possessionWithHomeEnd(){
        timeline.writePossession("00:00", true);
        timeline.writePossession("00:10", false);
        timeline.writePossession("00:20", true);
        timeline.writeEndGame("00:40",true);
        // 30/40 == 0.75
        assertEquals(0.75f,timeline.calculatePossession()
            ,0.0002);
```

```
    }


}
```

8.0.8 TournamentTest.java

```
import org.junit.jupiter.api.Test;
import uk.ac.glos.ct5025.s1804317.footballStats.*;

import static org.junit.Assert.assertEquals;

public class TournamentTest {

    @org.junit.jupiter.api.BeforeEach
    void setUp(){
        Tournament.createTournament("Tournament Zero");
    }

    @org.junit.jupiter.api.AfterEach
    void tearDown(){

    }

    @Test
    void getActiveTournament(){
        assertEquals("Tournament Zero", Tournament.
            getActiveTournament().getTournamentName());
        Tournament.createTournament("Tournament One");
        // Previous tournament remains active
        assertEquals("Tournament Zero", Tournament.
            getActiveTournament().getTournamentName());
        Tournament.selectTournament(1);
        // Changed active tournament
        assertEquals("Tournament One", Tournament.
            getActiveTournament().getTournamentName());
    }

    @Test
    void addElements(){
        Tournament tournament = Tournament.
            getActiveTournament();
        Team team0 = new Team("Team Zero");
        tournament.addTeam(team0);
```

```
        // team added correctly
        assertEquals(team0,tournament.getTeam(0));
        Team team1 = new Team("Team One");
        Game game = new Game(team0,team1);
        StaticGame staticGame = new StaticGame(game);
        tournament.addGame(staticGame);
        // game added correctly
        assertEquals(staticGame,tournament.getGame(0));
    }
}
```