Grade 12
Key 3

PAT Technical Document

Five a Side Football

Nikhar Ramlakhan
TRINITYHOUSE HIGH SCHOOL 2020

Table of Contents

Working Class Code

Database Class

```
package ramlakhan;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.swing.JOptionPane;

public class Database {

    //Variables required for Database object:
    private Connection connection;

    /*
    Database Constructor:
    * Connect with the database 'Five a Side Football' to allow for data transfer
      into the program.
    */
    public Database(){
        try{
            connection = DriverManager.getConnection
            ("jdbc:ucanaccess://Five_a_Side_Football.accdb");
        }//try
        catch (Exception e){
            System.out.println ("Connection NOT successful \n");
            e.printStackTrace();
        }//end try-catch
    }//Database Object

    /*
    getConnection Accessor Method:
    * Return the connection of the database which allows the database to be
      accessed from other classes.
    */
    public Connection getConnection (){
        return connection;
    }//getConnection

    /*
    runTeamInsertQuery Mutator Method:
    * Adds the User Team's details to the Teams Table in the Database.
    - Receives a Team object.
    - Creates a string with the values that will be transfered to the database.
    - Creates the query to write to the database.
    - Indicates if the transfer was successful or not.
    */
    public void runTeamInsertQuery (Team user){
        String values;
        values = "'" + user.getUsername() + "', '" + user.getTeamName() + "', '"
                                    + user.getTournamentName() + "', '"
                                    + user.getGK().getPlayerID()+ "', '"
```

```
                                    + user.getCB().getPlayerID() + "', '"
                                    + user.getCDM().getPlayerID() + "', '"
                                    + user.getCAM().getPlayerID() + "', '"
                                    + user.getST().getPlayerID() + "'";

        int appended; //number of rows appended.
        try {
            Statement s = connection.createStatement();
            String qry = "INSERT INTO Teams (Username, [Team Name], "
                    + "[Tournament Name], [GK ID], [CB ID], [CDM ID], [CAM ID], "
                    + "[ST ID]) VALUES (" + values + ");";

            appended = s.executeUpdate(qry);
            System.out.print(appended);
            s.close();
            JOptionPane.showMessageDialog(null, "Your team has successfully
saved!",
            "SAVED!", JOptionPane.INFORMATION_MESSAGE);
        }//try
        catch (Exception e){
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "There was a problem saving your"
            + "team!\nPlease contact the developer.", "TEAM WAS NOT SAVED!",
            JOptionPane.ERROR_MESSAGE);
        }//end try-catch
    }//runTeamInsertQuery

    /*
    getTop1StringRecord Accessor Method:
    * Used to access a single string value from the database.
    * Primarily used to get the most recent AutoNumber assigned with the
      creation of a new Team record.
    - Receives the field, table, where and order statements of the field
      required to be returned.
    - Runs a query in the database based on the parameters.
    - Gets the cell when the value has been found.
    - A statement is printed to indicate if the value was successfully accessed.
    */
    public String getTop1StringRecord (String field, String table, String where,
String order){
        String result = new String ();

        try {
            Statement s = connection.createStatement();
            String qry = "SELECT TOP 1 [" + field + "] FROM " + table +
                    " WHERE " + where + "ORDER BY [" + field + "] " + order;
            ResultSet rs = s.executeQuery(qry);
            while (rs.next()){
                result = rs.getString(field);
                System.out.print(result);
            }//end while loop
            rs.close();
            s.close();
            System.out.println (field + " was successfully accessed by the
database!");
```

```
        }//try
        catch (Exception e){
            e.printStackTrace();
            System.out.println (field + " was NOT accessed by the database!");
        }//end try-catch
        return result;
    }//getTop1StringRecord

    /*
    addOpponentsToDatabase Mutator Method:
    * Adds the Points acquired by the 5 opposition teams to the Opponents Table
      in the Database.
    - Receives a Rank Object.
    - Creates a string with the values that will be transfered to the database.
    - Creates the query to write to the database.
    - Indicates if the transfer was successful or not.
    */
    public void addOpponentsToDatabase (Rank tournament){
        String values = "'" + tournament.getTournamentID() + "', "
                                    + tournament.getTeamPoints(1) + ", "
                                    + tournament.getTeamPoints(2) + ", "
                                    + tournament.getTeamPoints(3) + ", "
                                    + tournament.getTeamPoints(4) + ", "
                                    + tournament.getTeamPoints(5);

        int appended; //number of rows appended.
        try {
            Statement s = connection.createStatement();
            String qry = "INSERT INTO Opponents ([Tournament ID], "
                    + "[Team1 Points], [Team2 Points], [Team3 Points], "
                    + "[Team4 Points], [Team5 Points]) VALUES (" + values + ");";

            appended = s.executeUpdate(qry);
            System.out.print(appended);
            s.close();
            JOptionPane.showMessageDialog(null, "Opponents have been saved!",
            "SAVED!", JOptionPane.INFORMATION_MESSAGE);
        }//try
        catch (Exception e){
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "There was a problem saving the"
            + "opponents!\nPlease contact the developer.", "OPPONENTS WERE NOT
SAVED!",
            JOptionPane.ERROR_MESSAGE);
        }//end try-catch
    }//addOpponentsToDatabase

    /*
    addTeamResultsToDatabase Mutator Method:
    * Adds the User Team's results to the Leaderboard Table of the Database.
    - Receives a Rank Object.
    - Creates a string with the values that will be transfered to the database.
    - Creates the query to write to the database.
    - Indicates if the transfer was successful or not.
    */
```

```
    public void addTeamResultsToDatabase (Rank tournament){
        String values = "'" + tournament.getTournamentID() + "', "
                            + tournament.getTeamPoints(0) + ", '"
                            + tournament.getWins() + "', '"
                            + tournament.getDraws() + "', '"
                            + tournament.getLosses() + "', '"
                            + tournament.getGoalsScored();

        int appended; //number of rows appended.
        try {
            Statement s = connection.createStatement();
            String qry = "INSERT INTO Leaderboard ([Tournament ID], Points, "
                    + "Wins, Draws, Losses, [Goals Scored]) "
                    + "VALUES (" + values + "');";

            appended = s.executeUpdate(qry);
            System.out.print(appended);
            s.close();
            JOptionPane.showMessageDialog(null, "Leaderboard has been saved!",
            "SAVED!", JOptionPane.INFORMATION_MESSAGE);
        }//try
        catch (Exception e){
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "There was a problem saving the"
            + "tournament!\nPlease contact the developer.", "TOURNAMENT WAS NOT
SAVED!",
            JOptionPane.ERROR_MESSAGE);
        }//end try-catch
    }//addTeamResultsToDatabase

}//Database Class
```

Player Class

```
package ramlakhan;
public class Player {

    //Properties of a Player Object
    private String playerID;
    private String position;
    private String firstName;
    private String surname;
    private String loq;
    private int overall;
    private int offStat;
    private int defStat;
    private String filePath;

    /*
    Player Constructor:
    * Properties required to create a Player Object.
    */
    public Player (String id, String pos, String fN, String sN, String lvl, int
ove, int off, int def, String fP){
        playerID = id;
        position = pos;
        firstName = fN;
        surname = sN;
        loq = lvl;
        overall = ove;
        offStat = off;
        defStat = def;
        filePath = fP;
    }//Player Object

    /*
    getPlayerID Accessor Method:
    * Return the Player ID (String) of a Player.
    */
    public String getPlayerID(){
        return playerID;
    }//getPlayerID

    /*
    getPosition Accessor Method:
    * Return the Position (String) of a Player.
    */
    public String getPosition(){
        return position;
    }//getPosition

    /*
    getFirstName Accessor Method:
    * Return the First Name (String) of a Player.
    */
    public String getFirstName(){
        return firstName;
```

```
    }//getFirstName

    /*
    getSurname Accessor Method:
    * Return the Surname (String) of a Player.
    */
    public String getSurname(){
        return surname;
    }//getSurname

    /*
    getLOQ Accessor Method:
    * Return the Level of Quality (String) of a Player.
    */
    public String getLOQ(){
        return loq;
    }//getLOQ

    /*
    getOverall Accessor Method:
    * Return the Overall (Integer) of a Player.
    */
    public int getOverall(){
        return overall;
    }//getOverall

    /*
    getOffStat Accessor Method:
    * Return the Offensive Stat (Integer) of a Player.
    */
    public int getOffStat(){
        return offStat;
    }//getOffStat

    /*
    getDefStat Accessor Method:
    * Return the Defensive Stat (Integer) of a Player.
    */
    public int getDefStat(){
        return defStat;
    }//getDefStat

    /*
    getFilePath Accessor Method:
    * Return the File Path (String) of a Player.
    */
    public String getFilePath(){
        return filePath;
    }//getFilePath
}//Player Class
```

PlayersArray Class

```
package ramlakhan;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

public class PlayersArray {

    //Properties of a PlayersArray Object
    private Player playerArray [] = new Player [125];

    //Additional objects required:
    private Database db = new Database ();
    private Connection connection = db.getConnection();

    /*
    PlayersArray Constructor:
    * Creates an array of Player Objects by transferring the Players from the
      Players Table of the Database.
    - Declares the temporary properties required to be parsed to create a Player.
    - Uses the ResultSet class to get each field from the Players Database Table.
    - Stores all 125 players and places them in order of their Player ID.
    */
    public PlayersArray (){
        String playerID, position, firstName, surname, loq, filePath;
        int overall, offStat, defStat;
        int count = 0;

        try {
            Statement s = connection.createStatement();
            String qry = "SELECT * FROM Players ORDER BY [Player ID] ASC";
            ResultSet rs = s.executeQuery(qry);
            while (rs.next() && count < 125){
                playerID = rs.getString("Player ID");
                position = rs.getString("Position");
                firstName = rs.getString("First Name");
                surname = rs.getString("Surname");
                loq = rs.getString("LOQ");
                overall = rs.getInt("Overall");
                offStat = rs.getInt("Off Stat");
                defStat = rs.getInt("Def Stat");
                filePath = rs.getString("Image Location");

                playerArray [count] = new Player (playerID, position, firstName,
                       surname, loq, overall, offStat, defStat, filePath);

                count++;
            }//end while loop
            rs.close();
            s.close();
        }//try
        catch (Exception e){
            e.printStackTrace();
            System.out.println ("Error transferring player data!");
```

```
        }//end try-catch

    }//PlayersArray Object

    /*
    findPlayer Accessor Method:
    * Returns the array index (Integer) of a player based off their Player ID.
    - Receives the Player ID of the player needed to be searched.
    */
    public int findPlayer (String searchID){
        int index = -1;

        for (int pos = 0; pos < 125; pos++){
            if (searchID.equals(playerArray [pos].getPlayerID())){
                index = pos;
            }//if then
        }
        return index;
    }//findPlayer

    /*
    getPlayer Accessor Method:
    * Returns a Player object based on the index received.
    - Primarily used in conjunction with the findPlayer method.
    */
    public Player getPlayer (int index){
        return playerArray [index];
    }//getPlayer

    /*
    formatSelectablePlayer Accessor and Mutator Method:
    * Formats and returns specific values of a Player that will be required for
      use on the TeamSelection screen (String).
    - Example: "L.Messi (94).//imageResources//player faces//CAM//5//455.png"
    */
    public String formatSelectablePlayer (int index){
        return playerArray [index].getFirstName().charAt(0) + "."
                + playerArray [index].getSurname() + " ("
                + playerArray [index].getOverall() + ")"
                + playerArray [index].getFilePath();
    }//formatSelectablePlayer

    /*
    pullPlayerDetails Accessor and Mutator Method:
    * Formats and returns specific details of a Player that will be displayed
      as the text of the JComboBoxes. (String)
    * Method is static as it needs to be accessed by the TeamSelection class
      independently of having a PlayersArray object.
    - Receives the formatted player details done by the formatSelectablePlayer
      method.
    - Searches for the position of the final bracket and removes player details
      after the last bracket.
    - Example: "L.Messi (94)"
    */
    public static String pullPlayerDetails (String playerDetails){
```

```
        String formattedPlayer = new String ();
        int bracketPos = playerDetails.lastIndexOf(")");
        formattedPlayer = playerDetails.substring(0, bracketPos + 1);
        return formattedPlayer;
    }//pullPlayerDetails

    /*
    pullPlayerFilePath Accessor and Mutator Method:
    * Formats and returns specific file path of a Player that will be displayed
      as an image in the labels on the TeamSelection screen. (String)
    * Method is static as it needs to be accessed by the TeamSelection class
      independently of having a PlayersArray object.
    - Receives the formatted player details done by the formatSelectablePlayer
      method.
    - Searches for the position of the final bracket and removes player details
      before the last bracket.
    - Example: ".//imageResources//player faces//CAM//5//455.png"
    */
    public static String pullPlayerFilePath (String playerDetails){
        String formattedPlayer = new String ();
        int bracketPos = playerDetails.lastIndexOf(")");
        formattedPlayer = playerDetails.substring(bracketPos + 1);
        return formattedPlayer;
    }//pullPlayerFilePath

    /*
    getPlayerFromFilePath Accessor Method:
    * Returns a Player object based on their File Path. (String)
    * Method is static as it needs to be accessed by the TeamSelection class
      independently of having a PlayersArray object.
    - Receives the File Path of the Player.
    - Searches the PlayersArray and returns the player with matching File Path.
    */
    public static Player getPlayerFromFilePath (String filepath){
        PlayersArray tempPlayerArr = new PlayersArray ();
        Player result = null;

        for (int index = 0; index < 125; index++){
            if (tempPlayerArr.getPlayer(index).getFilePath().equals(filepath)){
                result = tempPlayerArr.getPlayer(index);
            }//if the player has been found
        }//end for loop (search array)
        return result;
    }//getPlayerFromFilePath

}//PlayersArray Class
```

Team Class

```
package ramlakhan;
import java.security.SecureRandom;
import java.sql.Connection;

public class Team {

    //Properties of a Team Object:
    private String tournamentID;
    private String teamName;
    private String gkID;
    private String cbID;
    private String cdmID;
    private String camID;
    private String stID;
    private String username;
    private String tournamentName;
    private int points;
    private int wins;
    private int draws;
    private int losses;
    private int goalsScored;

    //Additional objects required:
    private Database db = new Database ();
    private static PlayersArray playerArr = new PlayersArray ();
    //made static to be referenced in static context contents

    /*
    Team Constructor: (User)
    * Creates a team for a user.
    - Receives the following properties to create the User's team:
        > The user's name
        > The user's team name
        > The user's tournament name
        > The Player ID of the Goalkeeper
        > The Player ID of the Centre Back
        > The Player ID of the Central Defensive Midfielder
        > The Player ID of the Central Attacking Midfielder
        > The Player ID of the Striker
    - Sets the team's points and results to 0.
    */
    public Team (String un, String tN, String toN, String gk, String cb, String
cdm, String cam, String st){
        username = un;
        teamName = tN;
        tournamentName = toN;
        gkID = gk;
        cbID = cb;
        cdmID = cdm;
        camID = cam;
        stID = st;

        points = 0;
```

```
        wins = 0;
        draws = 0;
        goalsScored = 0;
    }//Team Object (User)

    /*
    Team Constructor: (Generated)
    * An overloaded constructor that creates a team for an opponent
    - Receives the Tournament Name set by the user and the Tournament ID
      generated by the program.
    - Calls the generateLOQ method to generate numbers 1 to 5 in a random order
      and temporarily stores them in an array.
    - Calls the getOpponentPlayers method to randomly select a player by parsing
      the position and level of quality number as per the array's index.
    - Sets the team's points and results to 0.
    */
    public Team (String tID, String tN){
        tournamentID = tID;
        teamName = tN;
        int loq [] = generateLOQ ();
        gkID = getOpponentPlayers ("GK", "" + loq [0]);
        cbID = getOpponentPlayers ("CB", "" + loq [1]);
        cdmID = getOpponentPlayers ("CDM", "" + loq [2]);
        camID = getOpponentPlayers ("CAM", "" + loq [3]);
        stID = getOpponentPlayers ("ST", "" + loq [4]);

        points = 0;
        wins = 0;
        draws = 0;
        goalsScored = 0;
    }//Team Object (Opponent)

    /*
    updatePoints Mutator Method:
    * Updates the points acquired by a team based off their results.
    - Resets the number of points before each update.
    */
    public void updatePoints (){
        points = 0;
        points += wins * 1000;
        points += draws * 500;
        points += losses * 200;
        points += goalsScored * 100;
    }//setPoints

    /*
    updateForm Mutator Method:
    * Updates the results of a team.
    - Receives the number of wins, draws, losses and goals scored to append by.
    */
    public void updateForm (int wns, int drws, int los, int gs){
        wins += wns;
        draws += drws;
        losses += los;
        goalsScored +=  gs;
```

```
    }//updateForm

    /*
    getTournamentID Accessor Method:
    * Returns the Tournament ID of a Team. (String)
    */
    public String getTournamentID (){
        return tournamentID;
    }//getTournamentID

    /*
    getUsername Accessor Method:
    * Returns the Username of a User Team. (String)
    */
    public String getUsername (){
        return username;
    }//getUsername

    /*
    getTeamName Accessor Method:
    * Returns the Team Name of a Team. (String)
    */
    public String getTeamName (){
        return teamName;
    }//getTeamName

    /*
    getTournamentName Accessor Method:
    * Returns the Tournament Name of a Team. (String)
    */
    public String getTournamentName (){
        return tournamentName;
    }//getTournamentName

    /*
    getWins Accessor Method:
    * Returns the number of Wins of a Team. (String)
    - Typecasts the integer win into a String.
    */
    public String getWins (){
        return "" + wins;
    }//getWins

    /*
    getDraws Accessor Method:
    * Returns the number of Draws of a Team. (String)
    - Typecasts the integer draw into a String.
    */
    public String getDraws (){
        return "" + draws;
    }//getDraws

    /*
    getLosses Accessor Method:
    * Returns the number of Losses of a Team. (String)
```

```
    - Typecasts the integer losses into a String.
    */
    public String getLosses (){
        return "" + losses;
    }//getLosses

    /*
    getPoints Accessor Method:
    * Returns the number of Points of a Team. (Integer)
    */
    public int getPoints (){
        return points;
    }//getPoints

    /*
    getGoalsScored Accessor Method:
    * Returns the number of Goals Scored of a Team. (String)
    - Typecasts the integer goalsScored into a String.
    */
    public String getGoalsScored (){
        return "" + goalsScored;
    }//getGoalsScored

    /*
    getGK Accessor Method:
    * Returns the Goalkeeper of a Team as a Player object.
    - Creates a temporary Player object.
    - Initializes the Player object by calling the getPlayer method followed by
      the findPlayer method from the PlayersArray Class.
    - The findPlayer finds the player using the GK's Player ID (gkID).
    */
    public Player getGK (){
        Player gk;
        gk = playerArr.getPlayer(playerArr.findPlayer(gkID));
        return gk;
    }//getGK

    /*
    getCB Accessor Method:
    * Returns the Centre Back of a Team as a Player object.
    - Creates a temporary Player object.
    - Initializes the Player object by calling the getPlayer method followed by
      the findPlayer method from the PlayersArray Class.
    - The findPlayer finds the player using the CB's Player ID (cbID).
    */
    public Player getCB (){
        Player cb;
        cb = playerArr.getPlayer(playerArr.findPlayer(cbID));
        return cb;
    }//getCB

    /*
    getCDM Accessor Method:
    * Returns the Central Defensive Midfielder of a Team as a Player object.
    - Creates a temporary Player object.
```

```
- Initializes the Player object by calling the getPlayer method followed by
  the findPlayer method from the PlayersArray Class.
- The findPlayer finds the player using the CDM's Player ID (cdmID).
*/
public Player getCDM (){
    Player cdm;
    cdm = playerArr.getPlayer(playerArr.findPlayer(cdmID));
    return cdm;
}//getCDM

/*
getCAM Accessor Method:
* Returns the Central Attacking Midfielder of a Team as a Player object.
- Creates a temporary Player object.
- Initializes the Player object by calling the getPlayer method followed by
  the findPlayer method from the PlayersArray Class.
- The findPlayer finds the player using the CAM's Player ID (camID).
*/
public Player getCAM (){
    Player cam;
    cam = playerArr.getPlayer(playerArr.findPlayer(camID));
    return cam;
}//getCAM

/*
getST Accessor Method:
* Returns the Striker of a Team as a Player object.
- Creates a temporary Player object.
- Initializes the Player object by calling the getPlayer method followed by
  the findPlayer method from the PlayersArray Class.
- The findPlayer finds the player using the ST's Player ID (stID).
*/
public Player getST (){
    Player st;
    st = playerArr.getPlayer(playerArr.findPlayer(stID));
    return st;
}//getST

/*
getTeamPlayers Accessor and Mutator Method:
* Formats and returns a String array listing the Team's Players.
- A String array is created to store 5 Strings where each index stores a
  player of a specific position.
- Calls the get*Position* method of the Team's class and then calls the
  getFirstName and getSurname methods of the Player's class to generate a
  player's name neatly.
- Example: "[CAM] - Lionel Messi"
*/
public String [] getTeamPlayers (){
    String teamPlayers [] = new String [5];

    teamPlayers [0] = "[GK] - " + this.getGK().getFirstName() + " "
                                + this.getGK().getSurname();

    teamPlayers [1] = "[CB] - " + this.getCB().getFirstName() + " "
```

```
                                    + this.getCB().getSurname();

        teamPlayers [2] = "[CDM] - " + this.getCDM().getFirstName() + " "
                                     + this.getCDM().getSurname();

        teamPlayers [3] = "[CAM] - " + this.getCAM().getFirstName() + " "
                                     + this.getCAM().getSurname();

        teamPlayers [4] = "[ST] - " + this.getST().getFirstName() + " "
                                    + this.getST().getSurname();

        return teamPlayers;
    }//getTeamPlayers

    /*
    generateLOQ Accessor and Mutator Method:
    * Generates numbers 1 to 5 and places them in a random order into an int
array.
    * Method is static as it needs to be accessed by other classes independently
      of a Team object.
    - Code has been externally sourced and adapted to fit the program
requirements.

https://www.youtube.com/watch?v=Q8OWCDlFoVo&list=PL79nsbeypYKEFBUfWTczAixnTFOVqPin
6&index=3
    */
    public static int [] generateLOQ (){
        int randomNumbers [] = new int [5];
        int range [] = new int [5];
        int indexBounds = 5;
        int randomNum;

        SecureRandom random = new SecureRandom();

        for (int index = 0; index < 5; index++){
            range [index] = index + 1;
        }//end for loop (indexing and setting range)

        for (int index = 0; index < 5; index++ ){
            randomNum = random.nextInt(indexBounds);
            randomNumbers [index] = range [randomNum];
            range [randomNum] = range [indexBounds - 1];
            indexBounds--;
        }//end for loop (placing random numbers into array)

        return randomNumbers;
    }//generateLOQ

    /*
    getPossiblePlayers Accessor and Mutator Method:
    * Returns a formatted list of players that are selectable based on their
      position and level of quality.
    * Method is static as it needs to be accessed by other classes independently
      of a Team object.
    - Receives the position, level of quality number and boolean to indicate the
```

```
      type of formatting required.

    Algorithm Process:
    - A String array to store the list of players, as well as a temporary Player
      object and an index and count (both initialised to 0) is created.
    - A while loop tests that the index is less than 125 (max number of players)
      or that the count is not greater than 5 (max number of possible selections).
    - The tempPlayer object is overwritten at the beginning of the loop and is
      assigned the Player who has that index from the PlayersArray by calling
      the getPlayer method from the PlayersArray Class and parsing the index.
    - An if statement is used to check if the tempPlayer has the same position
      and level of quality number that was received by the method.
    - Based on the boolean:
        > if true: possiblePlayers array at count is initialized to tempPlayer
        which is formatted using the formatSelectablePlayer method of
PlayersArray.
        > if false: possiblePlayers array at count is initialized to the
tempPlayer
         Players ID by calling the getPlayer and getPlayerID methods of the
         PlayersArray and Player class respectively.
    - The count is incremeneted if the tempPlayer matched the parameters.
    - The index is incremented before the end of the while loop.
    */
    public static String [] getPossiblePlayers (String position, String loq,
boolean format){
        String possiblePlayers [] = new String [5];
        Player tempPlayer;
        int index = 0;
        int count = 0;

        while (index < 125 || count < 5){
            tempPlayer = playerArr.getPlayer (index);
            if (tempPlayer.getPosition().equals(position) &&
tempPlayer.getLOQ().equals(loq)){
                if (format == true){
                    possiblePlayers [count] =
playerArr.formatSelectablePlayer(index);
                }//nested if
                else if (format == false) {
                    possiblePlayers [count] =
playerArr.getPlayer(index).getPlayerID();
                }//else if
                count++;
            }//if then
            index++;
        }//end while loop

        return possiblePlayers;
    }//getPossiblePlayers

    /*
    getOpponentPlayers Accessor and Mutator Method:
    * Randomly selects and returns the Player ID of a randomly chosen player
      based on the position and level of quality. (String)
    - Receives the required position and level of quality.
```

```
    - Runs the getPossiblePlayers method with parsed parameters and a false
      boolean to only get the Players ID's.
    - Generates a random number between 1 to 5 using generateLOQ and takes
      the first index value of the returned array.
    - A String is assigned the value of the players array based on the randomly
      generated index number. (this number is subtracted by 1 to be withing range)
    */
    public static String getOpponentPlayers (String position, String loq){
        String players [] = Team.getPossiblePlayers(position, loq, false);
        int randomIndex = Team.generateLOQ() [0];
        String playerID = players [randomIndex - 1];
        return playerID;
    }//getOpponentPlayers

    /*
    addTeamToDatabase Mutator Method:
    * Adds a User team to the Teams table in the Database and calls the
      getTop1StringRecord Accessor method from the Database class to get the
      generated AutoNumber (the primary key) from the Teams table.
    */
    public void addTeamToDatabase (){
        db.runTeamInsertQuery(this);
        tournamentID = db.getTop1StringRecord("Tournament ID", "Teams",
                "Username = '" + username + "' AND [Team Name] = '"
                        + teamName + "' AND [Tournament Name] = '"
                        + tournamentName + "' ", "DESC");
    }//addTeamToDatabase

}//Team Class
```

League Class

```
package ramlakhan;
import java.util.Random;
import javax.swing.JOptionPane;

public class League {

    //Properties of a League object:
    private Team [] teamsLeagueArr = new Team [6];
    private String rankingsTable [] = new String [6];

    /*
    League Constructor:
    * Creates an instance of the League class.
    * Essentially an array of 6 teams.
    - Uses an array of 6 teams to store each team.
    - Receives a user team and 5 generated teams to be put into the array.
    - Creates a Ranking Table in the correct format.
    */
    public League (Team uT, Team t1, Team t2, Team t3, Team t4, Team t5){
        teamsLeagueArr [0] = uT;
        teamsLeagueArr [1] = t1;
        teamsLeagueArr [2] = t2;
        teamsLeagueArr [3] = t3;
        teamsLeagueArr [4] = t4;
        teamsLeagueArr [5] = t5;

        for (int index = 0; index < 6; index++){
            rankingsTable [index] = "[" + (index + 1) + "] " + teamsLeagueArr
            [index].getTeamName() + " - " + teamsLeagueArr [index].getPoints()
            + " Points";
        }//end for loop

    }//League Constructor

    //Variables required for sorting and updating the Rankings Table:
    private int sortedPoints [] = new int [6];
    private String rankTeamNames [] = new String [6];

    /*
    updateTable Mutator Method:
    * Updates the Rankings Table of the League.
    - Gathers each teams name and points and stores them in two separate arrays
      of type String and int respectively.
    - Uses a selection sort to sort the array descending Team Points order.
    - Updates the Ranking Table in the correct format.
    - Example: [1] Leonidas - 10000 Points
    */
    public void updateTable (){
        for (int index = 0; index < 6; index++){
            sortedPoints [index] = teamsLeagueArr [index].getPoints();
            rankTeamNames [index] = teamsLeagueArr [index].getTeamName();
        }//end for loop
```

```
        for (int x = 0; x < 5; x++){
            for (int y = x + 1; y < 6; y++){

                if (sortedPoints [y] > sortedPoints [x]){
                    int pointTemp = sortedPoints [x];
                    sortedPoints [x] = sortedPoints [y];
                    sortedPoints [y] = pointTemp;

                    String nameTemp = rankTeamNames [x];
                    rankTeamNames [x] = rankTeamNames [y];
                    rankTeamNames [y] = nameTemp;
                }//if then
            }//end for loop (sort inner)
        }//end for loop (sort outer)

        for (int rank = 0; rank < 6; rank++){
            rankingsTable [rank] = "[" + (rank + 1) + "] " + rankTeamNames
                        [rank] + " - " + sortedPoints [rank] + " Points";
        }//end for loop
    }//updateTable

    /*
    getRankTable Accessor Method:
    * Returns the Rankings Table. (String)
    */
    public String [] getRankTable (){
        return rankingsTable;
    }//getRankTable accessor

    /*
    getTeam Accessor Method:
    * Returns a specfic Team object in the league.
    - Receives the index of the team in the main teamsLeagueArr array.
    */
    public Team getTeam (int index){
        Team find = teamsLeagueArr [index];
        return find;
    }

    /*
    enums have been used to make more logical sense in the program and remove
    the need of having to use chars and assign letters with them for specific
    tactics.
    */

    /*
    Mentality enum:
    * Possible selections when selecting a Mentality tactic.
    */
    enum Mentality {
        ATTACKING,
        DEFENSIVE,
        BALANCED
    }//Mentality enum
```

```
    /*
    InPossession enum:
    Possible selections when selecting an In Possession tactic.
    */
    enum InPossession {
        TikiTaka,
        ShortPassing,
        FastBuildUp
    }//InPossession enum

    /*
    OutOfPossession:
    * Possible selections when selecting an Out of Possession tactic.
    */
    enum OutOfPossession {
        TeamPress,
        CounterAttack,
        ParkTheBus
    }//GameOutcomes enum

    //Variables required for simulating matchweeks:
    private League.Mentality userMentality;
    private League.Mentality aiMentality;
    private League.InPossession userInPossession;
    private League.InPossession aiInPossession;
    private League.OutOfPossession userOutOfPossession;
    private League.OutOfPossession aiOutOfPossession;
    private String userResult;
    private String game2Result;
    private String game3Result;

    /*
    playMatchWeek Accessor and Mutator Method:
    * Runs all the methods required to simulate a specific matchweek.
    - Receives the matchweek number and the user's selected tactics (Mentality,
      InPossession and OutOfPossession),.
    - Overides the current local tactic enum variables.
    - Generates a random value for each enum tactic.
        > These enums will be used for the simulation of the other teams
          matches as well.
    - Uses a switch statement which determines which matchweek to simulate.
        > Overides the userTeamResult String.
        > Does not require a default as matchweek cannot be any other case as it
          is program defined.
    - Runs the method updateTable to update the Rankings Table.
    - Displays a Message Dialog Box with the results of the 3 games simulated.
    - Returns the user teams result which is displayed on the Tournament Screen.
    */
    public String playMatchWeek (int matchweek, League.Mentality selMentality,
    League.InPossession selInPossession, League.OutOfPossession
selOutOfPossession){
        String userTeamResult = "";
        userMentality = selMentality;
        userInPossession = selInPossession;
        userOutOfPossession = selOutOfPossession;
```

```
        aiMentality = League.Mentality.values()
                [new Random().nextInt(League.Mentality.values().length)];
        aiInPossession = League.InPossession.values()
                [new Random().nextInt(League.InPossession.values().length)];
        aiOutOfPossession = League.OutOfPossession.values()
                [new Random().nextInt(League.OutOfPossession.values().length)];

        switch (matchweek){
            case 1:
                userTeamResult = matchWeek1 ();
            break;

            case 2:
                userTeamResult = matchWeek2 ();
            break;

            case 3:
                userTeamResult = matchWeek3 ();
            break;

            case 4:
                userTeamResult = matchWeek4 ();
            break;

            case 5:
                userTeamResult = matchWeek5 ();
            break;
        }//end switch
        updateTable ();

        JOptionPane.showMessageDialog(null, "Matchday " + matchweek + " Results: "
                    + "\n" + userResult + "\n"+ game2Result + "\n" + game3Result,
                            "Matchday Results", JOptionPane.INFORMATION_MESSAGE);

        return userTeamResult;
    }//playMatchWeek

    /*
    matchWeek1 Accessor Method:
    * Returns the result of the user team. (String)
    * Simulates all the games for Matchweek 1.
        > User Team vs AI Team 1
        > AI Team 2 vs AI Team 3
        > AI Team 4 vs AI Team 5
    * Calls the method playMatch and parses the two teams and a boolean that
      indicates if it is a user team or not.
    - The first team parsed is the team on the left when a result is displayed.
    - Returns the result of the user's team.
    */
    public String matchWeek1 (){
        userResult = playMatch (teamsLeagueArr [0], teamsLeagueArr [1], true);
        game2Result = playMatch (teamsLeagueArr [2], teamsLeagueArr [3], false);
        game3Result = playMatch (teamsLeagueArr [4], teamsLeagueArr [5], false);
```

```
    return userResult;
}//matchWeek1

/*
matchWeek2 Accessor Method:
* Returns the result of the user team. (String)
* Simulates all the games for Matchweek 2.
    > User Team vs AI Team 2
    > AI Team 1 vs AI Team 4
    > AI Team 3 vs AI Team 5
* Calls the method playMatch and parses the two teams and a boolean that
  indicates if it is a user team or not.
- The first team parsed is the team on the left when a result is displayed.
- Returns the result of the user's team.
*/
public String matchWeek2 (){
    userResult = playMatch (teamsLeagueArr [0], teamsLeagueArr [2], true);
    game2Result = playMatch (teamsLeagueArr [1], teamsLeagueArr [4], false);
    game3Result = playMatch (teamsLeagueArr [3], teamsLeagueArr [5], false);

    return userResult;
}//matchWeek2

/*
matchWeek3 Accessor Method:
* Returns the result of the user team. (String)
* Simulates all the games for Matchweek 3.
    > User Team vs AI Team 3
    > AI Team 1 vs AI Team 5
    > AI Team 2 vs AI Team 4
* Calls the method playMatch and parses the two teams and a boolean that
  indicates if it is a user team or not.
- The first team parsed is the team on the left when a result is displayed.
- Returns the result of the user's team.
*/
public String matchWeek3 (){
    userResult = playMatch (teamsLeagueArr [0], teamsLeagueArr [3], true);
    game2Result = playMatch (teamsLeagueArr [1], teamsLeagueArr [5], false);
    game3Result = playMatch (teamsLeagueArr [2], teamsLeagueArr [4], false);

    return userResult;
}//matchWeek3

/*
matchWeek4 Accessor Method:
* Returns the result of the user team. (String)
* Simulates all the games for Matchweek 4.
    > User Team vs AI Team 4
    > AI Team 1 vs AI Team 3
    > AI Team 2 vs AI Team 5
* Calls the method playMatch and parses the two teams and a boolean that
  indicates if it is a user team or not.
- The first team parsed is the team on the left when a result is displayed.
- Returns the result of the user's team.
*/
```

```
public String matchWeek4 (){
    userResult = playMatch (teamsLeagueArr [0], teamsLeagueArr [4], true);
    game2Result = playMatch (teamsLeagueArr [1], teamsLeagueArr [3], false);
    game3Result = playMatch (teamsLeagueArr [2], teamsLeagueArr [5], false);

    return userResult;
}//matchWeek4

/*
matchWeek5 Accessor Method:
* Returns the result of the user team. (String)
* Simulates all the games for Matchweek 5.
    > User Team vs AI Team 5
    > AI Team 1 vs AI Team 2
    > AI Team 3 vs AI Team 4
* Calls the method playMatch and parses the two teams and a boolean that
  indicates if it is a user team or not.
- The first team parsed is the team on the left when a result is displayed.
- Returns the result of the user's team.
*/
public String matchWeek5 (){
    userResult = playMatch (teamsLeagueArr [0], teamsLeagueArr [5], true);
    game2Result = playMatch (teamsLeagueArr [1], teamsLeagueArr [2], false);
    game3Result = playMatch (teamsLeagueArr [3], teamsLeagueArr [4], false);

    return userResult;
}//matchWeek5

//Variables required for simulating individual matches:
private int t1Goals;
private int t2Goals;
private double t1RunGoals;
private double t2RunGoals;
private double t1OffBoost;
private double t2DefBoost;

/*
The following integer arrays store the values based on the tactics selected:
[0] - Mentality Offensive Boost
[1] - Mentality Defensive Boost
[2] - In Possession Offensive Boost
[3] - In Possession Defensive Boost
[4] - Out of Possession Offensive Boost
[5] - Out of Possession Defensive Boost
*/
private int team1Tactics [] = new int [6];
private int team2Tactics [] = new int [6];

/*
playMatch Accessor and Mutator Method:
* Returns the result of the game. (String)
* Runs all the processes involved in simulating a match between two teams.
- Receives the two Teams that will play each other and a boolean indicating
  if it involves a user team.
- Variables that keep track of goals are reset to 0.
```

```
        > There is an integer and double object that store goals.
        > This is because the goals are not rounded off until all player vs player
          methods have run.
    - The result and Game Outcome enum are reset.
    - Calls the determineBoosts accessor method to overide the team1Tactics and
      team2Tactics boost values.
        > Parses the selected mentality, inPossession, outOfPossession and array
          of the specific team.
        > The tactics that a user team selects will be the same for other teams
          that are parsed as t1.
    - Runs the teamVSteam mutator method twice, changing the order of the teams
      to make both play offensive and defensive.
        > This is explain in the teamVSteam method.
    - Rounds off the goals and sets those values to the integers storing the
goals.
    - Determines which team won the game using if statements.
        > Calls the updateMethod mutator method of the Team class to update
          each teams goals, wins, losses and draws.
        > Sets the GameOutcomes enum value based on the result.
    - Checks the received boolean user to determine if a message box needs to
      be displayed to indicate the match outcome to the user.
    - Calls the updatePoints mutator method of the Teams class to determine
      the points of the team based on their form.
    - Formats the result string to look aesthetic on the results board in the
      Tournament Screen.
    - Example: Leonidas | 17 - 13 | AI Team 2
    - Returns the formatted string result.
    */
    public String playMatch (Team t1, Team t2, boolean user){
        t1Goals = 0;
        t2Goals = 0;
        t1RunGoals = 0;
        t2RunGoals = 0;
        String result = "";
        League.GameOutcomes outcome = null;

        determineBoosts (userMentality, userInPossession, userOutOfPossession,
                         team1Tactics);
        determineBoosts (aiMentality, aiInPossession, aiOutOfPossession,
                         team2Tactics);

        teamVSteam (t1, t2, team1Tactics, team2Tactics, 1, 2);
        teamVSteam (t2, t1, team2Tactics, team1Tactics, 2, 1);

        t1Goals = (int) Math.round(t1RunGoals);
        t2Goals = (int) Math.round(t2RunGoals);

        if (t1Goals > t2Goals){
            t1.updateForm(1, 0, 0, t1Goals);
            t2.updateForm(0, 0, 1, t2Goals);
            outcome = League.GameOutcomes.Win;
        }//if then
        else if (t1Goals == t2Goals){
            t1.updateForm(0, 1, 0, t1Goals);
            t2.updateForm(0, 1, 0, t2Goals);
```

```
            outcome = League.GameOutcomes.Draw;
        }//else if
        else if (t1Goals < t2Goals){
            t1.updateForm(0, 0, 1, t1Goals);
            t2.updateForm(1, 0, 0, t2Goals);
            outcome = League.GameOutcomes.Loss;
        }//else if

        if (user == true){
            displayMatchOutcome (outcome);
        }//if then

        t1.updatePoints();
        t2.updatePoints();

        result = t1.getTeamName() + " | " + t1Goals + " - " + t2Goals + " | "
                + t2.getTeamName();
        return result;
    }//playMacth

    /*
    teamVSteam Mutator Method:
    * Simulates a match between two teams.
    * It is critical that parsing is done correctly to determine which team is
      offensive and which team is defensive!
    - Receives the two team objects as well as their integer array of tactic
      boosts and a number that indicates which team is playing offensive and
      which team is playing defensive.
        offTeam - The offensive team
        defTeam - The defensve team
        team1Tac - the tactics array of the offensive team
        team2Tac - the tactics array of the defensive team
        team number of the offensive team
        team number of the defensive team
    - team1Player will be constantly overwritten and is just the offensive team
player's
      offensive stat multiplied by the boost.
    - team2Player will be constantly overwritten and is just the defensive team
player's
      defensive stat multiplied by the boost.
    - Sets the offensive team and defensive team MENTALITY boosts.
        > the offensive team gets their Offensive Mentality Boost
        > the defensive team gets their Defensive Mentality Boost
    - [Algorithm]

    Algorithm Process:
    * All are simulated using the playerVSplayer method by parsing the offensive
      boosted stat and defensive boosted stat of each player respectfully.
    1) Offensive Team's Goalkeeper plays against Defensive Teams's Striker
    2) Offensive Team's Centre Back plays against Defensive Teams's Attacking Mid
    3) Offensive Team's Defensive Mid plays against Defensive Teams's Defensive
Mid
    4) Offensive Team's Attacking Mid plays against Defensive Teams's Centre Back
    5) Offensive Team's Striker plays against Defensive Teams's Goalkeeper
    */
```

```
    public void teamVSteam (Team offTeam, Team defTeam, int[] offTeamTactics,
        int defTeamTactics [], int offTeamNumber, int defTeamNumber){
        int team1Player;
        int team2Player;
        t1OffBoost = (double) offTeamTactics [0] / 100;
        t2DefBoost = (double) defTeamTactics [1] / 100;

        team1Player = (int) Math.round(offTeam.getGK().getOffStat() * t1OffBoost);
        team2Player = (int) Math.round(defTeam.getST().getDefStat() * t2DefBoost);
        playerVSplayer (team1Player, team2Player, offTeamTactics, defTeamTactics,
                        offTeamNumber, defTeamNumber);

        team1Player = (int) Math.round(offTeam.getCB().getOffStat() * t1OffBoost);
        team2Player = (int) Math.round(defTeam.getCAM().getDefStat() *
t2DefBoost);
        playerVSplayer (team1Player, team2Player, offTeamTactics, defTeamTactics,
                        offTeamNumber, defTeamNumber);

        team1Player = (int) Math.round(offTeam.getCDM().getOffStat() *
t1OffBoost);
        team2Player = (int) Math.round(defTeam.getCDM().getDefStat() *
t2DefBoost);
        playerVSplayer (team1Player, team2Player, offTeamTactics, defTeamTactics,
                        offTeamNumber, defTeamNumber);

        team1Player = (int) Math.round(offTeam.getCAM().getOffStat() *
t1OffBoost);
        team2Player = (int) Math.round(defTeam.getCB().getDefStat() * t2DefBoost);
        playerVSplayer (team1Player, team2Player, offTeamTactics, defTeamTactics,
                        offTeamNumber, defTeamNumber);

        team1Player = (int) Math.round(offTeam.getST().getOffStat() * t1OffBoost);
        team2Player = (int) Math.round(defTeam.getGK().getDefStat() * t2DefBoost);
        playerVSplayer (team1Player, team2Player, offTeamTactics, defTeamTactics,
                        offTeamNumber, defTeamNumber);
    }//teamVSteam

    /*
    playerVSplayer Mutator Method:
    * Compares two players to determine the goals scored.
    - Recives an offensive and defensive player stat as well as the tactic arrays
      for each team and a number to represent if they are attacking or defensive.
      This is a key to ensure that the correct team's points are added because
      the running goal total variables are local.
        > 1 - Offensive
        > 2 - Defensive

    Algorithm Process:
    - If the offensive player's stat is greater than the defensive player's stat:
        > Update the offensive team's boost by adding the In Possession offensive
          boost [2] with the Mentality offensive boost [0]
        > Update the defensive team's boost by adding the Out of Possession
          defensive boost [5] with the Mentality defensive boost [1]
        > Work out the difference between the stats.
        > Update the goals for the correct team.
```

```
    - If the defensive player's stat is greater than the offensive player's stat:
        > Update the offensive team's boost by adding the Out of Possession
offensive
          boost [4] with the Mentality offensive boost [0]
        > Update the defensive team's boost by adding the In Possession
          defensive boost [3] with the Mentality defensive boost [1]
        > Work out the difference between the stats.
        > Update the goals for the correct team.
    */
    public void playerVSplayer (int p1Off, int p2Def, int t1Tac [], int t2Tac [],
                                int offTeam, int defTeam){
        int difference;
        if (p1Off > p2Def){
            t1OffBoost = (double) (t1Tac [0] + t1Tac [2]) / 100;
            t2DefBoost = (double) (t2Tac [1] + t2Tac [5]) / 100;
            difference = p1Off - p2Def;

            if (offTeam == 1){
                t1RunGoals += (double) difference / 10;
            }//if then
            else if (offTeam == 2){
                t2RunGoals += (double) difference / 10;
            }//else if
        }//if then
        else if (p2Def > p1Off){
            t1OffBoost = (double) (t1Tac [0] + t1Tac [4]) / 100;
            t2DefBoost = (double) (t2Tac [1] + t2Tac [3]) / 100;
            difference = p2Def - p1Off;

            if (defTeam == 1){
                t1RunGoals += (double) difference / 10;
            }//if then
            else if (defTeam == 2){
                t2RunGoals += (double) difference / 10;
            }//else if
        }//else if
    }//playerVSplayer

    /*
    displayMatchOutcome Method:
    * Displays a message box to the user with the game outcome.
    - Receives the GameOutcome and uses a switch to determine what must be put
      into the message box.
    - No default required as there are no other possible cases.
    */
    public void displayMatchOutcome (League.GameOutcomes outcome){
        String captainComment = "";
        String result = "";

        switch (outcome){
            case Win:
                result = "WON!";
                captainComment = "Team Captain: \nTactics worked well gaffer! "
                               + "A good victory!";
```

```
            break;

            case Draw:
                result = "DREW!";
                captainComment = "Team Captain: \nNo differences between the two"
                               + " teams but we'll take the draw and move on!";
            break;

            case Loss:
                result = "LOST!";
                captainComment = "Team Captain: \nTeam didn't give it their all!"
                               + " Sorry to let you down boss.";
            break;
        }//end switch

        JOptionPane.showMessageDialog(null, captainComment, "YOUR TEAM " +
                                    result, JOptionPane.INFORMATION_MESSAGE);
    }//displayMatchOutcomes

    /*
    determineBoosts Mutator Method:
    * Sets the value of the different boosts of a team.
    - Receives the mentality, in possession, out of possession and team tactic
      array that will store the values.
    - Uses a switch to determine which enum values were chosen and updates the
      tactic accordingly using the correct index.
        [0] - Mentality Offensive Boost
        [1] - Mentality Defensive Boost
        [2] - In Possession Offensive Boost
        [3] - In Possession Defensive Boost
        [4] - Out of Possession Offensive Boost
        [5] - Out of Possession Defensive Boost
    */
    public void determineBoosts (League.Mentality mentality, League.InPossession
          inPossession, League.OutOfPossession outOfPossession, int tactics []){

        switch (mentality){
            case ATTACKING:
                tactics [0] = 125;
                tactics [1] = 105;
            break;

            case DEFENSIVE:
                tactics [0] = 105;
                tactics [1] = 125;
            break;

            case BALANCED:
                tactics [0] = 115;
                tactics [1] = 115;
            break;
        }//end switch

        switch (inPossession){
            case TikiTaka:
```

```
                tactics [2] = 15;
                tactics [3] = -5;
            break;

            case ShortPassing:
                tactics [2] = 10;
                tactics [3] = 5;
            break;

            case FastBuildUp:
                tactics [2] = 20;
                tactics [3] = -10;
            break;
        }//end switch

        switch (outOfPossession){
            case TeamPress:
                tactics [4] = -5;
                tactics [5] = 20;
            break;

            case CounterAttack:
                tactics [4] = 10;
                tactics [5] = 10;
            break;

            case ParkTheBus:
                tactics [4] = 5;
                tactics [5] = 15;
            break;
        }//end switch

    }//determineBoosts
}//League Class
```

Rank Class

```
package ramlakhan;
import java.sql.Connection;

public class Rank{

    //Properties of a Rank object:
    private String userPlayers [] = new String [5];
    private String userName;
    private String userTeamName;
    private String tournamentName;
    private String tournamentID;
    private int points;
    private String wins;
    private String draws;
    private String losses;
    private String goalsScored;
    private String leagueTableRankings [] = new String [6];
    private int teamPoints [] = new int [6];
    private Database db = new Database ();

    /*
    Rank Constructor: (post Tournament)
    * Creates a Rank after a Tournament has occured.
    * Essentially a modified League object.
    - Receives a League object and use League class accessor methods to set
      the value of all the properties of a Rank object equal to the
      relevant matching League property.
    - Stores all 6 team's points in an array.
        > Order: User, AI1, AI2, AI3, AI4, AI5
        > Used to write to the database.
    */
    public Rank (League rankedLeague){
        userPlayers = rankedLeague.getTeam(0).getTeamPlayers();
        userTeamName = rankedLeague.getTeam(0).getTeamName();
        tournamentName = rankedLeague.getTeam(0).getTournamentName();
        tournamentID = rankedLeague.getTeam(0).getTournamentID();
        wins = rankedLeague.getTeam(0).getWins();
        draws = rankedLeague.getTeam(0).getDraws();
        losses = rankedLeague.getTeam(0).getLosses();
        goalsScored = rankedLeague.getTeam(0).getGoalsScored();

        leagueTableRankings = rankedLeague.getRankTable();

        for (int index = 0; index < 6; index++){
            teamPoints [index] = rankedLeague.getTeam(index).getPoints();
        }//end for loop
    }//Rank Object (In-Game)

    /*
    Rank Constructor: (creation from database)
    * Used to create a Rank from the database.
    - Receives the following properties:
        > Username
```

```
        > User Team's Name
        > Tournament Name
        > User's Goalkeeper Player ID
        > User's Centre Back Player ID
        > User's Central Defensive Midfielder Player ID
        > User's Central Attacking Midifelder Player ID
        > User's Striker Player ID
        > User's Amount of Wins
        > User's Amount of Draws
        > User's Amount of Losses
        > User's Amount of Goals Scored
        > AI Team 1's Points
        > AI Team 2's Points
        > AI Team 3's Points
        > AI Team 4's Points
        > AI Team 5's Points
    - Creates a temporary Team to use the method getTeamPlayers to get a
      formatted list of the user team's players based on their player ID.
    - Example: [CAM] Lionel Messi
    - Sets all the Rank properties to the corresponding parsed value.
    - Calls the method generateLeagueTable and parses the points acquired by the
      5 generated teams.
    */
    public Rank (String uN, String tN, String toN, String gkID, String cbID,
            String cdmID, String camID, String stID, int pts, String wns,
            String drws, String ls, String gS, int t1Pts, int t2Pts, int t3Pts,
            int t4Pts, int t5Pts ){
        Team tempUserTeam = new Team (uN, tN, toN, gkID, cbID, cdmID, camID,
stID);

        userPlayers = tempUserTeam.getTeamPlayers();
        userName = uN;
        userTeamName = tN;
        tournamentName = toN;
        points = pts;
        wins = wns;
        draws = drws;
        losses = ls;
        goalsScored = gS;
        leagueTableRankings = generateLeagueTable (t1Pts, t2Pts, t3Pts, t4Pts,
t5Pts);

    }//Rank Object (From Database)

    /*
    generateLeagueTable Accessor and Mutator Method:
    * Creates and returns a rankings table of a Rank. (String)
    - Receives the 5 generated team points in the correct name order.
    - Uses a for loop to store the team names in a string array.
    - Sets the values of the points in a parallel integer array to the string
      array that stores the team names.
    - Uses a selection sort to sort the arrays in descending order of points
      acquired.
    - Uses a for loop to neatly format the table in the new order.
    - Example: [2] Leonidas - 10000 Points
```

```
    */
    public String [] generateLeagueTable (int t1Pts, int t2Pts, int t3Pts,
    int t4Pts, int t5Pts){
        String formattedRankings [] = new String [6];
        String teamNames [] = new String [6];

        //User team properties:
        teamNames [0] = userTeamName;
        teamPoints [0] = points;

        for (int index = 1; index < 6; index++){
            teamNames [index] = "AI Team " + index;
        }//end for loop
        teamPoints [1] = t1Pts;
        teamPoints [2] = t2Pts;
        teamPoints [3] = t3Pts;
        teamPoints [4] = t4Pts;
        teamPoints [5] = t5Pts;

        for (int x = 0; x < 5; x++){
            for (int y = x + 1; y < 6; y++){
                if (teamPoints [y] > teamPoints [x]){
                    int tempPoints = teamPoints [x];
                    teamPoints [x] = teamPoints [y];
                    teamPoints [y] = tempPoints;

                    String tempName = teamNames [x];
                    teamNames [x] = teamNames [y];
                    teamNames [y] = tempName;
                }//if then (sort arrays)
            }//end for loop (sort y)
        }//end for loop (sort x)

        for (int position = 0; position < 6; position++){
            formattedRankings [position] = "[" + (position + 1) + "] "
            + teamNames [position] + " - " + teamPoints [position] + " Points";
        }//end for loop

        return formattedRankings;
    }//generateLeagueTable

    /*
    getUserPlayer Accessor Method:
    * Returns a specific player of the user team. (String)
    - Receives the index of the player in the array.
        [0] - Goalkeeper
        [1] - Centre Back
        [2] - Central Defensive Midfielder
        [3] - Central Attacking Midfielder
        [4] - Striker
    */
    public String getUserPlayer (int index){
        return userPlayers [index];
    }//getUserPlayer
```

```
    /*
    getUserTeamName Accessor Method:
    * Returns the user's team name. (String)
    */
    public String getUserTeamName (){
        return userTeamName;
    }//getUserTeamName

    /*
    getTournamentName Accessor Method:
    * Returns the user's tournament name. (String)
    */
    public String getTournamentName() {
        return tournamentName;
    }//getTournamentName

    /*
    getTournamentID Accessor Method:
    * Returns the Tournament ID that will be used as the primary key in the
      database. (String)
    - This number has to be the same as the Teams, Opponent and Leaderboard
      tables are all joined using the same primary key number.
    */
    public String getTournamentID() {
        return tournamentID;
    }//getTournamentID

    /*
    getWins Accessor Method:
    * Returns the user's amount of wins. (String)
    */
    public String getWins() {
        return wins;
    }//getWins

    /*
    getDraws Accessor Method:
    * Returns the user's amount of draws. (String)
    */
    public String getDraws() {
        return draws;
    }//getDraws

    /*
    getLosses Accessor Method:
    * Returns the user's amount of losses. (String)
    */
    public String getLosses() {
        return losses;
    }//getLosses

    /*
    getGoalsScored Accessor Method:
    * Returns the user team's goals scored. (String)
    */
```

```
    public String getGoalsScored() {
        return goalsScored;
    }//getGoalsScored

    /*
    getLeagueTableRankings Accessor Method:
    * Returns the formatted ranking of a team based on their position in the
      table. (String)
    - Used to update JLabels when displaying the rankings table.
    */
    public String getLeagueTableRankings(int position) {
        return leagueTableRankings [position];
    }//getLeagueTableRankings

    /*
    //getTeamPoints Accessor Method:
    * Returns the points of a specific team. (Integer)
    - Receives the index of the team required.
    */
    public int getTeamPoints(int team) {
        return teamPoints [team];
    }//getTeamPoints

    /*
    getRankTitle Accessor Method:
    * Returns the rank title (String) that will be displayed on a leaderboard
button.
    */
    public String getRankTitle (){
        String rank = " " + userName + " - [" + points + " Points]";
        return rank;
    }//getRankTitle

    /*
    addTournamentToDatabase Mutator Method:
    * Adds the tournament results to the database.
    - Calls the addOpponentsToDatabase and addTeamResultsToDatabase methods in
      the Database class.
    */
    public void addTournamentToDatabase (){
        db.addOpponentsToDatabase(this);
        db.addTeamResultsToDatabase(this);
    }//addTournamentToDatabase

}//Rank Class
```

Rankings Class

```
package ramlakhan;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

public class Rankings {

    //Properties of the Rankings class:
    private Database db = new Database ();
    private Connection connection = db.getConnection();
    private Rank leaderboard [] = new Rank [10];

    /*
    Rankings Constructor:
    * Creates an array of the top 10 user teams from the Database.
    - Creates a query to join the Teams, Opponents and Leaderboard Tables
      together by matching the same Tournament ID (primary key).
    - Uses a ResultSet to get each value.
    - Stores each value separately in the correct typed variable.
    - Creates a Rank object with the relevant tournament data.
    */
    public Rankings (){
        String username, teamName, tournamentName, gkID, cbID, cdmID, camID,
        stID, wins, draws, losses, goalsScored;
        int userPoints, ai1Points, ai2Points, ai3Points, ai4Points, ai5Points;
        int count = 0;

            try {
            Statement s = connection.createStatement();
            String qry = "SELECT TOP 10 Teams.Username, Teams.[Team Name], "
                    + "Teams.[Tournament Name], Teams.[GK ID], Teams.[CB ID], "
                    + "Teams.[CDM ID], Teams.[CAM ID], Teams.[ST ID], "
                    + "Leaderboard.Points, Leaderboard.Wins, Leaderboard.Draws, "
                    + "Leaderboard.Losses, Leaderboard.[Goals Scored], "
                    + "Opponents.[Team1 Points], Opponents.[Team2 Points], "
                    + "Opponents.[Team3 Points], Opponents.[Team4 Points], "
                    + "Opponents.[Team5 Points]\n" +
                     "FROM Teams, Leaderboard, Opponents\n" +
                     "WHERE  Teams.[Tournament ID] = Opponents.[Tournament ID] "
                    + "AND Teams.[Tournament ID] = Leaderboard.[Tournament ID]\n"
                    + "ORDER BY Leaderboard.Points DESC;";
            ResultSet rs = s.executeQuery(qry);

            while (rs.next() && count < 10){
                username = rs.getString("Username");
                teamName = rs.getString("Team Name");
                tournamentName = rs.getString("Tournament Name");
                gkID = rs.getString("GK ID");
                cbID = rs.getString("CB ID");
                cdmID = rs.getString("CDM ID");
                camID = rs.getString("CAM ID");
                stID = rs.getString("ST ID");
                userPoints = rs.getInt("Points");
```

```
                wins = rs.getString("Wins");
                draws = rs.getString("Draws");
                losses = rs.getString("Losses");
                goalsScored = rs.getString("Goals Scored");
                ai1Points = rs.getInt("Team1 Points");
                ai2Points = rs.getInt("Team2 Points");
                ai3Points = rs.getInt("Team3 Points");
                ai4Points = rs.getInt("Team4 Points");
                ai5Points = rs.getInt("Team5 Points");

                leaderboard [count] = new Rank (username, teamName,
tournamentName,
                gkID, cbID, cdmID, camID, stID, userPoints, wins, draws, losses,
                goalsScored, ai1Points, ai2Points, ai3Points, ai4Points,
ai5Points);
                count++;
            }
            rs.close();
            s.close();
            //System.out.println ("Player Data was successfully accessed from the
database.");
        } catch (Exception e){
            e.printStackTrace();
            System.out.println ("Error transferring leaderboard data!");
        }//try-catch (access Players table and place data into array)
    }//Rankings constructor

    /*
    getRank Accessor Method:
    * Returns a single Rank object based on the position received.
    - Receives the position of the Rank to be returned.
    */
    public Rank getRank (int position){
        return leaderboard [position];
    }//getRank

}//Rankings Class
```

GUI Class Code

Start Class

```
package ramlakhan;
import java.awt.Color;
public class Start extends javax.swing.JFrame {

    public Start() {
        initComponents();

    }

    /*
    btnStartGame Action Perfomed Event Method:
    * Proceeds to the Home screen.
    */
    private void btnStartGameActionPerformed(java.awt.event.ActionEvent evt) {
        new Home ().setVisible(true);
        this.dispose();
    }

    /*
    btnQuit Action Perfomed Event Method:
    * Closes the program.
    */
    private void btnQuitActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();
    }


    // Variables declaration - do not modify
    private javax.swing.JButton btnQuit;
    private javax.swing.JButton btnStartGame;
    private javax.swing.JLabel lblBackgroundImage;
    // End of variables declaration
}
```

Home Class

```
package ramlakhan;
public class Home extends javax.swing.JFrame {

    public Home() {
        initComponents();
    }

    /*
    btnPlayGame Action Perfomed Event Method:
    * Open the UserDetails screen and close the Home screen.
    */
    private void btnPlayGameActionPerformed(java.awt.event.ActionEvent evt) {
        new UserDetails().setVisible(true);
        this.dispose();
    }

    /*
    btnLeaderboard Action Perfomed Event Method:
    * Open the Leaderboard screen and close the Home screen.
    */
    private void btnLeaderboardActionPerformed(java.awt.event.ActionEvent evt) {
        new Leaderboard().setVisible(true);
        this.dispose();
    }

    /*
    btnHelp Action Perfomed Event Method:
    * Open the Help screen as a secondary screen.
    */
    private void btnHelpActionPerformed(java.awt.event.ActionEvent evt) {
        new Help ().setVisible(true);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnHelp;
    private javax.swing.JButton btnLeaderboard;
    private javax.swing.JButton btnPlayGame;
    private javax.swing.JLabel lblBackground;
    // End of variables declaration
}
```

UserDetails Class

```
package ramlakhan;
import java.awt.Color;
import javax.swing.JOptionPane;

public class UserDetails extends javax.swing.JFrame {

    //Variables used for customization:
    private Color errorColour = new Color (255, 0, 0, 30);
    private Color validColour = new Color (0, 255, 0, 30);

    //Variables used for valdiation:
    private boolean validCheck;
    private boolean isLength;
    private boolean isLetter;
    private boolean input;
    private boolean nameValid;
    private boolean teamValid;
    private boolean tournamentValid;
    private boolean allValid;
    private String errorMessage = new String ();

    /*
    Input Variables:
    - These variables are static so that they can be accessed easily by the
      TeamSelection screen.
    */
    public static String username = new String ();
    public static String teamName = new String ();
    public static String tournamentName = new String ();

    public UserDetails() {
        initComponents();

    }

    /*
    btnHelp Action Performed Event Method:
    * Open the Help screen as a secondary screen.
    */
    private void btnHelpActionPerformed(java.awt.event.ActionEvent evt) {
        new Help ().setVisible(true);
    }

    /*
    btnCancel Action Performed Event Method:
    * Return to the Home screen and close the UserDetails screen.
    */
    private void btnCancelActionPerformed(java.awt.event.ActionEvent evt) {
        new Home().setVisible(true);
        this.dispose();
    }

    /*
```

```
    btnStart Action Performed Event Method:
    * Ensures that data input is valid and proceeds to the TeamSelection screen.
    - Gets the username, team name and tournament name from the text fields and
      performs validation on each of them.
    - Determines which fields had valdiation issues and turns them red.
    - If the validation is false, an error message is displayed.
    - If validation is true, the user is taken to the TeamSelection screen and
      is prompted with a message box explaining the TeamSelection process.
    */
    private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
        jtfUsername.setBackground(validColour);
        jtfTeamName.setBackground(validColour);
        jtfTournamentName.setBackground(validColour);

        username = jtfUsername.getText();
        teamName = jtfTeamName.getText();
        tournamentName = jtfTournamentName.getText();

        nameValid = true;
        teamValid = true;
        tournamentValid = true;
        allValid = true;
        errorMessage = "";

        nameValid = validateText (username);
        teamValid = validateText (teamName);
        tournamentValid = validateText (tournamentName);

        if (nameValid == false || teamValid == false || tournamentValid == false){
            allValid = false;
        }//if then

        if (allValid == false){
            if (nameValid == false){
                jtfUsername.setBackground(errorColour);
                errorMessage = errorMessage + " - Username \n";
            }//if username is invalid
            if (teamValid == false){
                jtfTeamName.setBackground(errorColour);
                errorMessage = errorMessage + " - Team Name \n";
            }//if team name is invalid
            if (tournamentValid == false){
                jtfTournamentName.setBackground(errorColour);
                errorMessage = errorMessage + " - Tournament \n";
            } //if tournament name is invalid
            JOptionPane.showMessageDialog(null, "The following entries did not \n"
            + "meet the validation requirments:\n" + errorMessage + "\nPlease "
            + "ensure that validation rules \nhave been followed or that \nthere"
            + " is input!", "Error Starting Game", JOptionPane.ERROR_MESSAGE);
        }//if then
        else if (allValid == true){
            new TeamSelection().setVisible(true);
            JOptionPane.showMessageDialog(null, "Welcome " + username + "! "
            + "(aka Gaffer)\nYou are about to select to your team. \nWhen "
            + "selecting a team, you will see that \neach position has a level "
```

```
            + "followed by a number.\nThis number represents the level of quality
"
            + "of \nthe players you can select for that position.\n\nUnhappy with
"
            + "the quality for a position?\nHit the refresh button at the bottom
left!"
            + "\n\nHINT: Use players that have a good balance offensively and
defensively.",
            "YOU ARE ABOUT TO SELECT TO YOUR TEAM!", JOptionPane.WARNING_MESSAGE);
            this.dispose();
        }//else if

    }

    /*
    valdiateText Accessor and Mutator Method:
    * Validates the inputs made by the user.
    - Receives the text required to validate.
    - Trims spaces that may be present at the end of the string.
    - Performs a length check to check that the length is not longer thant 15.
    - Performs a presence check to ensure that data has been input.
    - Performs a isLetter check to make sure no numbers or special characters
      were used.
    - Returns if the text is valid as a boolean.
    */
    public boolean validateText (String transfer){
        validCheck = true;
        isLength = true;
        isLetter = true;
        input = true;

        transfer.trim();

        int length = transfer.length();
        char currentLetter;

        if (length == 0){
            input = false;
            validCheck = false;
        }//if then
        else if (length > 15){
            isLength = false;
            validCheck = false;
        }//else if

        for (int pos = 0; pos < length; pos++){
            currentLetter = transfer.charAt(pos);
            if (!Character.isLetter(currentLetter)){
                if (currentLetter == ' '){
                    isLetter = true;
                }//if
                else {
                    isLetter = false;
                    validCheck = false;
                }//else
```

```
            }//if then
        }//end for loop

        return validCheck;
    }//validateText

    // Variables declaration - do not modify
    private javax.swing.JButton btnCancel;
    private javax.swing.JButton btnHelp;
    private javax.swing.JButton btnStart;
    private javax.swing.JTextField jtfTeamName;
    private javax.swing.JTextField jtfTournamentName;
    private javax.swing.JTextField jtfUsername;
    private javax.swing.JLabel lblBackground;
    // End of variables declaration
}
```

TeamSelection Class

```
package ramlakhan;
import java.awt.Color;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class TeamSelection extends javax.swing.JFrame {

    //Variables to customize looks:
    private Color comboBoxColour = new Color (246, 26, 156, 10);
    private ImageIcon playerFace;
    private Image scale;

    //Properties of the TeamSelection class:
    //These store the possible selectable players for each position.
    private String [] possibleGKs = new String [5];
    private String [] possibleCBs = new String [5];
    private String [] possibleCDMs = new String [5];
    private String [] possibleCAMs = new String [5];
    private String [] possibleSTs = new String [5];

    //Team object variables
    private String username = UserDetails.username; //static reference
    private String teamName = UserDetails.teamName; //static reference
    private String tournamentName = UserDetails.tournamentName; //static reference
    private String selectedGK = new String ();
    private String selectedCB = new String ();
    private String selectedCDM = new String ();
    private String selectedCAM = new String ();
    private String selectedST = new String ();
    private String gkID = new String ();
    private String cbID = new String ();
    private String cdmID = new String ();
    private String camID = new String ();
    private String stID = new String ();
    private Player tempPlayer;

    /*
    The following has been made static so that the Tournament class can easily
    access the user's team details:
    */
    public static Team selectedUserTeam;

    //Validation Variables:
    private boolean gkInput;
    private boolean cbInput;
    private boolean cdmInput;
    private boolean camInput;
    private boolean stInput;
    private boolean allInput;
    private String errorMessage = new String ();

    /*
```

```
    TeamSelection Construtor:
    * Initializes components.
    - Displays the user's name and team user name.
    - Runs the refreshPlayers method.
    */
    public TeamSelection() {
        initComponents();

        lblUsername.setText(username);
        lblTeamName.setText(teamName);
        refreshPlayers();

    }//TeamSelection


    /*
    refreshPlayers Mutator Method:
    * Refreshes the possible player options and displays them.
    - Calls the generateLOQ method from the Teams class.
    - Calls the method getPossiblePlayers from the Teams class and parses the
      position, loq number and boolean indicating the format required.
    - Sets the label associated with the position to the loq generated.
    - Runs the displayTeamChoices method parsing the possible selections.
        > Discussed further in the displayTeamChoices method.
    */
    public void refreshPlayers (){
        int [] loq = Team.generateLOQ();

        possibleGKs = Team.getPossiblePlayers("GK", ("" + loq [0]), true);
        lblGKplayerFace.setText("Level " + loq [0]);

        possibleCBs = Team.getPossiblePlayers("CB", ("" + loq [1]), true);
        lblCBplayerFace.setText("Level " + loq [1]);

        possibleCDMs = Team.getPossiblePlayers("CDM", ("" + loq [2]), true);
        lblCDMplayerFace.setText("Level " + loq [2]);

        possibleCAMs = Team.getPossiblePlayers("CAM", ("" + loq [3]), true);
        lblCAMplayerFace.setText("Level " + loq [3]);

        possibleSTs = Team.getPossiblePlayers("ST", ("" + loq [4]), true);
        lblSTplayerFace.setText("Level " + loq [4]);

        displayTeamChoices (possibleGKs, possibleCBs, possibleCDMs, possibleCAMs,
possibleSTs);
    }//refreshPlayers



    /*
    displayTeamChoices Mutator Method:
    * Displays the possible player selections into the associated combo box.
    - Receives the arrays containing the possible players for each position.
    - Runs the method displayInComboBox parsing the associated combo box
      and possible player array for the position.
```

```
    */
    public void displayTeamChoices (String [] possibleGK, String [] possibleCB,
            String [] possibleCDM, String [] possibleCAM, String [] possibleST){
        displayInComboBox (jcbGK, possibleGK);
        displayInComboBox (jcbCB, possibleCB);
        displayInComboBox (jcbCDM, possibleCDM);
        displayInComboBox (jcbCAM, possibleCAM);
        displayInComboBox (jcbST, possibleST);
    }//displayTeamChoices

    /*
    * Adds the possible players into a specfic combo box.
    - Receives the position combo box and associated possible players array.
    - Adds the first item as a default which is "---SELECT---".
    - Uses a for loop to run through the array and append the combo box items
      with the players from the possible players array.
    */
    public void displayInComboBox (javax.swing.JComboBox jcb, String []
possiblePlayers){
        jcb.addItem("---SELECT---");

        for (int run = 0; run < 5; run++){
            jcb.addItem(PlayersArray.pullPlayerDetails(possiblePlayers [run]));
        }//endfor loop

    }//displayInComboBox


    /*
    btnRefresh Action Performed Event Method:
    * Disposes the open TeamSelection screen and starts a new one.
    - Used in order to reset the combo boxes and add new selections.
    */
    private void btnRefreshActionPerformed(java.awt.event.ActionEvent evt) {
        new TeamSelection ().setVisible(true);
        this.dispose();
    }

    /*
    btnCancel Action Performed Event Method:
    * Cancels team selections and takes the user back to the home screen.
    */
    private void btnCancelActionPerformed(java.awt.event.ActionEvent evt) {
        new Home ().setVisible(true);
        this.dispose();
    }

    /*
    btnConfirm Action Performed Event Method:
    * Checks that all positions have been selected and proceeds to the Tournament.
    - Performs a presence check validation to ensure that all players have been
      selected.
    - If input is valid, a team is created and the team data is added to the
      Teams table in the database. The program then proceeds to the Tournament
      screen.
```

```
    - If input is not valid, multiple if statements are used to determine which
      positions have not been entered. An error message box is also displayed
      informing the user of non selected positions.
    */
    private void btnConfirmActionPerformed(java.awt.event.ActionEvent evt) {
        //validate that user has selected 5 playersS
        allInput = checkValidity ();
        errorMessage = "";
        if (allInput == true){
            //proceed to next tournament screen
            selectedUserTeam = new Team (username, teamName, tournamentName,
                                         gkID, cbID, cdmID, camID, stID);
            selectedUserTeam.addTeamToDatabase();
            this.dispose();
            new Tournament().setVisible(true);
        }//if then

        else {
            if (gkInput == false){
                errorMessage += "- GK\n";
            }//if then
            if (cbInput == false){
                errorMessage += "- CB\n";
            }//if then
            if (cdmInput == false){
                errorMessage += "- CDM\n";
            }//if then
            if (camInput == false){
                errorMessage += "- CAM\n";
            }//if then
            if (stInput == false){
                errorMessage += "- ST\n";
            }//if then

            JOptionPane.showMessageDialog(null, "Please select a player for: \n"
            + errorMessage, "Missing Selections", JOptionPane.ERROR_MESSAGE);
        }//else

    }

    /*
    checkValidity Mutator Method:
    * Uses all the individual booleans to determine if the overall validation is
      false.
    */
    public boolean checkValidity (){
        boolean valid = true;

        if (gkInput == false || cbInput == false || cdmInput == false || camInput
== false || stInput == false){
            valid = false;
        }//if one of the fields has not been selected

        return valid;
    }//checkValidity
```

```
/*
btnHelp Action Performed Event Method:
* Opens the Help screen as a secondary screen.
*/
private void btnHelpActionPerformed(java.awt.event.ActionEvent evt) {
    new Help ().setVisible(true);
}

/*
displayPlayer Accessor and Mutator Method:
* Displays the player face image and their offensive and defensive stats
  of the selected player and returns a boolean that determines if the player
  is selected or not.
* Performs presence validation and also accesses relevant information that
  is required to create the user's Team.
- Receives the text from the combo box, the array of the possible players,
  a number that indicates the position and the various labels required for
  changing.
- Checks if the combo box's selection is "---SELECT---" and returns false for
  the input.
- If the combo box does not read "---SELECT---":
    > A for loop runs to display the selected players face image by using the
      pullPlayerDetails, getPlayerFromFilePath and pullPlayerFilePath methods
      from the PlayersArray class.
    > Displays the player face and offensive and defensive stats.
    > Gets the playerID of the player using the position number that is
      received.
      0 - GK
      1 - CB
      2 - CDM
      3 - CAM
      4 - ST
    > Uses a switch with position number and sets the correct position
      ID string with the correct player ID. The switch doesn't require a
      default as all possible cases are defines.
*/
public boolean displayPlayer (String selected, String [] possiblePlayers,
int positionNumber, javax.swing.JLabel lblImage, javax.swing.JLabel lblOff,
javax.swing.JLabel lblDef){
    boolean input = true;
    Player selectedPlayer = null;

    if (selected.equals("---SELECT---")){
        lblImage.setIcon(null);
        input = false;
    }//if then
    else {
        for (int run = 0; run < 5; run++){
            if (selected.equals(PlayersArray.pullPlayerDetails
                (possiblePlayers [run]))){

                selectedPlayer = PlayersArray.getPlayerFromFilePath(
                PlayersArray.pullPlayerFilePath(possiblePlayers [run]));
```

```
                playerFace = new ImageIcon (selectedPlayer.getFilePath());

                scale = playerFace.getImage().getScaledInstance(128, 128,
                                                    Image.SCALE_DEFAULT);

                playerFace = new ImageIcon (scale);
                lblImage.setIcon(playerFace);
                lblOff.setText("" + selectedPlayer.getOffStat());
                lblDef.setText("" + selectedPlayer.getDefStat());

                switch (positionNumber){
                    case 0:
                        gkID = selectedPlayer.getPlayerID();
                    break;

                    case 1:
                        cbID = selectedPlayer.getPlayerID();
                    break;

                    case 2:
                        cdmID = selectedPlayer.getPlayerID();
                    break;

                    case 3:
                        camID = selectedPlayer.getPlayerID();
                    break;

                    case 4:
                        stID = selectedPlayer.getPlayerID();
                    break;
                }//end switch
            }//if then
        }//end for loop
    }//else
    return input;
}//displayPlayer

/*
jcbGK Item State Changed Event Method:
* Displays the goalkeeper's player face and stats when combo box item has been
  changed.
- Gets the selected text.
- Calls the displayPlayer method and parses the selected text, the possible
  Striker selections, the number indicating the position and the player face,
  offensive stat and defensive stat labels.
*/
private void jcbGKItemStateChanged(java.awt.event.ItemEvent evt) {
    selectedGK = jcbGK.getSelectedItem().toString();
    gkInput = displayPlayer (selectedGK, possibleGKs, 0, lblGKplayerFace,
                                lblGKoffensiveStat, lblGKdefensiveStat );
}

/*
jcB Item State Changed Event Method:
```

```
    * Displays the centre back's player face and stats when combo box item has
been
      changed.
    - Gets the selected text.
    - Calls the displayPlayer method and parses the selected text, the possible
      Striker selections, the number indicating the position and the player face,
      offensive stat and defensive stat labels.
    */
    private void jcbCBItemStateChanged(java.awt.event.ItemEvent evt) {
        selectedCB = jcbCB.getSelectedItem().toString();
        cbInput = displayPlayer (selectedCB, possibleCBs, 1, lblCBplayerFace,
                                 lblCBoffensiveStat, lblCBdefensiveStat );
    }

    /*
    jcbCDM Item State Changed Event Method:
    * Displays the central defensive midfielder's player face and stats when
      combo box item has been changed.
    - Gets the selected text.
    - Calls the displayPlayer method and parses the selected text, the possible
      Striker selections, the number indicating the position and the player face,
      offensive stat and defensive stat labels.
    */
    private void jcbCDMItemStateChanged(java.awt.event.ItemEvent evt) {
        selectedCDM = jcbCDM.getSelectedItem().toString();
        cdmInput = displayPlayer (selectedCDM, possibleCDMs, 2, lblCDMplayerFace,
                                 lblCDMoffensiveStat, lblCDMdefensiveStat );
    }

    /*
    jcbCAM Item State Changed Event Method:
    * Displays the central attacking midfielder's player face and stats when
      combo box item has been changed.
    - Gets the selected text.
    - Calls the displayPlayer method and parses the selected text, the possible
      Striker selections, the number indicating the position and the player face,
      offensive stat and defensive stat labels.
    */
    private void jcbCAMItemStateChanged(java.awt.event.ItemEvent evt) {
        selectedCAM = jcbCAM.getSelectedItem().toString();
        camInput = displayPlayer (selectedCAM, possibleCAMs, 3, lblCAMplayerFace,
                                 lblCAMoffensiveStat, lblCAMdefensiveStat );
    }

    /*
    jcbST Item State Changed Event Method:
    * Displays the striker's player face and stats when combo box item has been
      changed.
    - Gets the selected text.
    - Calls the displayPlayer method and parses the selected text, the possible
      Striker selections, the number indicating the position and the player face,
      offensive stat and defensive stat labels.
    */
    private void jcbSTItemStateChanged(java.awt.event.ItemEvent evt) {
        selectedST = jcbST.getSelectedItem().toString();
```

```
        stInput = displayPlayer (selectedST, possibleSTs, 4, lblSTplayerFace,
                                    lblSToffensiveStat, lblSTdefensiveStat );
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnCancel;
    private javax.swing.JButton btnConfirm;
    private javax.swing.JButton btnHelp;
    private javax.swing.JButton btnRefresh;
    private javax.swing.JComboBox<String> jcbCAM;
    private javax.swing.JComboBox<String> jcbCB;
    private javax.swing.JComboBox<String> jcbCDM;
    private javax.swing.JComboBox<String> jcbGK;
    private javax.swing.JComboBox<String> jcbST;
    private javax.swing.JLabel lblBackground;
    private javax.swing.JLabel lblCAMdefensiveStat;
    private javax.swing.JLabel lblCAMoffensiveStat;
    private javax.swing.JLabel lblCAMplayerFace;
    private javax.swing.JLabel lblCBdefensiveStat;
    private javax.swing.JLabel lblCBoffensiveStat;
    private javax.swing.JLabel lblCBplayerFace;
    private javax.swing.JLabel lblCDMdefensiveStat;
    private javax.swing.JLabel lblCDMoffensiveStat;
    private javax.swing.JLabel lblCDMplayerFace;
    private javax.swing.JLabel lblGKdefensiveStat;
    private javax.swing.JLabel lblGKoffensiveStat;
    private javax.swing.JLabel lblGKplayerFace;
    private javax.swing.JLabel lblSTdefensiveStat;
    private javax.swing.JLabel lblSToffensiveStat;
    private javax.swing.JLabel lblSTplayerFace;
    private javax.swing.JLabel lblTeamName;
    private javax.swing.JLabel lblUsername;
    // End of variables declaration
}
```

Tournament Class

```
package ramlakhan;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

public class Tournament extends javax.swing.JFrame {

    //Properties of the Tournament class:
    private Team userTeam = TeamSelection.selectedUserTeam; //static access
    private Team aiTeam1 = new Team (userTeam.getTournamentID(), "AI Team 1");
    private Team aiTeam2 = new Team (userTeam.getTournamentID(), "AI Team 2");
    private Team aiTeam3 = new Team (userTeam.getTournamentID(), "AI Team 3");
    private Team aiTeam4 = new Team (userTeam.getTournamentID(), "AI Team 4");
    private Team aiTeam5 = new Team (userTeam.getTournamentID(), "AI Team 5");
    private League tournament = new League (userTeam, aiTeam1, aiTeam2, aiTeam3,
aiTeam4, aiTeam5);
    private int matchweek = 1;
    private String rankTable [] = tournament.getRankTable();
    private String result;
    private League.Mentality mentality;
    private League.InPossession inPossession;
    private League.OutOfPossession outPossession;
    private ImageIcon playerFace;
    private Image scale;


    //A Rank object is made static to be easily accessed by the Results screen:
    public static Rank tournamentRank;

    /*
    Tournament Constructor:
    * Initializes the various components and displays the user's team and the
      first opponent team.
    - Two Message Boxes are displayed prompting the user of what to do.
    */
    public Tournament() {
        initComponents();
        lblTournamentName.setText(userTeam.getTournamentName());

        btnPlayMatch.setVisible(false);
        displayTeam (userTeam, true);
        JOptionPane.showMessageDialog(null, "You are about to play against other "
        + "teams in your tournament! \n\nPoints are acquired as follows: \n +1000
"
        + "points for a WIN \n +500 points for a DRAW \n +200 points for a LOSS \n
"
        + "+100 per GOAL SCORED \n\nYou will play against 5 other teams in "
        + "simulation.", "WELCOME TO THE TOURNAMENT!",
JOptionPane.INFORMATION_MESSAGE);

        displayTeam (aiTeam1, false);
        displayRankTable (rankTable);
```

```
        JOptionPane.showMessageDialog (null, "Before each game, you will be
required "
        + "to select a tactic.\nMake sure you scout your opponent find their "
        + "strengths and \nweaknesses before selecting your tactics. \n\nPress
the"
        + " 'Set Tactics' button located at the bottom of the screen. \nThe
tactics "
        + "you choose will influence game outcomes. \n\nGood Luck Gaffer!",
        "SETTING TACTICS", JOptionPane.INFORMATION_MESSAGE);

    }//Tournament

    /*
    displayTeam Mutator Method:
    * Displays a team in the Tournament
    - Receives the team to display and a boolean that indicates if it is a
      user team or opponent team to be updated.
        > This ensures that the correct components are updated.
    - Displays the name of the team.
    - Runs the displayPlayer method for each position and parses the the player
      object and the 4 associated labels that will be updated.
        > Face Image label
        > Name label
        > Overall label
        > Stats label
    */
    public void displayTeam (Team dTeam, boolean user){
        //Updates an Opponent Team:
        if (user == false){
            lblAIteamName.setText(dTeam.getTeamName());
            displayPlayer (dTeam.getGK(), lblAIFaceGK, lblGKnameAI,
            lblGKoverallAI, lblGKstatsAI);
            displayPlayer (dTeam.getCB(), lblAIFaceCB, lblCBnameAI,
            lblCBoverallAI, lblCBstatsAI);
            displayPlayer (dTeam.getCDM(), lblAIFaceCDM, lblCDMnameAI,
            lblCDMoverallAI, lblCDMstatsAI);
            displayPlayer (dTeam.getCAM(), lblAIFaceCAM, lblCAMnameAI,
            lblCAMoverallAI, lblCAMstatsAI);
            displayPlayer (dTeam.getST(), lblAIFaceST, lblSTnameAI,
            lblSToverallAI, lblSTstatsAI);
        }//if then

        //Updates a User Team:
        else if (user == true){
            lblUSERteamName.setText(dTeam.getTeamName());
            displayPlayer (dTeam.getGK(), lblUserFaceGK, lblGKnameUSER,
            lblGKoverallUSER, lblGKstatsUSER);
            displayPlayer (dTeam.getCB(), lblUserFaceCB, lblCBnameUSER,
            lblCBoverallUSER, lblCBstatsUSER);
            displayPlayer (dTeam.getCDM(), lblUserFaceCDM, lblCDMnameUSER,
            lblCDMoverallUSER, lblCDMstatsUSER);
            displayPlayer (dTeam.getCAM(), lblUserFaceCAM, lblCAMnameUSER,
            lblCAMoverallUSER, lblCAMstatsUSER);
            displayPlayer (dTeam.getST(), lblUserFaceST, lblSTnameUSER,
            lblSToverallUSER, lblSTstatsUSER);
```

```
        }//else if
    }//displayTeam

    /*
    displayPlayer Mutator Method:
    * Updates the labels of a specfic player block.
    - Receives the Player object and the labels assocaiated with the player.
    - Sets the various labels with the correct data.
    - Scales the face image and reduces it's size to 80x80.
    */
    public void displayPlayer (Player dPlayer, JLabel lblFaceImage,
    JLabel lblName, JLabel lblOverall, JLabel lblStats){

        lblName.setText(dPlayer.getPosition() + " - " + dPlayer.getFirstName()
        + " " + dPlayer.getSurname());

        lblOverall.setText("Overall: " + dPlayer.getOverall());

        lblStats.setText("Offensive: " + dPlayer.getOffStat() + " | Defensive: "
        + dPlayer.getDefStat());

        playerFace = new ImageIcon (dPlayer.getFilePath());
        scale = playerFace.getImage().getScaledInstance(80, 80,
Image.SCALE_DEFAULT);
        playerFace = new ImageIcon (scale);
        lblFaceImage.setIcon(playerFace);
    }//displayPlayer

    /*
    displayRankTable Mutator Method:
    * Updates the rank table.
    - Receives the array that contains the sorted rankings.
    - Sets each label to the text from the array in the sorted array.
        [0] - Top Team (First)
        [5] - Bottom Team (Sixth)
    */
    public void displayRankTable (String [] formattedRankings){
        lblRank1Team.setText(formattedRankings [0]);
        lblRank2Team.setText(formattedRankings [1]);
        lblRank3Team.setText(formattedRankings [2]);
        lblRank4Team.setText(formattedRankings [3]);
        lblRank5Team.setText(formattedRankings [4]);
        lblRank6Team.setText(formattedRankings [5]);

    }//displayRankTable

    /*
    btnHelp Action Performed Event Method:
    * Opens the Help screen as a secondary screen.
    */
    private void btnHelpActionPerformed(java.awt.event.ActionEvent evt) {
        new Help().setVisible(true);
    }

    /*
```

```
    btnSetTactics Action Performed Event Method:
    * Opens the Tactic screen.
    - Checks that the amount of matches occured is less than or equal to 5.
        > Opens the Tactics screen and hides the setTactics button.
    - If matchweek is 6:
        > The static tournament Rank is initialized and the Results screen is
          opened. The Tournament screen closes.
    */
    private void btnSetTacticsActionPerformed(java.awt.event.ActionEvent evt) {
        if (matchweek <= 5){
            new Tactics ().setVisible(true);
            btnSetTactics.setVisible(false);
        }
        else if (matchweek == 6){
            tournamentRank = new Rank (tournament);
            new Results ().setVisible(true);
            this.dispose();
        }
    }

    /*
    btnPlayMatch Action Performed Event Method:
    * Runs the necessary process required to simulate the matches of a matchweek
      and update the screens.
    * This button is static as it needs to be chnaged from the Tactics class.
    - Calls the playMatchWeek method from the League class, parsing the matchweek,
      User's selected Mentality, In Possession and Out of Possession Tactics.
        > Return's the User's team result of the specific matchweek.
    - Hides the playMatch button.
    - Uses a switch statement to determine which macthweek was played and what
      visual screen elements need to be updated.
        > Default is not required as all cases are code defined.
    - Updates the Rank Table.
    - Increments the matchweek.
    - Makes the setTactics button visible again.
    */
    private void btnPlayMatchActionPerformed(java.awt.event.ActionEvent evt) {
        mentality = Tactics.mentality;
        inPossession = Tactics.inPossession;
        outPossession = Tactics.outOfPossession;

        result = tournament.playMatchWeek(matchweek, mentality, inPossession,
outPossession);
        btnPlayMatch.setVisible(false);

        switch (matchweek){
            case 1:
                lblResult1.setText(result);
                displayTeam (aiTeam2, false);
            break;

            case 2:
                lblResult2.setText(result);
                displayTeam (aiTeam3, false);
            break;
```

```
            case 3:
                lblResult3.setText(result);
                displayTeam (aiTeam4, false);
            break;

            case 4:
                lblResult4.setText(result);
                displayTeam (aiTeam5, false);
            break;

            case 5:
                lblResult5.setText(result);
                btnSetTactics.setText("End Tournament");
            break;
        }//end switch

        rankTable = tournament.getRankTable();
        displayRankTable (rankTable);
        matchweek++;
        btnSetTactics.setVisible(true);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnHelp;
    public static javax.swing.JButton btnPlayMatch;
    private javax.swing.JButton btnSetTactics;
    private javax.swing.JLabel lblAIFaceCAM;
    private javax.swing.JLabel lblAIFaceCB;
    private javax.swing.JLabel lblAIFaceCDM;
    private javax.swing.JLabel lblAIFaceGK;
    private javax.swing.JLabel lblAIFaceST;
    private javax.swing.JLabel lblAIteamName;
    private javax.swing.JLabel lblBackground;
    private javax.swing.JLabel lblCAMnameAI;
    private javax.swing.JLabel lblCAMnameUSER;
    private javax.swing.JLabel lblCAMoverallAI;
    private javax.swing.JLabel lblCAMoverallUSER;
    private javax.swing.JLabel lblCAMstatsAI;
    private javax.swing.JLabel lblCAMstatsUSER;
    private javax.swing.JLabel lblCBnameAI;
    private javax.swing.JLabel lblCBnameUSER;
    private javax.swing.JLabel lblCBoverallAI;
    private javax.swing.JLabel lblCBoverallUSER;
    private javax.swing.JLabel lblCBstatsAI;
    private javax.swing.JLabel lblCBstatsUSER;
    private javax.swing.JLabel lblCDMnameAI;
    private javax.swing.JLabel lblCDMnameUSER;
    private javax.swing.JLabel lblCDMoverallAI;
    private javax.swing.JLabel lblCDMoverallUSER;
    private javax.swing.JLabel lblCDMstatsAI;
    private javax.swing.JLabel lblCDMstatsUSER;
    private javax.swing.JLabel lblGKnameAI;
    private javax.swing.JLabel lblGKnameUSER;
    private javax.swing.JLabel lblGKoverallAI;
```

```
    private javax.swing.JLabel lblGKoverallUSER;
    private javax.swing.JLabel lblGKstatsAI;
    private javax.swing.JLabel lblGKstatsUSER;
    private javax.swing.JLabel lblRank1Team;
    private javax.swing.JLabel lblRank2Team;
    private javax.swing.JLabel lblRank3Team;
    private javax.swing.JLabel lblRank4Team;
    private javax.swing.JLabel lblRank5Team;
    private javax.swing.JLabel lblRank6Team;
    private javax.swing.JLabel lblResult1;
    private javax.swing.JLabel lblResult2;
    private javax.swing.JLabel lblResult3;
    private javax.swing.JLabel lblResult4;
    private javax.swing.JLabel lblResult5;
    private javax.swing.JLabel lblSTnameAI;
    private javax.swing.JLabel lblSTnameUSER;
    private javax.swing.JLabel lblSToverallAI;
    private javax.swing.JLabel lblSToverallUSER;
    private javax.swing.JLabel lblSTstatsAI;
    private javax.swing.JLabel lblSTstatsUSER;
    private javax.swing.JLabel lblTournamentName;
    private javax.swing.JLabel lblUSERteamName;
    private javax.swing.JLabel lblUserFaceCAM;
    private javax.swing.JLabel lblUserFaceCB;
    private javax.swing.JLabel lblUserFaceCDM;
    private javax.swing.JLabel lblUserFaceGK;
    private javax.swing.JLabel lblUserFaceST;
    private javax.swing.JPanel pnlAICAM;
    private javax.swing.JPanel pnlAICB;
    private javax.swing.JPanel pnlAICDM;
    private javax.swing.JPanel pnlAIGK;
    private javax.swing.JPanel pnlAIST;
    private javax.swing.JPanel pnlResults;
    private javax.swing.JPanel pnlTournamentRanks;
    private javax.swing.JPanel pnlUserCAM;
    private javax.swing.JPanel pnlUserCB;
    private javax.swing.JPanel pnlUserCDM;
    private javax.swing.JPanel pnlUserGK;
    private javax.swing.JPanel pnlUserST;
    // End of variables declaration
}
```

Tactics Class

```
package ramlakhan;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JOptionPane;

public class Tactics extends javax.swing.JFrame {

    /*
    The following enums are made to be static so that the Tournmanet class can
    get the values easily.
    */
    public static League.Mentality mentality = null;
    public static League.InPossession inPossession = null;
    public static League.OutOfPossession outOfPossession = null;

    private boolean primaryValidation = true;
    private boolean mentalityVal = false;
    private boolean inPosVal = false;
    private boolean outOfPosVal = false;

    /*
    Tactics Constructor:
    * Initializes the components of the Tactics screen.
    - Prompts the user to select a tactic for each element through the text
      on the label.
    */
    public Tactics() {
        initComponents();

        txaMentalityDetails.setText("Please select your team Mentality. "
        + "Details of the mentality phase will appear here when it has been
selected.");

        txaInPossessionDetails.setText("Please select a tactic to use when your "
        + "team is In Possession. Details of the tactic will appear here when it "
        + "has been selected.");

        txaOutOfPossessionDetails.setText("Please select a tactic to use when "
        + "your team is Out of Possession. Details of the tactic will appear "
        + "here when it has been selected.");

    }//Tactics

    /*
    rbtnAttacking Action Performed Event:
    * Updates the Mentality Tactic.
    - Displays the relevant details of the Attacking Mentality.
    - Part of the radio button group Mentality.
    */
    private void rbtnAttackingActionPerformed(java.awt.event.ActionEvent evt) {
        mentalityVal = true;
        mentality = League.Mentality.ATTACKING;
        txaMentalityDetails.setText("Your team will dominate the game when in "
```

```
        + "possession of the ball but can be vulnerable to strong offensive "
        + "opposition players.");
    }

    /*
    rbtnDefensive Action Performed Event:
    * Updates the Mentality Tactic.
    - Displays the relevant details of the Defensive Mentality.
    - Part of the radio button group Mentality.
    */
    private void rbtnDefensiveActionPerformed(java.awt.event.ActionEvent evt) {
        mentalityVal = true;
        mentality = League.Mentality.DEFENSIVE;
        txaMentalityDetails.setText("Your team will be stronger defensively but "
        + "may lack composure when attacking.");
    }

    /*
    rbtnBalanced Action Performed Event:
    * Updates the Mentality Tactic.
    - Displays the relevant details of the Balanced Mentality.
    - Part of the radio button group Mentality.
    */
    private void rbtnBalancedActionPerformed(java.awt.event.ActionEvent evt) {
        mentalityVal = true;
        mentality = League.Mentality.BALANCED;
        txaMentalityDetails.setText("Your team will profit offensively and "
        + "defensively and maintain a good balance between them.");
    }

    /*
    rbtnTikiTaka Action Performed Event:
    * Updates the In Possession Tactic.
    - Displays the relevant details of the Tiki Taka In Possession Tactic.
    - Part of the radio button group InPossession.
    */
    private void rbtnTikiTakaActionPerformed(java.awt.event.ActionEvent evt) {
        inPosVal = true;
        inPossession = League.InPossession.TikiTaka;
        txaInPossessionDetails.setText("Players will be more expressive and "
        + "creative which can consequently result in defensive woes.");
    }

    /*
    rbtnShortPassing Action Performed Event:
    * Updates the In Possession Tactic.
    - Displays the relevant details of the Short Passing In Possession Tactic.
    - Part of the radio button group InPossession.
    */
    private void rbtnShortPassingActionPerformed(java.awt.event.ActionEvent evt) {
        inPosVal = true;
        inPossession = League.InPossession.ShortPassing;
        txaInPossessionDetails.setText("Composed and simple build up play that "
        + "will allow your team to attack efficiently and intelligently, "
        + "resulting in less mistakes and defensive errors.");
```

```
    }

    /*
    rbtnFastBuildUp Action Performed Event:
    * Updates the In Possession Tactic.
    - Displays the relevant details of the Fast Build Up In Possession Tactic.
    - Part of the radio button group InPossession.
    */
    private void rbtnFastBuildUpActionPerformed(java.awt.event.ActionEvent evt) {
        inPosVal = true;
        inPossession = League.InPossession.FastBuildUp;
        txaInPossessionDetails.setText("An all out attack system where players "
        + "will dominate offensively but are extremely vulnerable when"
        + " possession is lost.");
    }

    /*
    rbtnTeamPress Action Performed Event:
    * Updates the Out of Possession Tactic.
    - Displays the relevant details of the Team Press Out of Possession Tactic.
    - Part of the radio button group OutOfPossession.
    */
    private void rbtnTeamPressActionPerformed(java.awt.event.ActionEvent evt) {
        outOfPosVal = true;
        outOfPossession = League.OutOfPossession.TeamPress;
        txaOutOfPossessionDetails.setText("Players will press opposition with "
        + "more intensity while being slightly less offensively minded.");
    }

    /*
    rbtnCounterAttack Action Performed Event:
    * Updates the Out of Possession Tactic.
    - Displays the relevant details of the Counter Attack Out of Possession
Tactic.
    - Part of the radio button group OutOfPossession.
    */
    private void rbtnCounterAttackActionPerformed(java.awt.event.ActionEvent evt)
{
        outOfPosVal = true;
        outOfPossession = League.OutOfPossession.CounterAttack;
        txaOutOfPossessionDetails.setText("Players will use their own defensive "
        + "abilities to exploit the opposition's offensive weaknesses but will "
        + "be stronger in attack when possession is regained.");
    }

    /*
    rbtnParkTheBus Action Performed Event:
    * Updates the Out of Possession Tactic.
    - Displays the relevant details of the Park The Bus Out of Possession Tactic.
    - Part of the radio button group OutOfPossession.
    */
    private void rbtnParkTheBusActionPerformed(java.awt.event.ActionEvent evt) {
        outOfPosVal = true;
        outOfPossession = League.OutOfPossession.ParkTheBus;
        txaOutOfPossessionDetails.setText("The team will set up in a way that "
```

```
        + "prevents the opposing team to get through while keeping composure "
        + "when regaining possession.");
    }

    /*
    btnConfirm Action Performed Event Method:
    * Performs a presence validity check and ensures that a selection has been
      made for all 3 tactics.
        > First level is also done by using button group components to
          ensure that only one option is selected for each tactic.
    - Validates selections and prompts the user where selections have not been
      made.
    - If validity is fine, the Tournament playMatch btn is made visible and
      the Tactics screen closes.
    */
    private void btnConfirmActionPerformed(java.awt.event.ActionEvent evt) {
        //Validation:
        String unselectedTactic = "";

        if (mentalityVal == false || inPosVal == false || outOfPosVal == false){
            primaryValidation = false;

            if (mentalityVal == false){
                unselectedTactic += " - Mentality \n";
            }//if then
            if (inPosVal == false){
                unselectedTactic += " - In Possession Tactic \n";
            }//if then
            if (outOfPosVal == false){
                unselectedTactic += " - Out of Possession Tactic \n";
            }//if then
        } else{
            primaryValidation = true;
        }//else

        if (primaryValidation == true){
            Tournament.btnPlayMatch.setVisible(true);
            this.dispose();
        }//if then
        else if (primaryValidation == false){
            JOptionPane.showMessageDialog (null, "The following tactics have not "
            + "been selected: \n" + unselectedTactic, "Unselected Tactics!",
            JOptionPane.ERROR_MESSAGE );
        }//else if
    }


    // Variables declaration - do not modify
    private javax.swing.JButton btnConfirm;
    private javax.swing.ButtonGroup btngInPossession;
    private javax.swing.ButtonGroup btngMentality;
    private javax.swing.ButtonGroup btngOutOfPossession;
    private javax.swing.JLabel lblBackground;
    private javax.swing.JRadioButton rbtnAttacking;
    private javax.swing.JRadioButton rbtnBalanced;
```

```
    private javax.swing.JRadioButton rbtnCounterAttack;
    private javax.swing.JRadioButton rbtnDefensive;
    private javax.swing.JRadioButton rbtnFastBuildUp;
    private javax.swing.JRadioButton rbtnParkTheBus;
    private javax.swing.JRadioButton rbtnShortPassing;
    private javax.swing.JRadioButton rbtnTeamPress;
    private javax.swing.JRadioButton rbtnTikiTaka;
    private javax.swing.JTextArea txaInPossessionDetails;
    private javax.swing.JTextArea txaMentalityDetails;
    private javax.swing.JTextArea txaOutOfPossessionDetails;
    // End of variables declaration
}
```

Results Class

```
package ramlakhan;
public class Results extends javax.swing.JFrame {

    //Properties of the Results class
    private Rank finalRank = Tournament.tournamentRank;

    /*
    Results Constructor:
    * Initalizes components and sets the labels to all it's necessary
      data.
    */
    public Results() {
        initComponents();

        //Basic Details:
        lblTournamentName.setText(finalRank.getTournamentName());
        lblTeamName.setText(finalRank.getUserTeamName());

        //Team Players:
        lblPlayerGK.setText(finalRank.getUserPlayer(0));
        lblPlayerCB.setText(finalRank.getUserPlayer(1));
        lblPlayerCDM.setText(finalRank.getUserPlayer(2));
        lblPlayerCAM.setText(finalRank.getUserPlayer(3));
        lblPlayerST.setText(finalRank.getUserPlayer(4));

        //Ranking Table:
        lblRank1Team.setText(finalRank.getLeagueTableRankings(0));
        lblRank2Team.setText(finalRank.getLeagueTableRankings(1));
        lblRank3Team.setText(finalRank.getLeagueTableRankings(2));
        lblRank4Team.setText(finalRank.getLeagueTableRankings(3));
        lblRank5Team.setText(finalRank.getLeagueTableRankings(4));
        lblRank6Team.setText(finalRank.getLeagueTableRankings(5));

        //Form Table:
        lblStatWins.setText(finalRank.getWins() + " Wins");
        lblStatDraws.setText(finalRank.getDraws() + " Draws");
        lblStatLosses.setText(finalRank.getLosses() + " Losses");
        lblStatGoalsScored.setText(finalRank.getGoalsScored() + " Goals Scored");
    }//Results

    /*
    btnSave Action Performed Event Method:
    * Saves the tournament to the databas and proceeds to the Leaderboard.
    */
    private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
        finalRank.addTournamentToDatabase();
        new Leaderboard ().setVisible(true);
        this.dispose();
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnSave;
    private javax.swing.JLabel lblBackground;
```

```
    private javax.swing.JLabel lblPlayerCAM;
    private javax.swing.JLabel lblPlayerCB;
    private javax.swing.JLabel lblPlayerCDM;
    private javax.swing.JLabel lblPlayerGK;
    private javax.swing.JLabel lblPlayerST;
    private javax.swing.JLabel lblRank1Team;
    private javax.swing.JLabel lblRank2Team;
    private javax.swing.JLabel lblRank3Team;
    private javax.swing.JLabel lblRank4Team;
    private javax.swing.JLabel lblRank5Team;
    private javax.swing.JLabel lblRank6Team;
    private javax.swing.JLabel lblStatDraws;
    private javax.swing.JLabel lblStatGoalsScored;
    private javax.swing.JLabel lblStatLosses;
    private javax.swing.JLabel lblStatWins;
    private javax.swing.JLabel lblTeamName;
    private javax.swing.JLabel lblTournamentName;
    private javax.swing.JPanel pnlFormAndStats;
    private javax.swing.JPanel pnlTeamSquad;
    private javax.swing.JPanel pnlTournament;
    // End of variables declaration
}
```

Leaderboard Class

```
package ramlakhan;
import java.awt.Color;
import javax.swing.border.Border;
import javax.swing.border.LineBorder;
public class Leaderboard extends javax.swing.JFrame {

    //Customization variables:
    private Border selected = new LineBorder (Color.WHITE, 5);
    private Border unselected = new LineBorder (Color.BLACK, 0);

    //Leaderboard Properties:
    private Rankings leaderboard = new Rankings ();

    /*
    Leaderboard Constructor:
    * Initializes components and runs the refreshLeaderboard method.
    */
    public Leaderboard() {
        initComponents();
        refreshLeaderboard();

    }//Leaderboard

    /*
    refreshLeaderboard Mutator Method:
    * Displays the rankings onto the relevant button texts.
    */
    public void refreshLeaderboard (){
        btnRank1.setText("|#1|" + leaderboard.getRank(0).getRankTitle());
        btnRank2.setText("|#2|" + leaderboard.getRank(1).getRankTitle());
        btnRank3.setText("|#3|" + leaderboard.getRank(2).getRankTitle());
        btnRank4.setText("|#4|" + leaderboard.getRank(3).getRankTitle());
        btnRank5.setText("|#5|" + leaderboard.getRank(4).getRankTitle());
        btnRank6.setText("|#6|" + leaderboard.getRank(5).getRankTitle());
        btnRank7.setText("|#7|" + leaderboard.getRank(6).getRankTitle());
        btnRank8.setText("|#8|" + leaderboard.getRank(7).getRankTitle());
        btnRank9.setText("|#9|" + leaderboard.getRank(8).getRankTitle());
        btnRank10.setText("|#10|" + leaderboard.getRank(9).getRankTitle());

    }//refreshLeaderboard method

    /*
    The following methods are used to change the data shown on the right hand
    side of the screen.
    The method will first remove any borders visible around any buttons and put
    a border on the selected button [or ranking].
    The action proceeded will then pull off data from the database and display
    the information required of the team.
    */

    /*
    btnRank Action Performed Event Method:
    * Goes to the Home screen and closes the Leaderboard.
```

```
    */
    private void btnBackActionPerformed(java.awt.event.ActionEvent evt) {
        new Home().setVisible(true);
        this.dispose();
    }

    /*
    btnRank1 Action Performed Event Method:
    * Displays the Team data of the Top 1 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 1 Team.
    */
    private void btnRank1ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank1.setBorder(selected);
        displayRank (leaderboard.getRank(0));
    }

    /*
    btnRank4 Action Performed Event Method:
    * Displays the Team data of the Top 4 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 4 Team.
    */
    private void btnRank4ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank4.setBorder(selected);
        displayRank (leaderboard.getRank(3));
    }

    /*
    btnRank2 Action Performed Event Method:
    * Displays the Team data of the Top 2 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 2 Team.
    */
    private void btnRank2ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank2.setBorder(selected);
        displayRank (leaderboard.getRank(1));
    }

    /*
    btnRank5 Action Performed Event Method:
    * Displays the Team data of the Top 5 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 5 Team.
    */
    private void btnRank5ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank5.setBorder(selected);
```

```
        displayRank (leaderboard.getRank(4));
    }

    /*
    btnRank6 Action Performed Event Method:
    * Displays the Team data of the Top 6 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 6 Team.
    */
    private void btnRank6ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank6.setBorder(selected);
        displayRank (leaderboard.getRank(5));
    }

    /*
    btnRank7 Action Performed Event Method:
    * Displays the Team data of the Top 7 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 7 Team.
    */
    private void btnRank7ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank7.setBorder(selected);
        displayRank (leaderboard.getRank(6));
    }

    /*
    btnRank8 Action Performed Event Method:
    * Displays the Team data of the Top 8 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 8 Team.
    */
    private void btnRank8ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank8.setBorder(selected);
        displayRank (leaderboard.getRank(7));
    }

    /*
    btnRank10 Action Performed Event Method:
    * Displays the Team data of the Top 10 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 10 Team.
    */
    private void btnRank10ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank10.setBorder(selected);
        displayRank (leaderboard.getRank(9));
    }
```

```
    /*
    btnRank3 Action Performed Event Method:
    * Displays the Team data of the Top 3 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 3 Team.
    */
    private void btnRank3ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank3.setBorder(selected);
        displayRank (leaderboard.getRank(2));
    }

    /*
    btnRank9 Action Performed Event Method:
    * Displays the Team data of the Top 9 Team.
    - Shows the selection of the team button by removing borders for all
      other buttons and only showing a border for that rank button.
    - Runs the method displayRank and parses the Top 9 Team.
    */
    private void btnRank9ActionPerformed(java.awt.event.ActionEvent evt) {
        unselectButtons ();
        btnRank9.setBorder(selected);
        displayRank (leaderboard.getRank(8));
    }

    /*
    unselectButtons Mutator Method:
    * Removes borders around all buttons.
    */
    public void unselectButtons (){
        btnRank1.setBorder(unselected);
        btnRank2.setBorder(unselected);
        btnRank3.setBorder(unselected);
        btnRank4.setBorder(unselected);
        btnRank5.setBorder(unselected);
        btnRank6.setBorder(unselected);
        btnRank7.setBorder(unselected);
        btnRank8.setBorder(unselected);
        btnRank9.setBorder(unselected);
        btnRank10.setBorder(unselected);
    }//unselectButtons

    /*
    displayRank Mutator Method:
    * Displays the Team's data on the right hand side of the screen.
    - Receives a Rank object and breaks it down to display the relevant
      information on the screens.
    */
    public void displayRank (Rank display){
        lblTeamAndTournament.setText(display.getUserTeamName() + " - "
        + display.getTournamentName());

        //Rankings Table:
        lblRank1Team.setText(display.getLeagueTableRankings(0));
```

```
        lblRank2Team.setText(display.getLeagueTableRankings(1));
        lblRank3Team.setText(display.getLeagueTableRankings(2));
        lblRank4Team.setText(display.getLeagueTableRankings(3));
        lblRank5Team.setText(display.getLeagueTableRankings(4));
        lblRank6Team.setText(display.getLeagueTableRankings(5));

        //Team Players:
        lblPlayerGK.setText(display.getUserPlayer(0));
        lblPlayerCB.setText(display.getUserPlayer(1));
        lblPlayerCDM.setText(display.getUserPlayer(2));
        lblPlayerCAM.setText(display.getUserPlayer(3));
        lblPlayerST.setText(display.getUserPlayer(4));

        //Form:
        lblStatWins.setText(display.getWins() + " Wins");
        lblStatDraws.setText(display.getDraws()+ " Draws");
        lblStatLosses.setText(display.getLosses()+ " Losses");
        lblStatGoalsScored.setText(display.getGoalsScored()+ " Goals Scored");

    }//displayRank

    // Variables declaration - do not modify
    private javax.swing.JButton btnBack;
    private javax.swing.JButton btnRank1;
    private javax.swing.JButton btnRank10;
    private javax.swing.JButton btnRank2;
    private javax.swing.JButton btnRank3;
    private javax.swing.JButton btnRank4;
    private javax.swing.JButton btnRank5;
    private javax.swing.JButton btnRank6;
    private javax.swing.JButton btnRank7;
    private javax.swing.JButton btnRank8;
    private javax.swing.JButton btnRank9;
    private javax.swing.JLabel lblBackground;
    private javax.swing.JLabel lblPlayerCAM;
    private javax.swing.JLabel lblPlayerCB;
    private javax.swing.JLabel lblPlayerCDM;
    private javax.swing.JLabel lblPlayerGK;
    private javax.swing.JLabel lblPlayerST;
    private javax.swing.JLabel lblRank1Team;
    private javax.swing.JLabel lblRank2Team;
    private javax.swing.JLabel lblRank3Team;
    private javax.swing.JLabel lblRank4Team;
    private javax.swing.JLabel lblRank5Team;
    private javax.swing.JLabel lblRank6Team;
    private javax.swing.JLabel lblStatDraws;
    private javax.swing.JLabel lblStatGoalsScored;
    private javax.swing.JLabel lblStatLosses;
    private javax.swing.JLabel lblStatWins;
    private javax.swing.JLabel lblTeamAndTournament;
    private javax.swing.JPanel pnlFormAndStats;
    private javax.swing.JPanel pnlTeamSquad;
    private javax.swing.JPanel pnlTournament;
    // End of variables declaration
}
```

Help Class

```
package ramlakhan;
import java.awt.Color;
import javax.swing.ImageIcon;
import javax.swing.border.Border;
import javax.swing.border.LineBorder;
public class Help extends javax.swing.JFrame {

    //Customization Properties:
    Border selected = new LineBorder (Color.WHITE, 3);
    Border unselected = new LineBorder (Color.BLACK, 0);
    private ImageIcon helpContent;

    /*
    Help Constructor:
    * Intitialize components.
    */
    public Help() {
        initComponents();
    }

    /*
    btnPointsAndRanksHelp Action Performed Event Method:
    * Displays the Points And Ranks Help Image.
    - Shows button selection and displays the help image.
    */
    private void btnPointsAndRanksHelpActionPerformed(java.awt.event.ActionEvent
evt) {
        selectionDisplay();
        btnPointsAndRanksHelp.setBorder(selected);

        //display help content image
        helpContent = new ImageIcon
(".//imageResources//help//pointsAndRanks.png");
        lblHelpContentImage.setIcon (helpContent);
    }

    /*
    btnTeamSelectionHelp Action Performed Event Method:
    * Displays the Team Selection Help Image.
    - Shows button selection and displays the help image.
    */
    private void btnTeamSelectionHelpActionPerformed(java.awt.event.ActionEvent
evt) {
        selectionDisplay();
        btnTeamSelectionHelp.setBorder(selected);

        //display help content image
        helpContent = new ImageIcon
(".//imageResources//help//teamSelection.png");
        lblHelpContentImage.setIcon (helpContent);
    }

    /*
```

```
    btnBasicsHelp Action Performed Event Method:
    * Displays the Basics Help Image.
    - Shows button selection and displays the help image.
    */
    private void btnBasicsHelpActionPerformed(java.awt.event.ActionEvent evt) {
       selectionDisplay();
        btnBasicsHelp.setBorder(selected);

        //display help content image
        helpContent = new ImageIcon (".//imageResources//help//basic.png");
        lblHelpContentImage.setIcon (helpContent);
    }

    /*
    selectionDisplay Mutator Method:
    * Clears borders on all buttons and removes the initial text.
    */
    public void selectionDisplay (){
        btnTeamSelectionHelp.setBorder(unselected);
        btnPointsAndRanksHelp.setBorder(unselected);
        btnBasicsHelp.setBorder(unselected);
        lblHelpContentImage.setText("");
    }//selectionDisplay

    // Variables declaration - do not modify
    private javax.swing.JButton btnBasicsHelp;
    private javax.swing.JButton btnPointsAndRanksHelp;
    private javax.swing.JButton btnTeamSelectionHelp;
    private javax.swing.JLabel lblBackground;
    private javax.swing.JLabel lblHelpContentImage;
    // End of variables declaration
}
```

Externally Sourced Code

Random Numbers:

The process behind generating the numbers 1 to 5 in a random order and not storing duplicates was done by using a class called SecureRandom. The code was adapted from the YouTube video below made by 'Improve Your Programming Skills'.
https://www.youtube.com/watch?v=Q8OWCDlFoVo&list=PL79nsbeypYKEFBUfWTczAixnTFOVqPin6&index=3

The Match Simulation:

The algorithm designed to simulate matches was done by me, but the idea of making the matches work with probability and other elements came from Daniel Jung, another matric student in my Information Technology class.

Using the Combo Boxes:

The process involving changing the image of a player's face when a different player has been selected was done by adapting the code used in the below YouTube video made by 'Knowledge to Share'. The video provided the adequate information on how to use an item listener and update contents once a selection has been made.
https://www.youtube.com/watch?v=YxCm49O4zt8&list=PL79nsbeypYKEFBUfWTczAixnTFOVqPin6&index=2&t=1s

Reviewing and Debugging:

I had regular chats regarding my PAT progress with my I.T Teacher, Mr G Kenning. Mr Kenning often guided me during the development and was helped me find errors that would occur.

External Review:

I spoke to another I.T Teacher in my school, Mrs E. Emmett who offered a second opinion regarding the development and documentation of my PAT.

Player Face Images:

All player faces and names were sourced from Futbin.
https://www.futbin.com/

Backgrounds:

**Only backgrounds that had images are referenced:*

Home Screen: (Old Trafford)
https://lh3.googleusercontent.com/fHFuvYvswTXvkEl9zLqo8rDHVGpz39igO28xAJeOyrT0U77IBCRl2O00gfWovhZ9XTlv_Sg=s151

UserDetails Screen: (Ole Gunnar Solksjaer)
https://e0.365dm.com/19/01/2048x1152/skysports-ole-gunnar-solskjaer_4551615.jpg

TeamSelection Screen: (Antony Martial Goal)
https://lh3.googleusercontent.com/NhGo7ncurctvBOGatbHil2UlToEhf4yrfGLAtC2Nl9cY4UtkYBFym3QZ71SPlkav-480=s132

Tournament Screen: (Marcus Rashford Goal)
https://lh3.googleusercontent.com/CTUP-nk-MSZvrvwAun93bJF3jaESaqC-XsEzp4o-B6WNgDfjdw7NcVq6S1TuDZMtQ5GXhHc=s163

Results Screen: (Bruno Fernandes and Antony Martial Celebrating)
https://lh3.googleusercontent.com/-nOV7rYqmooh_lly7qABx851_A4NgKPgyA8Sd5ayhmR7NPWZ438zihkiGuR3qAZsw2bc5w=s157

Leaderboard Screen: (Sir Alex Ferguson)
https://lh3.googleusercontent.com/3x9JI1GaMMHNqv5urxE1jlyOW-_GJ24ZD7CHJZWa5ywnyQR5jyCqcGQ973ib4Vm9XLM1=s157

Help Screen: (Tactics Board)
https://lh3.googleusercontent.com/CBioHcF9ZvysNRBn-UyJskw3W8myzlUB9Bxv6RWJwlpEA7lmq3Hmv6cXz7hn7gCWD-la=s163

Explanation of Critical Algorithms

Core Algorithm: The process behind simulating a match

This algorithm is critical to ensuring that any team can essentially win a match. This process allows the entire game to work off probability and user choices, making the game more interactive and less predictable.

Key:
Team x represents the offensive team and takes the offensive stat of players. Team y represents the defensive team and takes the defensive stat of players.

1. Initialize Team 1 and Team 2's running goals to 0.
2. Determine the boosts of each team and keep track of these boosts by storing them in a number array.

* For first run:
Team x is Team 1
Team y is Team 2

3. Instantiate two numbers, one to store Team x Player's offensive stat and the other to store Team y Player's defensive stat.

4. Initialize Team x's offensive boost to the selected Mentality offensive boost.

5. Initialize Team y's defensive boost to the selected Mentality defensive boost.

6. Run the processes upcoming in line 7 to 9 five times replacing the square brackets each time with the following player position:

Run	Team x	Team y
1	Goalkeeper	Striker
2	Centre Back	Central Attacking Midfielder
3	Central Defensive Midfielder	Central Defensive Midfielder
4	Central Attacking Midfielder	Centre Back
5	Striker	Goalkeeper

7. Store ***[Team x Player's]*** stat set to the player's offensive stat multiplied by Team x's offensive boost.

8. Store ***[Team y Player's]*** stat set to the player's defensive stat multiplied by Team y's defensive boost.

9. Determine the match outcome and goals by checking the following:

If Player x's stat is greater than Player y's stat:

- Set Team x's Offensive Boost to the sum of their Mentality Offensive Boost and the InPossession Offensive Boost.
- Set Team y's Defensive Boost to the sum of their Mentality Defensive Boost and their OutOfPossession Defensive Boost.
- Calculate the difference between Player x's stat and Player y's stat.
- Append Teams x's goals by the difference divided by 10.

Else if Player y's stat is greater than Player x's stat:

- Set Team x's Offensive Boost to the sum of their Mentality Offensive Boost and the OutOfPossesion Offensive Boost.
- Set Team y's Defensive Boost to the sum of their Mentality Defensive Boost and their InPossession Defensive Boost.
- Calculate the difference between Player y's stat and Player x's stat.
- Append Teams y's goals by the difference divided by 10.

10. Repeat the processes from line 3 to 9 but switch the teams, hence: Make Team x be Team 2 and Team y be Team 1

11. Round off both teams' goals to the nearest whole number.

12. Update the record of the teams' using the following rules:

If Team 1's goals are greater than Team 2's goals:

- Add a win for Team 1 and a loss for Team 2 and their respective goals scored.

Else if Team 1's goals are equal to Team 2's goals:

- Add a draw for both teams and their respective goals scored.

Else if Team 2's goals are greater than Team 1's goals:

- Add a loss for Team 1 and a win for Team 2 and their respective goals.

13. Update the points for both teams by doing the following:

Initialize the points to 0.
Append the points by the number of wins multiplied by 1000.
Append the points by the number of draws multiplied by 500.
Append the points by the number of losses multiplied by 200.
Append the points by the number of goals scored multiplied by 100.

Advanced Techniques

Writing and Reading from a Database in Java

This process was showed a new way of storing data in secondary storage and accessing it from a Java program. The process behind ensuring that the SQL statements were written correctly to do what was expected required a lot of skill and patience to get the statements perfect.

Doing processing in a non-GUI class

It was required that separate classes had to be made to handle any type of processes in the program. This made the program extremely difficult to code as it often led to complication and multiple classes being used at once. However, at the end, the best possible working classes were created to make the program work.

Generating Random Numbers

This method required the use of the SecureRandom class to generate the numbers 1 to 5 and place in a random order without duplication. This method allowed for the team selections to be fair. The method was integrated well with the TeamSelection screen and the behind the scenes process of generating 5 random opponent teams. The same method was used to get the possible players and choose a random player from the possible choices.

Simulating the Matches

This algorithm took time, effort and patience to create. After many errors and bugs, the perfect algorithm to process the matches was designed. This was also an advanced technique as it required multiple classes in order to perfect the algorithm and make the code as efficient and little as possible.

Using Static Methods and Properties

The concept of the word 'static' had never made sense to me until I was required to implement it into my program. Static methods and variables made the program a lot easier to design and made the transfer of data across GUI classes a lot simpler.

Using enums

Enums removed the need to use chars to determine which tactics were used. The use of an enum simplified the way the tactics could be parsed across classes and made more logical sense when coding.

Using itemListeners

The process behind immediately changing the player's face image and stats once they had been selected from the combo box.

Dynamic Updating of Components

Most components were dynamically updated in the Tournament screen. This involved a process that requiring parsing the correct components to change and display the correct team information, results and rankings table.