Criterion C: Development

Java UserStat Class

Variables

```
//Setting Up Variable
private String agent;
private int kill = 0;
private int death = 0;
private int assist = 0;
private boolean win = false;
private int gameNumber;
private int id;

//Setting Decimal to 2 Places
private static DecimalFormat dfSharp = new DecimalFormat("#.##");
```

Initializing variables that are going to be used in the UserStat class along with the decimal format library which will let the program round double numbers to two decimal places.

Constructor

```
//Empty Constructor
UserStat(){

}

//Setter Constructor Without ID
UserStat(String agent, int kill, int death, int assist, boolean win){
    this.agent = agent;
    this.kill = kill;
    this.death = death;
    this.assist = assist;
    this.win = win;
}

//Setter Constructor
UserStat(String agent, int kill, int death, int assist, boolean win, int gameNumber, int id){
    this.agent = agent;
    this.kill = kill;
    this.death = death;
    this.assist = assist;
    this.win = win;
    this.gameNumber = gameNumber;
    this.id = id;
}
```

Creating constructors for the UserStat class. The first one being the empty constructor, while the second and third constructor take in data in order to set the global variables in the UserStat class. The difference between the second and third constructor is that it doesn't take in id, which is generated through MySql, and used for uploading the data to MySql. For the third constructor, it is used to retrieve the data from MySql along with the id generated by MySql and the game's number. The id is used in order to communicate between the program and the database when deleting data from the program and database. This will be explained in more detail later on. The game's number will let the user easily see the corresponding match with the graph.

Getter Method

```
//Getter
public  String getAgent(){
    return this.agent;
}
public  int getKill(){
    return this.kill;
}
public  int getDeath(){
    return this.death;
}
public  int getAssist(){
    return this.assist;
}
public  boolean getWin(){
    return this.win;
}
public  int getID(){
    return this.id;
}
public String[] getDataAsString(){
    String temp[] = {String.valueOf(gameNumber), agent, String.valueOf(kill), String.valueOf(death),
        String.valueOf(assist), String.valueOf(win), String.valueOf(calKDA())};
    return temp;
}
```

Creating getter methods for the UserStat class. All of them except getDataAsString() will return the global variables as its data type when being called. For getDataAsString(), it will return an array of global variables along with the KDA value, which is calculated through a method. This method is used for the purpose of displaying the data in the match history table (JTable).

KDA Calculation Method

```
//Claculating KDA Method
public  double calKDA() {
    return Double.parseDouble(dfSharp.format((double) (kill+assist)/death));
}
```

This method is used in order to calculate the value of KDA.

1. Kill plus assist
2. Divided the number by death
3. Parsed into double in order to keep it preciseness
4. Format through decimal format class to two decimal places
5. Parsed one more time to double
6. Return the value.

This method needs to be created separately and not done through when initializing the variables because it will crash the program since death is initialized as 0, which is impossible to divide mathematically.

Structure of the Program

The program consists of 2 tabs. The first one being the “Add Stat” page where the user can input data and save it in the MySql table for the program to use. The second page is the “View Stat” page where the user can see the KDA graph, match history, and all the stats.

The program itself doesn't have much of a structure because the program depends on the interaction between the user and the clickable button.

Java mainGUI Class

GUI

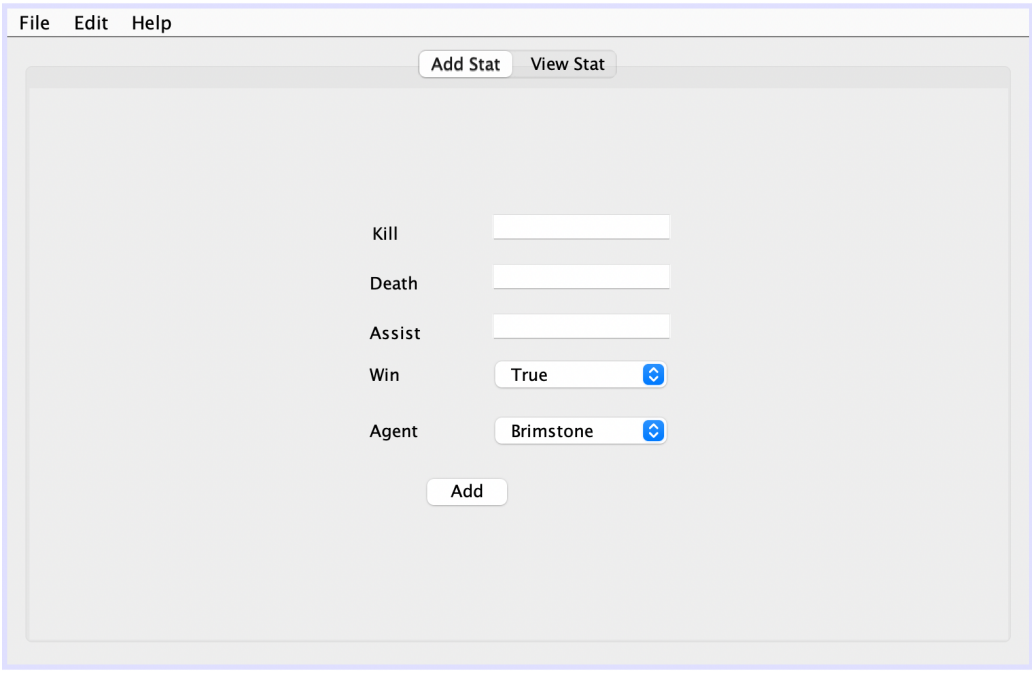
This program uses already existing Java Swing tools. Java Swing tools used in this program are JTextField, JLabel, JComboBox, JButton, and JPanel. JTextField lets the user input a string to the program. JLabel is a label that is used to indicate something. JComboBox lets the user choose input provided by the program. JButton lets the user click and interact with the program. Lastly, JPanel, in this program, is used to display the KDA graph.

Variables

```
//Initializing Stack and Linkedlist of UserStat Class
LinkedList<UserStat> usStack = new LinkedList<UserStat>();
LinkedList<UserStat> usLinkedList = new LinkedList<UserStat>();
```

Initializing linked list in the global variables which will be used to store data retrieved from MySql. Both of the linked lists will be storing the UserStat class. The difference between the two is that usStack will be acting as a stack while usLinkedList will be acting as a normal linkedlist.

Add Stat Page



Add Button

```
//Storing User's Input Into UserStat Class
UserStat us = new UserStat(agentComboBox.getSelectedItem().toString(), Integer.parseInt(killTextBox.getText()),
        Integer.parseInt(deathTextBox.getText()), Integer.parseInt(assistTextBox.getText()),
        Boolean.parseBoolean(winLoseComboBox.getSelectedItem().toString()));
```

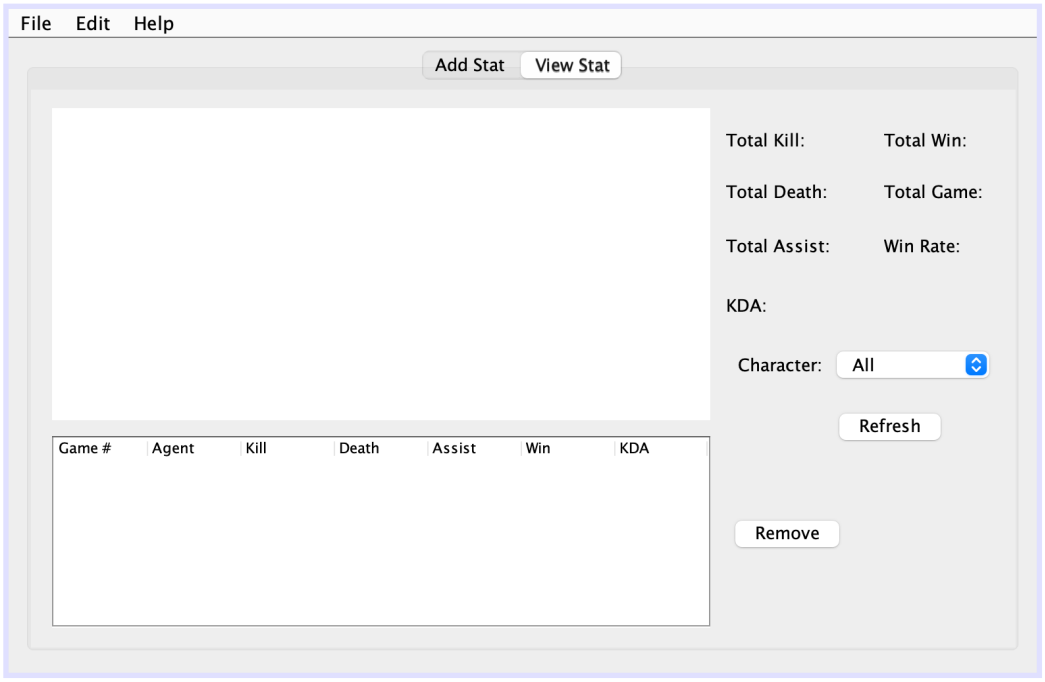
Creating a new instance of the UserStat class as “us”, which uses the constructor without the id value. The data that are inputted into UserStat class are from text boxes and combo boxes on the “Add Stat” page. For the text boxes, the string inputted by the user is parsed into integer through the uses Interger.parseInt() method. The agentComboBox object is changed to string with the use of toString() method. Lastly, the winLoseComboBox object is first changed to string with the uses of toString() method and then parsed into boolean through the uses of Boolean.parseBoolean() method.

```
//Check If User Input Data In Every Textbox
if(killTextBox.getText().equals(null) || deathTextBox.getText().equals(null) || assistTextBox.getText().equals(null)){
    //Display Message Dialog to User Saying that the User Needs to Fill In All the Textbox
    JOptionPane.showMessageDialog(this, "Please Fill All the Boxes");
} else {
    try{
        //Connecting to MySql
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/sys","root","12345678");

        //MySql Syntax for Inserting with User's Input Dta
        String query = "INSERT INTO sys.user (pkill, death, assist, win, agent) VALUES (" + us.getKill() + ", "
                + us.getDeath() + ", " + us.getAssist() + ", " + us.getWin() + ", '" + us.getAgent() + "')";

        //Creating Statement and Executing the MySql Syntax
        Statement stmt = con.createStatement();
        stmt.executeUpdate(query);

        //Closing the Connection
        con.close();

        //Displaying Message Dialog that the Data Has Been Succesfully Added
        JOptionPane.showMessageDialog(this, "Added Successfully");

        //Reseting the Text Box and Combo Box
        killTextBox.setText("");
        deathTextBox.setText("");
        assistTextBox.setText("");
        winLoseComboBox.setSelectedIndex(0);
        agentComboBox.setSelectedIndex(0);

        //Updating View Stat Page
        updateViewStat();

    } catch (Exception e){
        //Print Error
        System.out.println("error");
        System.out.println(e.getMessage());
    }
}
```

1. The program checks if the user has input something in all the textboxes by examining the string value being returned from the text box equal to “null” or not.
2. If it does, it shows a pop up window saying “Please Fill All the Boxes”.
3. If none of the text boxes is equal to “null”, it tries to connect to MySql.
4. The program creates a statement which is used to insert new data into the MySql table with the use of MySql syntax.
5. The program inserts new data into the database.
6. The program has a close connection between itself and MySql.
7. The program displays a pop up window saying “Added Successfully”.
8. The program reset all the text boxes to empty and set the item in the combo boxes to be the first one.
9. The program calls updateViewStat().

If there is an error when trying to connect to MySql, the program will print out “error” and the error message.

View Stat page



View Stat Update Method

```
public void updateViewStat(){
    try{
        //Setting Decimal to 2 Places
        DecimalFormat dfSharp = new DecimalFormat("#.##");

        //Getting Match History Table
        DefaultTableModel mhTable = (DefaultTableModel) matchHistoryTable.getModel();

        //Set Row to 0 Inorder to Clear the Table
        mhTable.setRowCount(0);

        //Clearing User Stat LinkedList and Stack
        usLinkedList.clear();
        usStack.clear();

        //Initializing Variables to Store Total Number of Each Stat
        int totalKill = 0;
        int totalDeath = 0;
        int totalAssist = 0;
        int totalWin = 0;
        int totalGame = 0;
        int gameNumber = 1;

        //Connecting to MySql
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/sys","root","12345678");

        //MySql Syntax for Selecting Data
        String sql = "select * from sys.user";

        //Creating Statement and Executing the MySql Syntax to Get the Data
        Statement stmt = con.createStatement();

        ResultSet rs = stmt.executeQuery(sql);
```

1. The program initializes variables, classes, and objects that will be used.
2. The program clears the match history table by setting the row count to 0 and clears both the UserStat stack and the linked list.
3. The program tries to connect itself with MySql.
4. The program retrieves data from the MySql table.
5. The program stores data retrieved from the MySql table in ResultSet rs.

```
//While There Is Still Data Left in MySql, Keeps Retrieving
while(rs.next()){
    //Getting the Agent's Name From MySql
    String agent = rs.getString("agent");

    //Check If the User Wants to See Specific Agents's Match History and Stats
    if(!agentFilterComboBox.getSelectedItem().equals("All")){
        //Matching the Agent's Name that the User Wants to See With Data in MySql
        if(agent.equals(agentFilterComboBox.getSelectedItem())){

            //Initializing Variables to Store Data From MySql
            int kill = rs.getInt("pkill");
            int death = rs.getInt("death");
            int assist = rs.getInt("assist");
            boolean win = rs.getBoolean("win");
            int id = rs.getInt("id");

            //Storing These Data in UserStat Class
            UserStat us = new UserStat(agent, kill, death, assist, win, gameNumber, id);

            //Adding MySql Data to the Total Stat Variables
            totalGame += 1;
            totalKill += kill;
            totalDeath += death;
            totalAssist += assist;
            gameNumber ++;

            //Check If that Game the User Won or Lost
            if(win){
                //If the User Won, Increase the Total Win by 1
                totalWin += 1;
            }

            //Pusing the UserStat Class Into User Stat Stack
            usStack.push(us);
            usLinkedList.add(us);
        }
```

1. The program checks if there is data left in rs.
2. If yes, the program gets the agent's name that is being retrieved at the moment.
3. The program checks if the combo box selected item is not equal to “All”.
4. If yes, the program retrieves and stores that data in variables.
5. These variables are then passed into an instance UserStat class with the id and game’s number constructor.
6. The program adds the total game and game number by one.
7. The program adds total kill, total death, and total assist by the value currently retrieved from the MySql table.
8. The program checks if win is true.
9. If yes the program adds win by one.
10. The program pushes this instance of UserStat class in the usStack and adds this instance of UserStat class in the usLinkedList

```
            } else {
                //Initializing Variables to Store Data From MySql
                int kill = rs.getInt("pkill");
                int death = rs.getInt("death");
                int assist = rs.getInt("assist");
                boolean win = rs.getBoolean("win");
                int id = rs.getInt("id");

                //Storing These Data in UserStat Class
                UserStat us = new UserStat(agent, kill, death, assist, win, gameNumber, id);

                //Adding MySql Data to the Total Stat Variables
                totalGame += 1;
                totalKill += kill;
                totalDeath += death;
                totalAssist += assist;
                gameNumber ++;

                //Check If that Game the User Won or Lost
                if(win){
                    //If the User Won, Increase the Total Win by 1
                    totalWin += 1;
                }

                //Pusing the UserStat Class Into User Stat Stack
                usStack.push(us);
                usLinkedList.add(us);
            }
        }
```

This is the second part of the retrieving data code. However, if the combo box equals “All”, the program will not check the agent’s name being retrieved with the value returned by the combo box. This means that all the data in the MySql table will be retrieved. And the rest of the process stays the same as above.

```
        //Initializing the KDA and Win Rate After Getting the Data Inorder to Avoid Dividing by Zero
        double totalKDA = Double.parseDouble(dfSharp.format((double) (totalKill + totalAssist)/totalDeath));
        double winRate = Double.parseDouble(dfSharp.format((double) totalWin/totalGame));
```

The program calculates the total KDA the same way as it does in the UserStat class. Moreover, it also calculates the win rate by using total win divided by the total game.

```
        //While User Stat Stack Is Not Empty, Keep Adding the Table
        while(!usStack.isEmpty()){
            //Poping Data From User Stat Stack to the Match History Table
            mhTable.addRow(usStack.pop().getDataAsString());
        }
```

After the program retrieves all the necessary data from MySql, it will start displaying them. First, the program displays a match history table.

1. The program checks if usStack is not empty.
2. While it is not empty, the program pops the newest instance of UserStat class pushed and displays it on top of the table

The reason that it needs to call getDataAsString() method is because the mhTable.addRow takes in an array of strings in order to display them.

```
        //Creating Dtatset for the Graph
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for(int i = 0; i < usLinkedList.size(); i ++){
            dataset.setValue(usLinkedList.get(i).calKDA(), "KDA", String.valueOf(i + 1));
        }

        //Creating Chart
        JFreeChart linechart = ChartFactory.createLineChart("","Game Number","KDA", dataset,
                PlotOrientation.VERTICAL, false, false, false);

        //Creating Plot Object
        CategoryPlot lineCategoryPlot = linechart.getCategoryPlot();
        // lineCategoryPlot.setRangeGridlinePaint(Color.BLUE);
        lineCategoryPlot.setBackgroundPaint(Color.white);

        //Creating Render Object In Order to Change the Color of the Line
        LineAndShapeRenderer lineRenderer = (LineAndShapeRenderer) lineCategoryPlot.getRenderer();
        Color lineChartColor = new Color(0,102,204);
        lineRenderer.setSeriesPaint(0, lineChartColor);
```

```
        //Displaying the Graph
        ChartPanel lineChartPanel = new ChartPanel(linechart);
        kdaGraph.removeAll();
        kdaGraph.add(lineChartPanel, BorderLayout.CENTER);
        kdaGraph.validate();
```

After displaying data on the match history table. The program will display the KDA graph. First, it creates a data set that will be used by the graph through a for loop. It loops through all data in usLinkedList and takes the KDA value of each one. It also sets the x-axis value of the graph by adding one to i and converting it into string. Then the program creates a line chart and displays it through the use of an external library.

```
        //Setting the Text to Display the Stats
        killLabel.setText("Total Kill: " + totalKill);
        deathLabel.setText("Total Death: " + totalDeath);
        assistLabel.setText("Total Assist: " + totalAssist);
        kdaLabel.setText("KDA: " + totalKDA);
        totalGameLabel.setText("Total Game: " + totalGame);
        totalWinLabel.setText("Total Win: " + totalWin);
        winRateLabel.setText("Win Rate: " + winRate);
```

Lastly, the program sets all the labels with the corresponding data retrieved and calculated from rs.

```
    } catch (Exception e){
        //Printing Error
        System.out.println("error");
        System.out.println(e.getMessage());
    }
}
```

In case there is an error, the program will print “error” and follow with the error message.

Remove Button Method

```
try{
    //Initializing Object and Variables
    DefaultTableModel mhTable = (DefaultTableModel) matchHistoryTable.getModel();
    int selectedRow = matchHistoryTable.getSelectedRow();
    int id = usLinkedList.get((usLinkedList.size() - selectedRow) - 1).getID();
```

Initializing match history table objects and variables. The variable selectedRow stores the current that the user is selecting, which gets from the getSelectedRow() method. The variable id is found by calling getID() method. The position of the LinkedList can be found by finding the difference between the size of the linked list and the selectedRow. This is because the selected row is in the opposite direct proportional position of the linked list. For example, if the match on the very top is being selected, the selected row is 0, meaning that it is the last instance of UserStat class added to the linked list, since it is the first one to be popped out, meaning that it is the last one to be pushed in. Thus, the size minus by 0 will return the collect position of the instance of the User Stat class in the linked list.

```
//Check How Many Row Are Selected
if(matchHistoryTable.getSelectedRowCount() == 1){
    //Remove the Row
    mhTable.removeRow(matchHistoryTable.getSelectedRow());

} else {
    JOptionPane.showMessageDialog(this, "Please Select Single Row For Delete");
}
```

The program checks if only one row is being selected, if ture, the program will remove that row. If there is a multiple of rows being selected, the program will pop up a window saying “Please Select Single Row For Delete”.

```
//Connecting to MySql
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/sys","root","12345678");

//MySql Syntax for Deleting User's Input Dta
String query = "DELETE FROM sys.user WHERE id=" + id;

//Creating Statement and Executing the MySql Syntax
Statement stmt = con.createStatement();
stmt.executeUpdate(query);

//Closing the Connection
con.close();
```

The program connects itself with MySql and executes a delete statement in order to delete a corresponding row being deleted in the program in the MySql table. To delete something off the MySql table, it requires a condition to tell which one to be deleted, in this case the id. Without the id, MySql can’t tell which one to delete. Thus, in order to correctly delete the data selected on the program on the MySql table, it needs to have an id.

```
//Displaying Message Dialog that the Data Has Been Succesfully Deleted
JOptionPane.showMessageDialog(this, "Deleted Successfully");

//Displaying the View Stat
updateViewStat();
```

Lastly, the program pops up a window saying “Deleted Successfully” and calls updateViewStat() in order to update the page.

```
} catch (Exception e){
    //Print Error
    System.out.println("error");
    System.out.println(e.getMessage());
}
```

In case there is an error, the program will print “error” and follow with the error message.

Refresh Button

```
updateViewStat();
```

The program calls the updateViewStat() method in order to update the page.

Word Count: 1141 (Does not include bullet points)