CS5003NI Data Structure and Specialist Programming

30% Individual Coursework

2023-24 Autumn

Student Name: samman majgainya

London Met ID: 22085773

College ID: np01ai4s230012

Assignment Due Date: Friday, January 12, 2024

Assignment Submission Date: Sunday, December 1, 2024

Word Count: 6181

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Contents

1Introduction

The coursework named the data structure and Specialist Programming has been created by our module leader Mr. Prithivi Maharjan with the purpose to design the smoothest experiences to the user for the cricket management system. As per the requirements users of the system can update personal information, search for addresses, create new records, delete old ones, and carry out other activities connected to maintaining address information.

One of the major key features of the coursework are.

1. create new data.

user can add the data like jersey no, name, address, role, and other stuff in the table which will help to user for the system.

2 Update the data.

user can update the data in the table whenever it is necessary.

3 delete.

user can delete the data in the table.

4 sorts.

Here we can apply the sorting feature where users can do for the maximum value. The sorting feature like insertion sort and merge sort.

5 searches

now the user can search for the data.

1.1 Aims and objectives.

The major aims of the coursework include.

- Data management
  with this user can get the smooth experience in the data management.
- smooth user experience
  the program should contain the smooth user experience with the proper exceptions and messages.
- functionality expansions
  it will help the expansions of the functionality based on the user.
- other aims include that user should have the proper programming conventions.

- also, it should have the proper use of the classes and methods.

2 Algorithms utilized.

The definition of the algorithm is that it is the set of well-defined instructions to solve the problem with the set of the required input and required output. (programiz, n.d.)

The major uses of the algorithms are.

They help to solve the problems effectively and efficiently.

they help to automate process. (RishabhPrabhu, n.d.).

2.1 algorithm used for the add player.

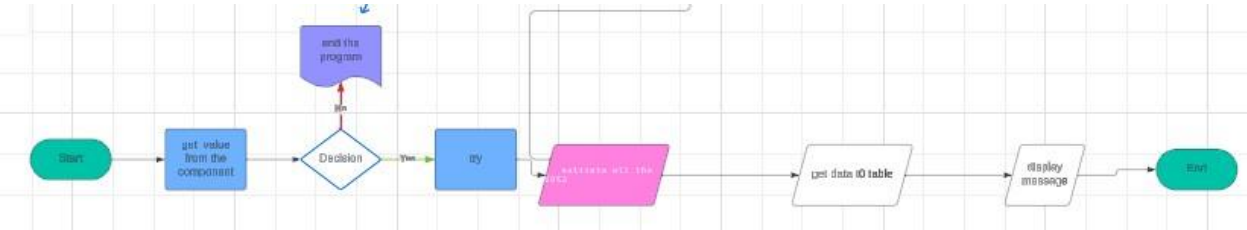

Figure 1algorithm for the add of the data

now lets breakdown for the alogrithms

2.1.1 jersey number

- get data from the text field.
- check the jersey number is empty or not.
- make sure that it has the numbers starting from the 1 to 15 as there are 13 players in the cricket team only. and check for the unique value in the jersey number.

2.1.2player name address status and role

- Verify whether they are not empty or not and also does not contain any alphanumeric characters.

2.1.3 top score

- make sure that it should not be empty and fall in between the range of the 300- 800.

2.1.4 total matches

- make sure that these should not be empty and fall between the range of the 300-500.

2.1.5 exceptions handling

handle all the required exceptions as per the entry from the user.

- use the try catch block for the exceptions handling

2.1.6 display message

At last display the success message for the user in the j option pane.

2.2 update player

the flowchart for the update player is

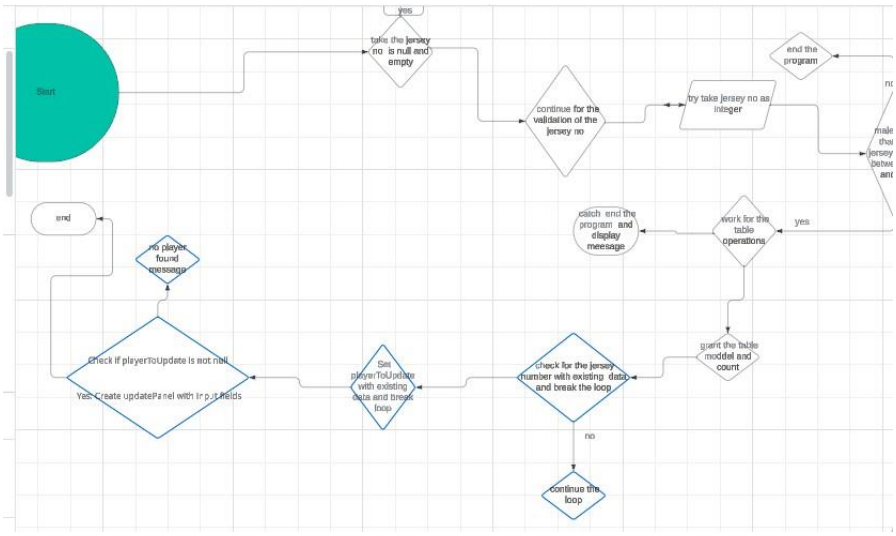

Figure 2 flowchart for the update player

now describe for the algorithm for the update player is

2.2.1 input

- ask the user using the j option pane for the jersey number.
- trim and store the jersey number by the help of the text field.

2.2.2 jersey number

- validate the jersey number should not be empty and fall in between the 1 and 15.
- make sure jersey number is the integer only.

2.2.3 work related the table operations

- work in the table related operations with the help of the addplayer table
- now count the total number of the row

2.2.4 player search

- initialize each array to null.
- use the for each loop in order to search the player.

2.2.4 update player

- check for the update player as null and not null.
- If true, create an update panel (updatePanel) with input fields for player information (name, address, role, status, marriage status, top score, total runs).
- if not, display the error message.

2.3.4end the algorithm.

2.3 delete player.

1. Get the selected row index from addplayertable1.
2. Check if the selected row index is not equal to -1.
    - If true, continue to row deletion.
    - If false, display an error message indicating that a row needs to be selected for deletion.
3. If a row is selected, access the DefaultTableModel associated with addplayertable1 and remove the selected row using tableModel.removeRow(selectedRow).

4 The algorithm concludes.

3 class diagram

The general-purpose modelling language which has been used to visualize the system for the development process

benefits of the class diagram

- can implement the oops in the simple way
- make communication more clear and real
- helps to acquire the entire system (Bhumika_Rani, 2018)

3.1 overall class diagram

3.2 individual class diagram

3.2.1 individual class diagram for the add player.

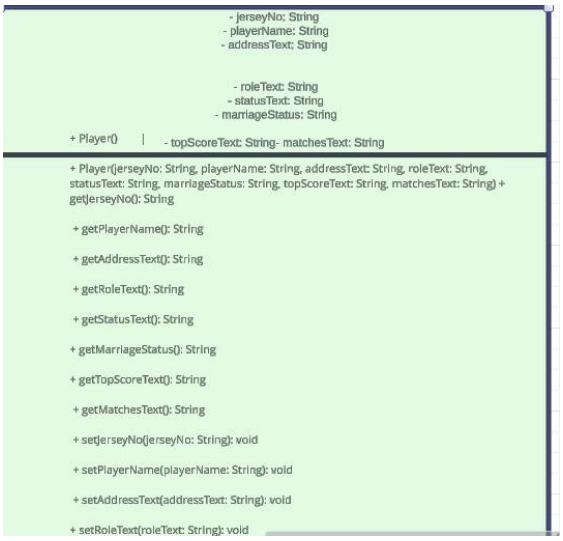

Figure 3 class diagram for the add of the player

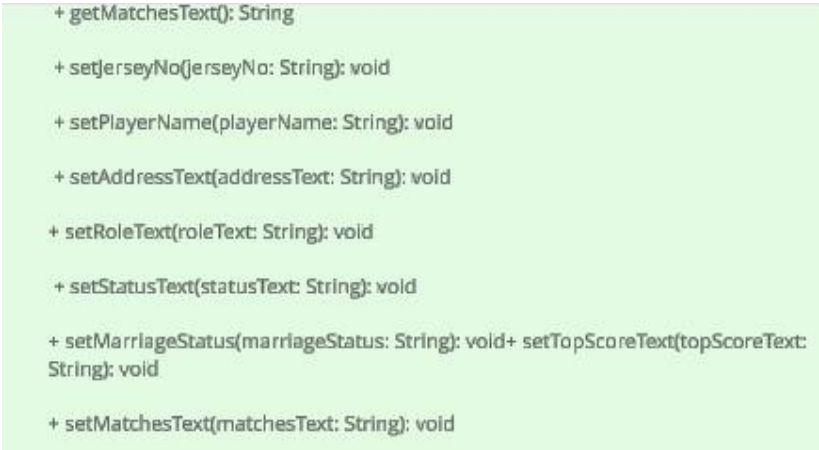

Figure 4 remaining class diagram for add player

3.2.2 update of the player

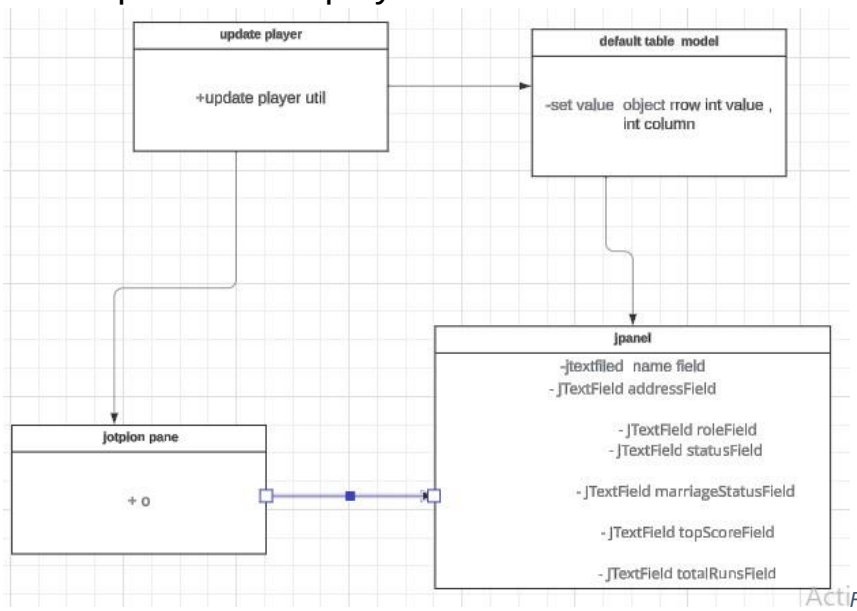

Figure 5 figure for update of player

3.2.3 delete of the player

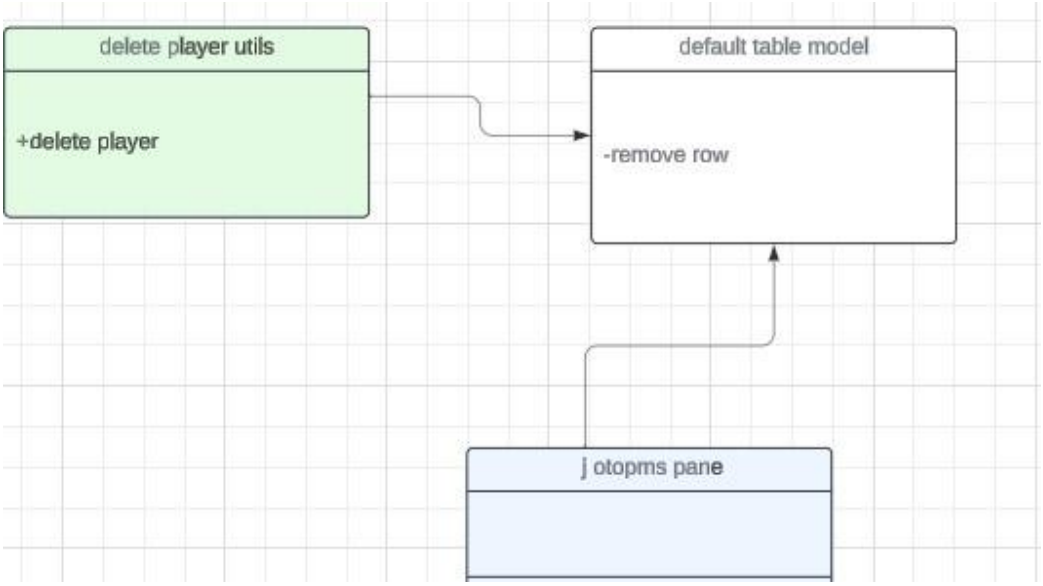

Figure 6 class diagram for the update of the player

3.2.4 binary search

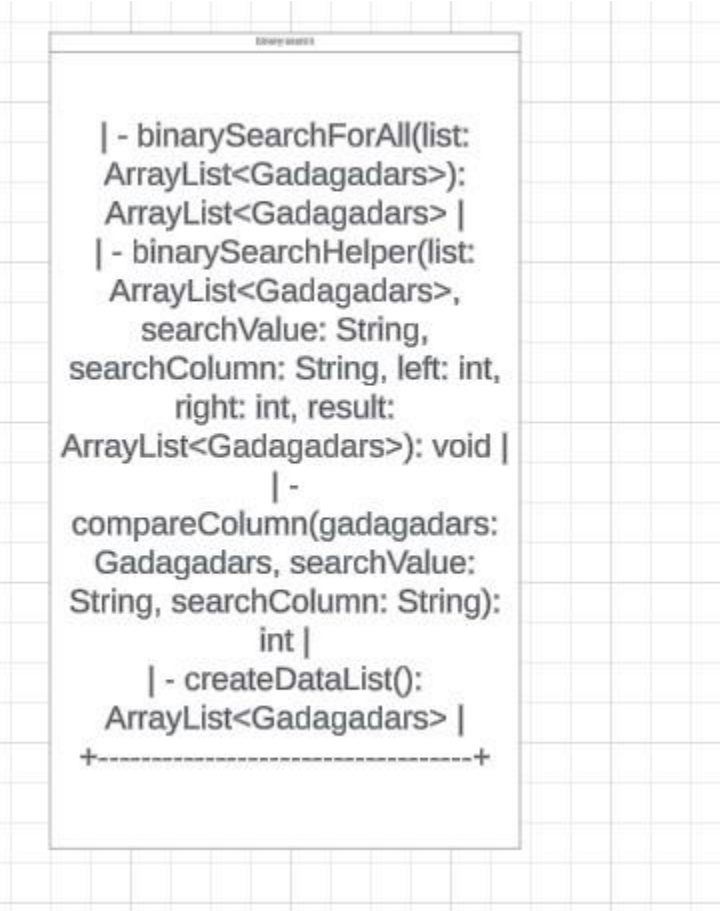

Figure 7 binary search

3.2.5 merge sort

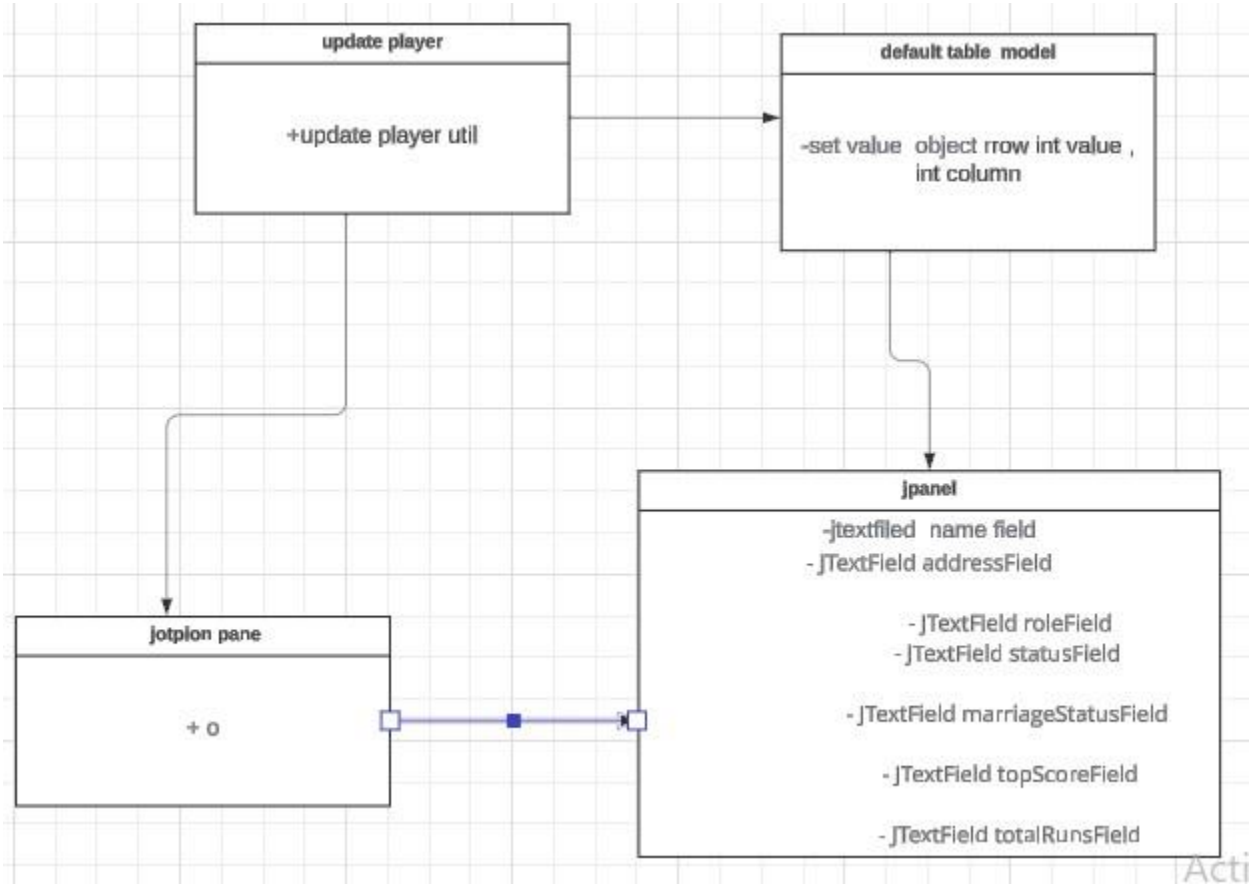

Figure 8 merge sort

4 method descriptions

method is the block of the code which only runs whenever it is called. they are also known as the functions in the java. (w3schools, n.d.).

several methods have been used in this coursework in order to complete the coursework in the given period of the time.

4.1 class add player

4.1.1 overall method used.

the following table is used to describe the different method used for the class add player

Method used	Described
Add student. set visible(true)	It will set the add student panel to true
Complete code. Set visible(false)	It will set the complete code panel to false it means the user will not be able to see the add student panel.
Repaint ()	Whenever the change is required, this method is called
Revalidate ()	It is used to cause the layout manager whenever the component are added and deleted dynamically;

Table 1 for the add player

```
    private void addplayerActionPerformed(java.awt.event.ActionEvent evt) {
// this is used to add the player in the main panel
/*
 the major purpose  of  the add player is that it will redirect to the user for the  next  panel  which consist al
 user  like textfield and so on
*/
        addstudent.setVisible(aFlag: true
        );
completecode.setVisible(aFlag: false);
repaint();
revalidate();

    }
```

Figure 9 method for the add student

4.2class addstudent1

the main purpose of this method is that it is used for the add of the student in the panel it consists of all the parameter like jersey no, name , address, role , status, marriage status, top score, total matches and so on

4.2.1 method used.

Method used	descriptions
Get text ()	It will receive the text from the Gui components.
Trim ()	It is used to remove the unnecessary whitespace from the text.
Get selected item ()	It used to get the value from the combo box.
Parse INT ()	It is used to get the integer value from the user.
Matches ()	Esure that they only accept the integer on the basis of the given regular expressions.
Add row ()	Used to add row in the table
Show message ()	Used to display the message in the user.
Print stack trace method ()	Used to debugging the user by providing the given informations about where exceptions occurred

Table 2 for the add player table

```
    private void addstudent1ActionPerformed(java.awt.event.ActionEvent evt) {
/*
        it is used to get text  for the value recieved by the  user for the data recived
       parameter  used jersey no, player name , address , role status  marrigestatus top score , matches
        method used get text and trim
       */
     String jerseyNoText = jerseynol.getText().trim();
String playerName = name.getText().trim();
String addressText = address.getText().trim();
String roleText = role.getText().trim();
String statusText = status.getText().trim();

String marriageStatus = (String) marriagestatus.getSelectedItem();
String topScoreText = topscore.getText().trim();
String matchesText = totalmatches.getText().trim();
DefaultTableModel tableModel = (DefaultTableModel) addplayertable1.getModel();// used to  updtate the value in th

try {
```

Previous Next Select

Figure 10 figure for the add player table

```java
try {
    if (!jerseyNoText.isEmpty()) {
        int jerseyNo = Integer.parseInt(s: jerseyNoText);
        if (jerseyNo < 1 || jerseyNo > 15) {
            throw new IllegalArgumentException(s: "Jersey number must be between 1 and 15");
        }// check for the jersey number is empty or not ot it should be in between the value of 1 and 1

        // Check if the jersey number is already in the table
        for (int row = 0; row < tableModel.getRowCount(); row++) {
            if (tableModel.getValueAt(row, column: 0).equals(obj:jerseyNoText)) {
                throw new IllegalArgumentException(s: "Jersey number must be unique");
            }
        }
    } else {
        throw new IllegalArgumentException(s: "Jersey number cannot be empty");// display the message
    }

    if (playerName.isEmpty()) {
        throw new IllegalArgumentException(s: "Player name cannot be empty");// check for the player nam
```

Figure 11 remaing value for add player table.

```java
if (addressText.isEmpty()) {
    throw new IllegalArgumentException(s: "Address cannot be empty"); // check for the address should be empty
}

if (roleText.isEmpty()) {
    throw new IllegalArgumentException(s: "Role cannot be empty");// check for thee  role is empty
}

if (statusText.isEmpty()) {
    throw new IllegalArgumentException(s: "Status cannot be empty");// check for the status is empty or not
}

// Additional checks for alphanumeric characters in playerName, addressText, roleText, and statusText
if (!playerName.matches(regex: "[a-zA-Z ]+")) {
    throw new IllegalArgumentException(s: "Player name should only contain alphabets and spaces");
}

if (!addressText.matches(regex: "[a-zA-Z0-9 ]+")) {
```

Figure 12 remaining value for add player table

```
    // Additional checks for alphanumeric characters in playerName, addressText, roleText, and statusText
    if (!playerName.matches(regex: "[a-zA-Z ]+")) {
        throw new IllegalArgumentException(s: "Player name should only contain alphabets and spaces");
    }

    if (!addressText.matches(regex: "[a-zA-Z0-9 ]+")) {
        throw new IllegalArgumentException(s: "Address should only contain alphanumeric characters and spaces");
    }

    if (!roleText.matches(regex: "[a-zA-Z ]+")) {
        throw new IllegalArgumentException(s: "Role should only contain alphabets and spaces");
    }

    if (!statusText.matches(regex: "[a-zA-Z ]+")) {
        throw new IllegalArgumentException(s: "Status should only contain alphabets and spaces");
    }
// check for the top score should be in between the 300 and 800
    if (!topScoreText.isEmpty()) {
        int topScore = Integer.parseInt(s: topScoreText);
        if (topScore < 300 || topScore > 800) {
```

Figure 13 remaining code for add player table

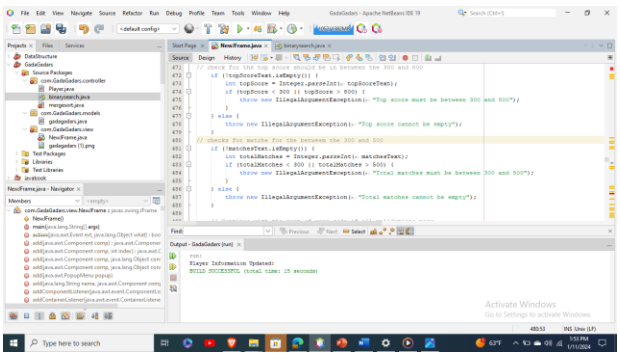

Figure 14 remaining code for add player

```
    // Continue with the rest of your code if all validations pass
    Object[] rowData = {jerseyNoText, playerName, addressText, roleText, statusText, marriageStatus, topScoreText,
    tableModel.addRow(rowData);

    // Display success message
    JOptionPane.showMessageDialog(parentComponent:null, message:"Player added successfully", title: "Success", messageType:JO

} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Invalid input for jersey number, top score, or total ma
    e.printStackTrace(); // Handle or log the exception as needed
} catch (IllegalArgumentException e) {
    JOptionPane.showMessageDialog(parentComponent:null, message:e.getMessage(), title: "Error", messageType:JOptionPane.ERROR
    e.printStackTrace(); // Handle or log the exception as needed
} catch (Exception e) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"An unexpected error occurred", title: "Error", messageType:
    e.printStackTrace(); // Handle or log the exception as needed
}
```

Figure 15 remianing value for add player

4.3 class update player

it is used to update value of player from the table it has only option pane and panel in the code.

4.3.1 method used in the table

The table is used to describe the method used

Method used	Descriptions
Jersey number text ()	It is used to ask the user for the jersey number which has to accept only the integer value and all the value between the 1 and 15.
Jerseynumber text== null()	It is used to accept the null value from the user.
Jersey number matches ==	Used to format the digits from the user
Integer .phraseint	Used to accept the integer value with the minimum value from 1 and max value of 15.
Show message dialog	Used to show the message for the user
Row count	Used to count the row in the table
model.getDataVector().elementAt(i).toArray();	Used to count the value in the table
Update panel()	Create the update panel in the text field
Update panel(). set value	It used to set the value in the table
Jtextfield ()	Used to create the textdfield
Update panel.add new jlabel	Used to create the j label.
Joption pane	Used to show the message

Table 3 method used for update table.

```
String jerseyNoText = JOptionPane.showInputDialog(message:"Enter Jersey Number:").trim();

// Check if jersey number is null or empty
if (jerseyNoText == null || jerseyNoText.isEmpty()) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Jersey number cannot be empty", title: "Error", messageType
    return; // Exit the method if there's an error
}

// Validate the entered jersey number for alphanumeric characters
if (!jerseyNoText.matches(regex: "\\d+")) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Invalid Jersey number format (only digits are allowed)"
    return; // Exit the method if there's an error
}

// Parse the jersey number to an integer
int jerseyNo;
try {
    jerseyNo = Integer.parseInt(s: jerseyNoText);
```

Figure 16 used for the update table

```
    // Validate the range of the jersey number
    if (jerseyNo < 1 || jerseyNo > 15) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Jersey number must be between 1 and 15", title: "Erro
        return; // Exit the method if there's an error
    }
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Invalid Jersey number format", title: "Error", messageType:
    return; // Exit the method if there's an error
}

// Get the table model and row count
DefaultTableModel model = (DefaultTableModel) addplayertable1.getModel();
int rowCount = model.getRowCount();

// Search for the player with the specified jersey number
Object[] playerToUpdate = null;
for (int i = 0; i < rowCount; i++) {
    int currentJerseyNo = Integer.parseInt(s: model.getValueAt(row:i, column: 0).toString());

    if (currentJerseyNo == jerseyNo) {
```

ut - GadaGadars (run) ×

Figure 17 used for the update table

```
if (playerToUpdate == null)
{...4 lines }

for (int i = 0; i < rowCount; i++) {
    int currentJerseyNo = Integer.parseInt(s: model.getValueAt(row:i, column: 0).toString());

    if (currentJerseyNo == jerseyNo) {

        playerToUpdate = model.getDataVector().elementAt(index: i).toArray();

        break;
    }
}

if (playerToUpdate != null) {
    // Create the update panel
    JPanel updatePanel = new JPanel();
    updatePanel.setLayout(new GridLayout(rows: 4, cols: 4));
```

Figure 18 for update player

```
if (playerToUpdate != null) {
    // Create the update panel
    JPanel updatePanel = new JPanel();
    updatePanel.setLayout(new GridLayout(rows: 4, cols: 4));
/*
     used to get the textfiled
    @param used name, address, role , status, marrige status, topscore and total runs
    */
    JTextField nameField = new JTextField(columns:8);
    JTextField addressField = new JTextField(columns:8);
    JTextField roleField = new JTextField(columns:8);
    JTextField statusField = new JTextField(columns:8);
    JTextField marriageStatusField = new JTextField(columns:8);
    JTextField topScoreField = new JTextField(columns:8);
    JTextField totalRunsField = new JTextField(columns:8);
```

Figure 19 continued update player

```
/*
     used to create the jlabel
    @param used name, address, role status, marrige staatus , topscore, total matches played
    */
    updatePanel.add(new JLabel(text: "Name:"));
    updatePanel.add(comp: nameField);
    updatePanel.add(new JLabel(text: "Address:"));
    updatePanel.add(comp: addressField);
    updatePanel.add(new JLabel(text: "Role:"));
    updatePanel.add(comp: roleField);
    updatePanel.add(new JLabel(text: "Status:"));
    updatePanel.add(comp: statusField);
    updatePanel.add(new JLabel(text: "Marriage Status:"));
    updatePanel.add(comp: marriageStatusField);
    updatePanel.add(new JLabel(text: "Top Score:"));
    updatePanel.add(comp: topScoreField);
    updatePanel.add(new JLabel(text: "Total matches played:"));
    updatePanel.add(comp: totalRunsField);
```

Figure 20 update player

```
 if matches the  result should have the   updated named and contains all the  value
@param used upated name , updated address, updated role, updated status, updaated marriage status  updated topsco
total runs
method used get text  and trim

*/
if (result == JOptionPane.OK_OPTION) {

    String updatedName = nameField.getText().trim();
    String updatedAddress = addressField.getText().trim();
    String updatedRole = roleField.getText().trim();
    String updatedStatus = statusField.getText().trim();
    String updatedMarriageStatus = marriageStatusField.getText().trim();
    String updatedTopScore = topScoreField.getText().trim();
    String updatedTotalRuns = totalRunsField.getText().trim();
```

Figure 21 update player

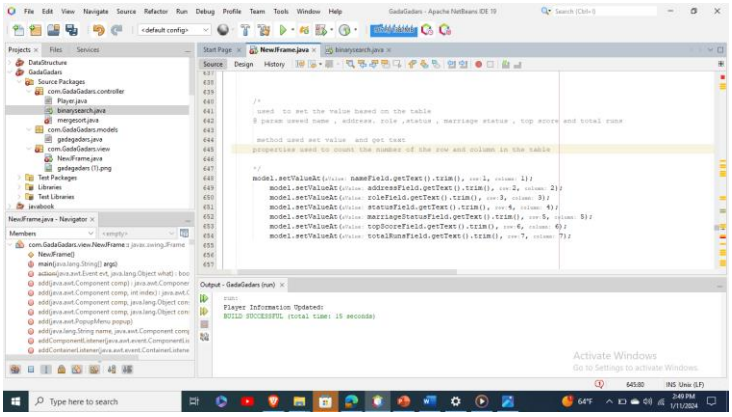

Figure 22 used for the update playe

4.3 delete player

it is used to delete the player based on the row of the user

4.3.1 method used.

Method used	Descriptions
Get selected row()	Used to get the row
Remove row ()	Used to remove the row

Table 4 method for the delete

```
    private void deleteplayerActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
int selectedRow = addplayertable1.getSelectedRow();
if (selectedRow != -1) {
    DefaultTableModel tableModel = (DefaultTableModel) addplayertable1.getModel();
    tableModel.removeRow(row:selectedRow);
} else {
    JOptionPane.showMessageDialog(parentComponent:null, message:"Please select a row to delete", title: "Error", messageType:
}

    }
```

Figure 23 method used in the delete

4.4 merge sort

4.4.1 method used in the merge sort

Method name	Descriptions
if (list.size() <= 1)	It used to fix the list is less then or equal to 1
Return list	Used to return the list in the table
int mid	Used to calculate the midpoint of the list
ArrayList<T> left	Used to create the array list
ArrayList<T> right	Used to create the array list
left = mergeSort	Call the merge sort in the leftcells
Right = mergeSort	Call the merge sort in the right index
ArrayList<T>	Used to calculate for the merge result
(leftIndex < left.size() && rightIndex < right.size())	Used to compare the element using the value
return	Used to return the value in the list
playerNameComparator	Used to compare by the player name
ArrayList<Gadagadars> sortedList	Used to show the sorted list in the table
main	It is the main method of the system

Table 5 method used in the merge sort

```java
import com.GadaGadars.models.Gadagadars;
import com.GadaGadars.models.Gadagadars.Gadagadars;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
//import java.util.Comparator;
//import java.util.List;
//import javax.swing.JTable;
//import javax.swing.table.DefaultTableModel;

/**
 *
 *  this is the controller class used for the merge sort
 */
public class mergesort {

/*
     it used for the return of the array list
    @param used list, string property , comparater and comparator
    */

  public <T> ArrayList<T> mergeSort(ArrayList<T> list, String property, Comparator<T> comparator) {
      if (list.size() <= 1) {
```

Figure 24 method used for merge sort

```java
private <T> ArrayList<T> merge(ArrayList<T> left, ArrayList<T> right, String property, Comparator<T> co
    ArrayList<T> result = new ArrayList<>();
    int leftIndex = 0;
    int rightIndex = 0;
    // test for the index of the element

    while (leftIndex < left.size() && rightIndex < right.size()) {
        if (comparator.compare(o1: left.get(index: leftIndex), o2: right.get(index: rightIndex)) < 0) {
            result.add(e: left.get(index: leftIndex));
            leftIndex++;
        } else {
            result.add(e: right.get(index: rightIndex));
            rightIndex++;
        }
    }
    //add all the element

    result.addAll(c: left.subList(fromIndex: leftIndex, toIndex:left.size()));
    result.addAll(c: right.subList(fromIndex: rightIndex, toIndex:right.size()));

    return result;
}
```


Figure 25 method used for the merge sort

```java
    public static void main(String[] args) {
        // Create an instance of the mergesort class
        mergesort mergeSort = new mergesort();

        // Example: Create a list of Gadagadars
        ArrayList<Gadagadars> gadagadarsList = new ArrayList<>();
        gadagadarsList.add(new Gadagadars(jerseyNo: "1", playerName: "John", addressText: "Address1", roleText: "Role1",
        gadagadarsList.add(new Gadagadars(jerseyNo: "2", playerName: "Jane", addressText: "Address2", roleText: "Role2",

        // Define a comparator for sorting by player name
        Comparator<Gadagadars> playerNameComparator = Comparator.comparing(Gadagadars::getPlayerName);

        // Perform the merge sort
        ArrayList<Gadagadars> sortedList = mergeSort.mergeSort(list: gadagadarsList, property: "playerName", comp

        // Display the sorted list
        System.out.println(x: "Sorted List:");
        for (Gadagadars gadagadars : sortedList) {
            System.out.println("Jersey No: " + gadagadars.getJerseyNo() +
                    ", Player Name: " + gadagadars.getPlayerName() +
                    ", Address: " + gadagadars.getAddressText() +
                    ", Role: " + gadagadars.getRoleText() +
                    ", Status: " + gadagadars.getStatusText() +
                    ", Marriage Status: " + gadagadars.getMarriageStatus() +
                    ", Top Score: " + gadagadars.getTopScoreText() +
                    ", Matches: " + gadagadars.getMatchesText());
        }
    }
}
```

Figure 26 method used for merge sort

4.5 binary search

it is used for the binary search of the element

4.5.1 method description

Method name	Descriptions
binarySearchHelper	Help to preform for the binary search
compareColumn	Used to perform for the calculate of the column

Table 6 total method used in the binary search

```
 * @author Lenovo
 */
public class binarysearch {

    /*
    this is the main method
     */

    public static void main(String[] args) {
        // Create a list of Gadagadars objects
        ArrayList<Gadagadars> dataList = createDataList();

        // Create an instance of BinarySearch
        binarysearch binarySearch = new binarysearch();
```

Figure 27 method used for the binary search

```
    // Display the result
    System.out.println("Search result for all values:");
    for (Gadagadars gadagadars : result) {
        System.out.println(gadagadars.getJerseyNo() + ", " + gadagadars.getPlayerName() + ", " + gadagad
                ", " + gadagadars.getRoleText() + ", " + gadagadars.getStatusText() + ", " +
                gadagadars.getMarriageStatus() + ", " + gadagadars.getTopScoreText() + ", " + gadagadars
    }
}



private void binarySearchHelper(ArrayList<Gadagadars> list, String searchValue, String searchColumn, int
    if (left <= right) {
        int mid = left + (right - left) / 2;
        Gadagadars midGadagadars = list.get(index: mid);

        // Compare the search value with the value in the specified column (case-insensitive)
        int comparisonResult = compareColumn(gadagadars: midGadagadars, searchValue, searchColumn);

        if (comparisonResult == 0) {
            // Element found, add it to the result list
            result.add(e: midGadagadars);

            // Recursively search left and right for more occurrences
            binarySearchHelper(list, searchValue, searchColumn, left, mid - 1, result);
            binarySearchHelper(list, searchValue, searchColumn, mid + 1, right, result);
        } else if (comparisonResult < 0) {
            binarySearchHelper(list, searchValue, searchColumn, mid + 1, right, result);
        } else {
```

Figure 28method binary search

```java
private int compareColumn(Gadagadars gadagadars, String searchValue, String searchColumn) {
    switch (searchColumn.toLowerCase()) {
        case "jerseyno":
            return gadagadars.getJerseyNo().compareTo(anotherString: searchValue);
        case "playername":
            return gadagadars.getPlayerName().compareToIgnoreCase(str:searchValue);
        case "addresstext":
            return gadagadars.getAddressText().compareToIgnoreCase(str:searchValue);
        case "roletext":
            return gadagadars.getRoleText().compareToIgnoreCase(str:searchValue);
        case "statustext":
            return gadagadars.getStatusText().compareToIgnoreCase(str:searchValue);
        case "marriagestatus":
            return gadagadars.getMarriageStatus().compareToIgnoreCase(str:searchValue);
        case "topscoretext":
            return gadagadars.getTopScoreText().compareTo(anotherString: searchValue);
        case "matchestext":
            return gadagadars.getMatchesText().compareTo(anotherString: searchValue);
        default:
            throw new IllegalArgumentException("Unknown column: " + searchColumn);
    }
}

private static ArrayList<Gadagadars> createDataList() {
    ArrayList<Gadagadars> dataList = new ArrayList<>();
    dataList.add(new Gadagadars(jerseyNo: "1", playerName: "Player1", addressText:"Address1", roleText: "Role1",
    dataList.add(new Gadagadars(jerseyNo: "2", playerName: "Player2", addressText:"Address2", roleText: "Role2",
    dataList.add(new Gadagadars(jerseyNo: "3", playerName: "Player3", addressText:"Address3", roleText: "Role3",
    // Add more data as needed
    return dataList;
}
```

Figure 29 method binary search

5 pseudocode

pseudocode is the methodology which allows the programmer for the representation of the algorithm,

the major advantage of the pseudocode are

- makes any method easier to read.
- It's among the greatest methods for beginning algorithm implementation.
- serves as a link between the algorithm or flowchart and the software.

- Additionally serves as a rough documentation, making it easy to understand a developer's software when written in pseudocode. (Bhumika_Rani, 2018)

now lets talk about the pseudocode for the binary search

class binary search

main

do

create the list of the gadagadars objects

create the instance of the binary search

call the method

display the result

use the for each loop in order to display the result

end do

do Check for termination condition:

do

If left > right, return (no more elements to search)

- Calculate midpoint:

mid = (left + right) / 2

Retrieve the object at the midpoint:

midGadagadars = list.get(mid)

Compare the search value with the specified column of the midpoint object:

comparisonResult = compareColumn(midGadagadars, searchValue, searchColumn)

If match found:

Add the midpoint object to the result list: result.add(midGadagadars)

Recursively search left and right for more occurrences:

binarySearchHelper(list, searchValue, searchColumn, left, mid - 1, result)

binarySearchHelper(list, searchValue, searchColumn, mid + 1, right, result)

If search value is less than the value at the midpoint:

Search in the left half: binarySearchHelper(list, searchValue, searchColumn, left, mid - 1, result)

If search value is greater than the value at the midpoint:

Search in the right half: binarySearchHelper(list, searchValue, searchColumn, mid + 1, right, result)

Here's the pseudocode for the provided code:

Function: compareColumn(gadagadars, searchValue, searchColumn)

Parameters:

- gadagadars: A Gadagadars object
- searchValue: The value to search for
- searchColumn: The name of the column to compare

Steps:

1. Convert searchColumn to lowercase for case-insensitive comparison.
2. Use a switch statement to compare the specified column:
    - If searchColumn is "jerseyno":
        - Return the result of comparing gadagadars.getJerseyNo() with searchValue.
    - If searchColumn is "playername":
        - Return the result of comparing gadagadars.getPlayerName() (case-insensitive) with searchValue.
    - Repeat for other supported columns (addresstext, roletext, statustext, marriagestatus, topscoretext, matchestext).
3. If no match found in the switch statement, throw an IllegalArgumentException indicating an unknown column.

Function: createDataList()

Steps:

1. Create an empty ArrayList of Gudivada’s objects.
2. Add three hardcoded Gadagadars objects to the list with sample data:

- ("1", "Player1", "Address1", "Role1", "Status1", "MarriageStatus1", "100", "200")
- ("2", "Player2", "Address2", "Role2", "Status2", "MarriageStatus2", "150", "250")
- ("3", "Player3", "Address3", "Role3", "Status3", "MarriageStatus3", "120", "220")

3. Return the created list.

6 test cases

6.1 test cases for the add player.

There are different test cases has been added in the cases in the add player

Table 7 table for testing of player

Test cases	descriptions
Purpose	The purpose of this test is that it is used to show the value that has been added or not.
Valued used	Jersey no 9 Name samman Address Kathmandu Role batsman Status playing Married status married Top score 400 Total matches 300
Problem encountered	No problem encountered
Status	Test was sucessful

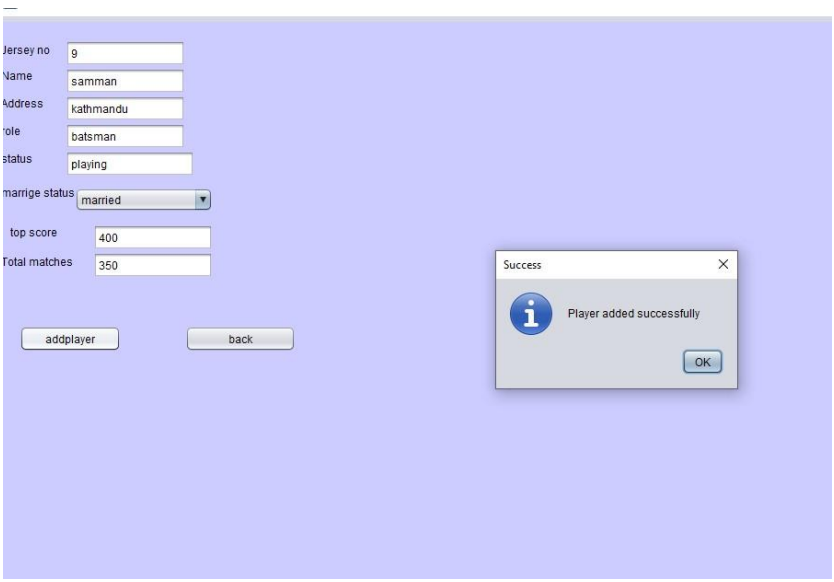

Figure 30 figure for the add of player

Method name	Descriptions
Purpose	Purpose whether the value to check for the validation of the jersey no
Valued used	Jersey no 16 Name samman Address Kathmandu Role bolower Marriage status unmarried Top score 400 Total matches 350
Problem faced	Here I have used the exceptions that the jersey number should between the 1 and 15 got the problem
Solutions	Change the jersey number from 15 to 14
Test	Test was successful

Table 8 table for the exception handeling of the add player

Figure 31 figure showing the error message

now I have changed the jersey number to 15 from and 16 and then the given message was successful as per the validations

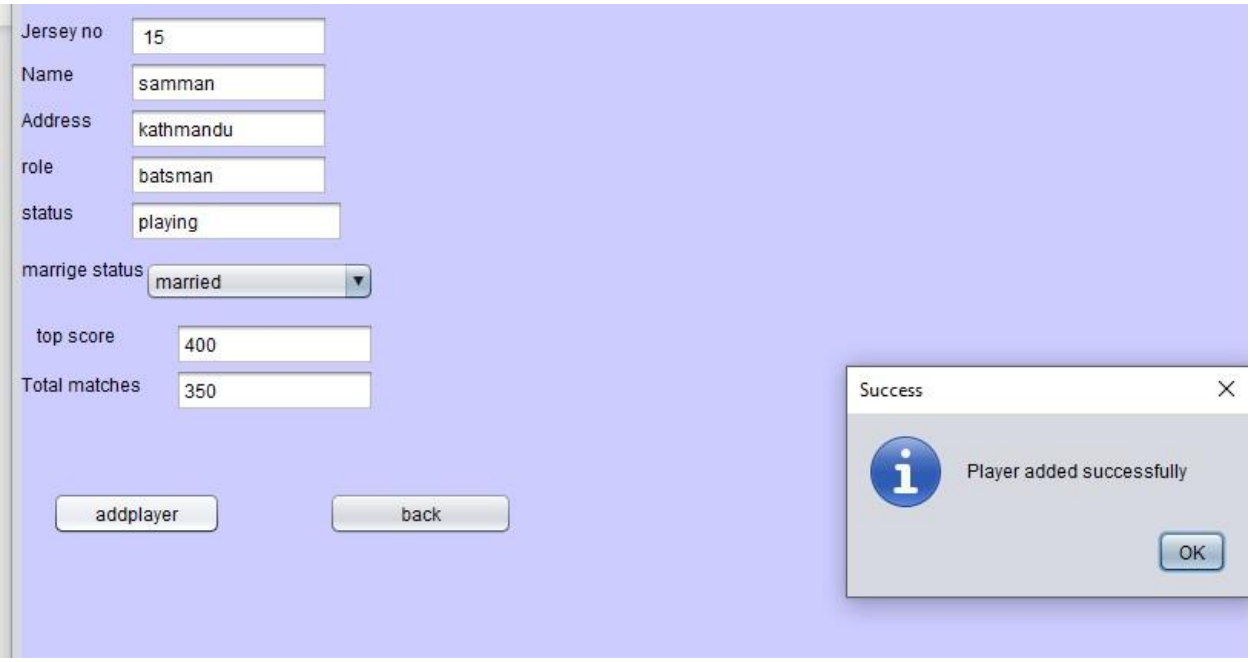

Figure 32 figure showing after the exceptions handeling

Method	Description
purpose	The purpose of this code is that it used to check

	for the one field are empty or not
Value used	Jersey number 15 Name Address Kathmandu Role batsman Status playing Marriage status married Top score 400 Total matches 450
Problem faced	The major problem we encountered that exceptions is thrown because of the fact that it has blank field in it
Solutions	No issue after the issue

Table 9 method used for add data

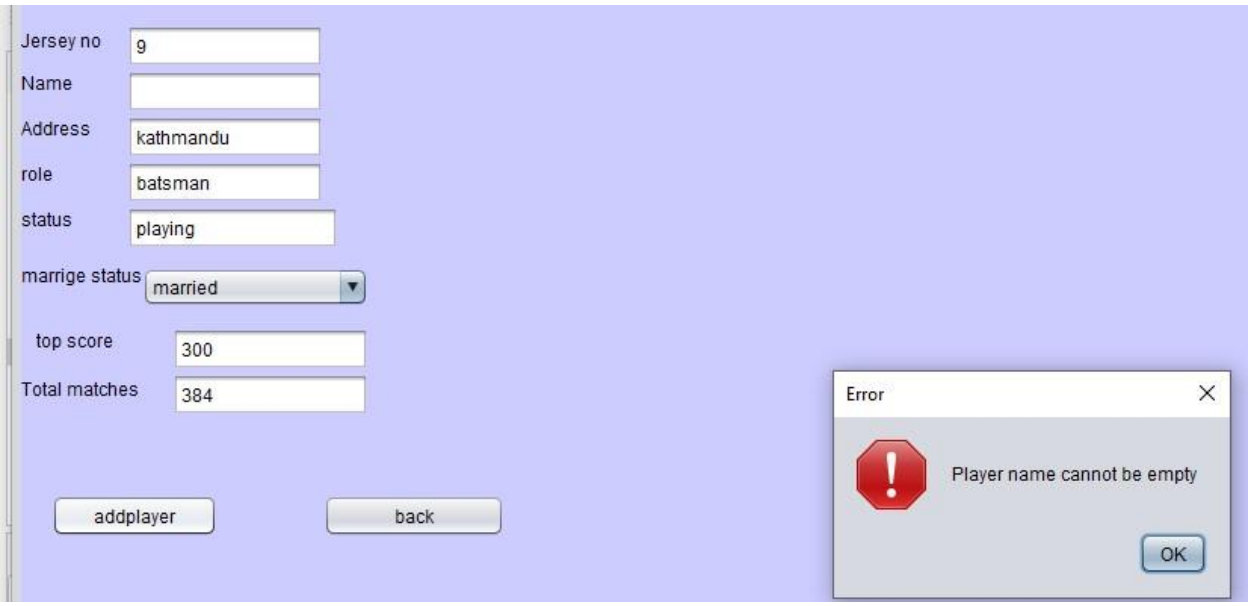

Figure 33 for the add player

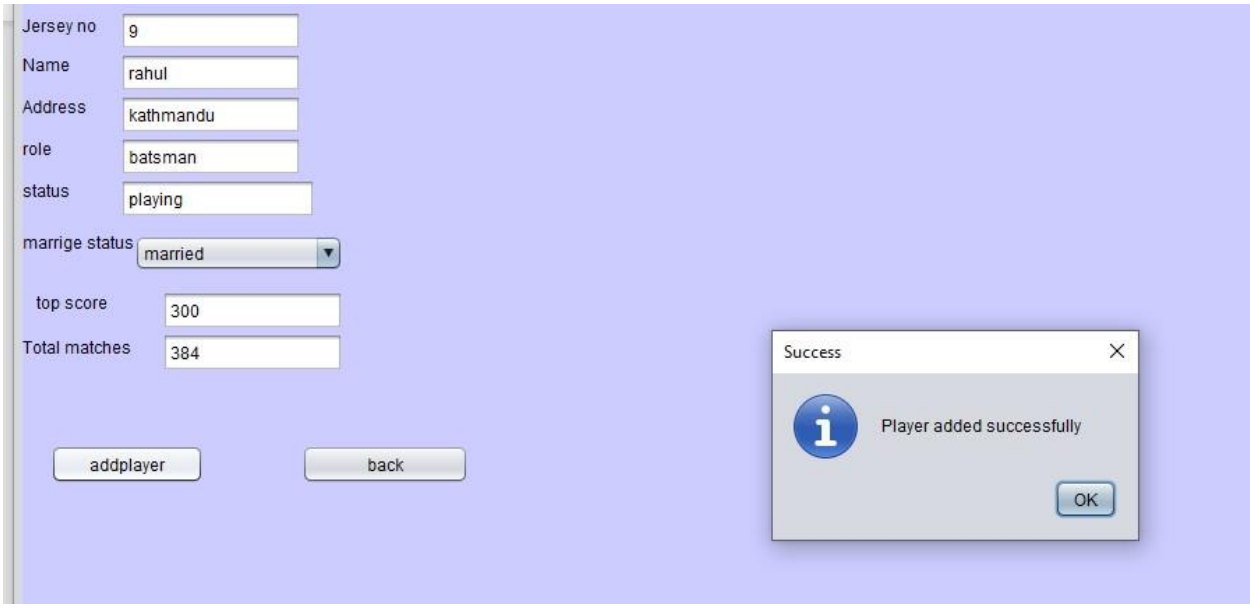

Figure 34 for addplayer

6.2 update player

Method	descriptions
purpose	Test for the value of the table
value	Jersey no 13 If the method it should it should be in the j panel
Problem faced	No issue faced
Problem encountered	It has no issue

Table 10 for the update player

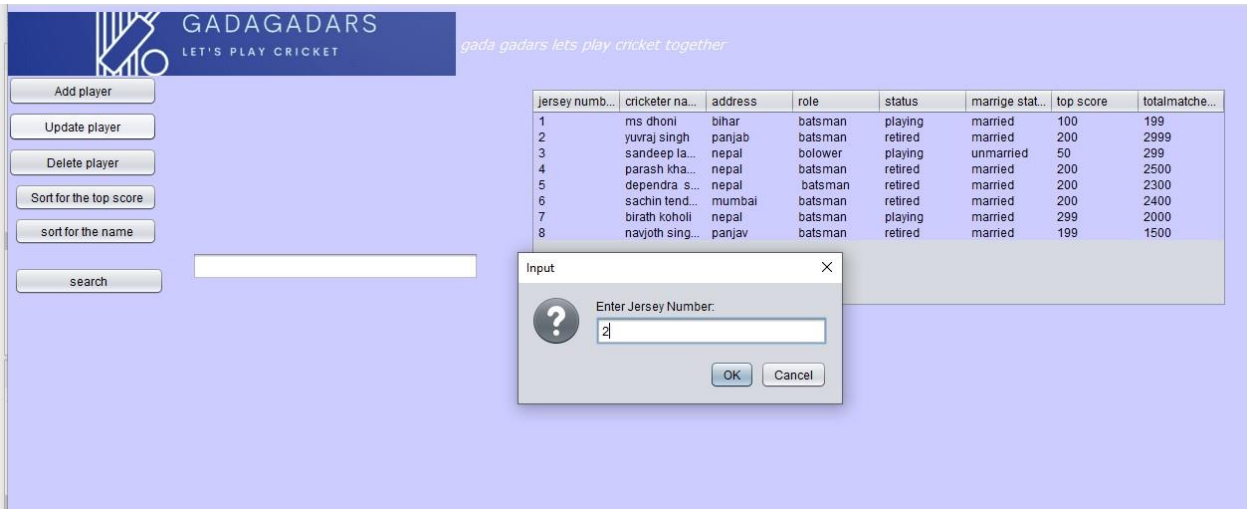

Figure 35 figure for the testing

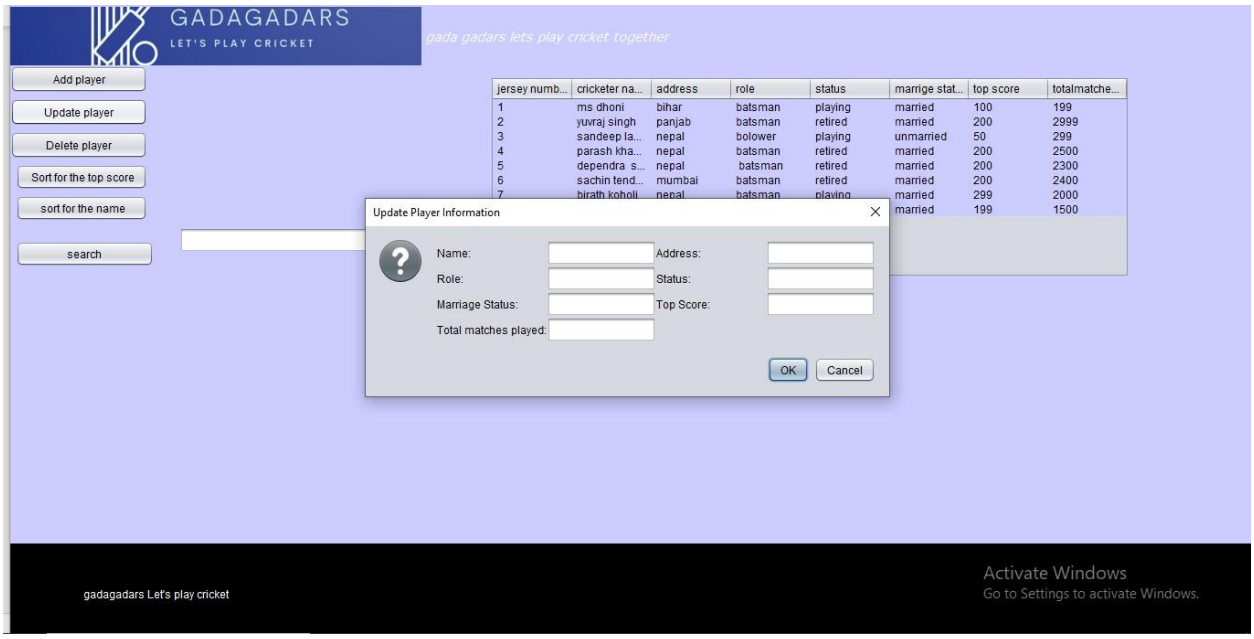

Figure 36 figure for update

Method	descriptions
purpose	It used to for the descriptions for the exceptions handling for the update
Value used	Used for the jersey number matches or not
Hurdle faced	For the exceptions handling for 17
solutions	I have the change the value from 17 to1

Table 11 for the method in the table in the value

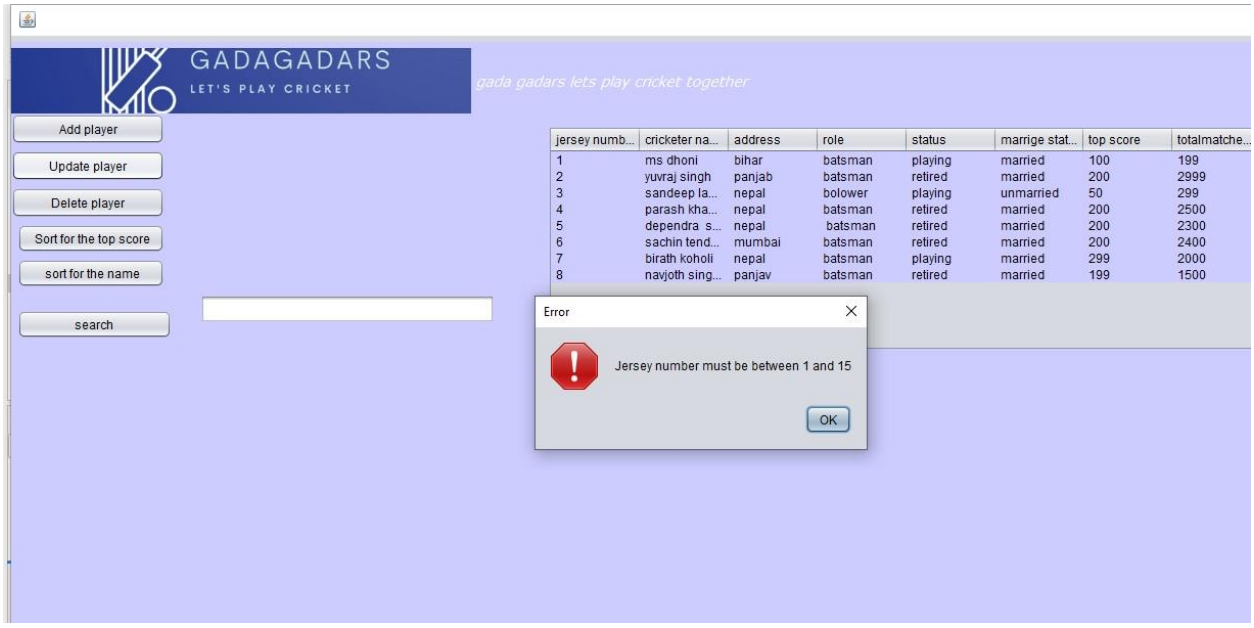

Figure 37 exceptions handing

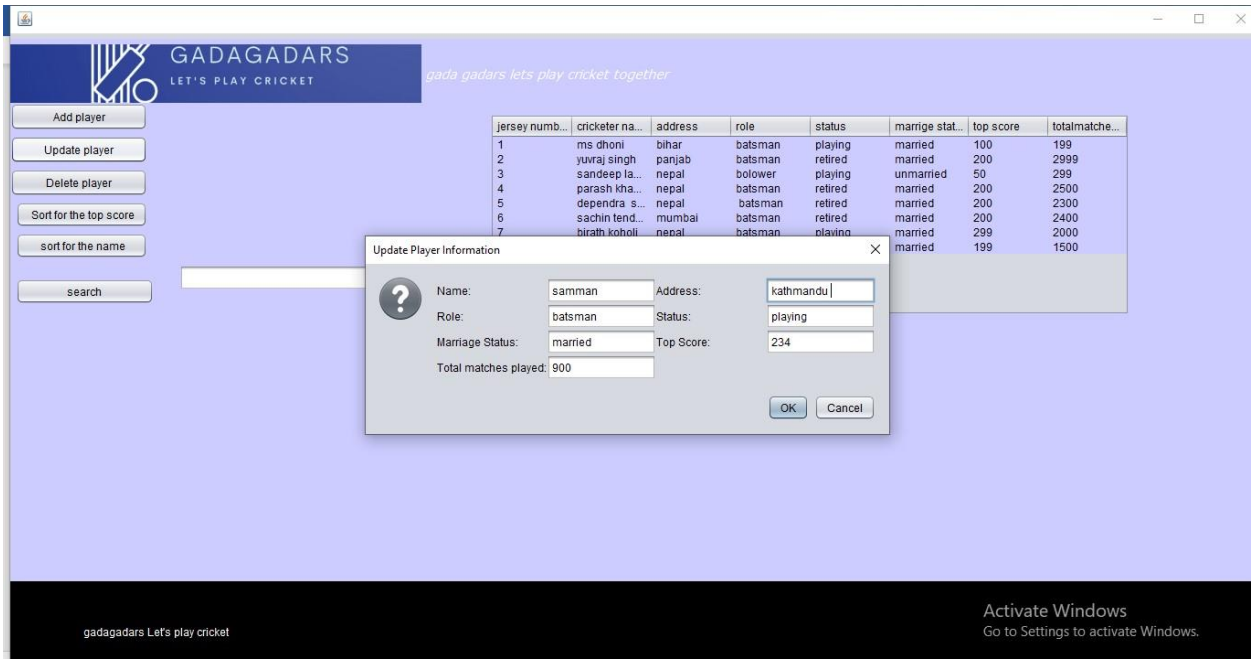

Figure 38 after the exception handling

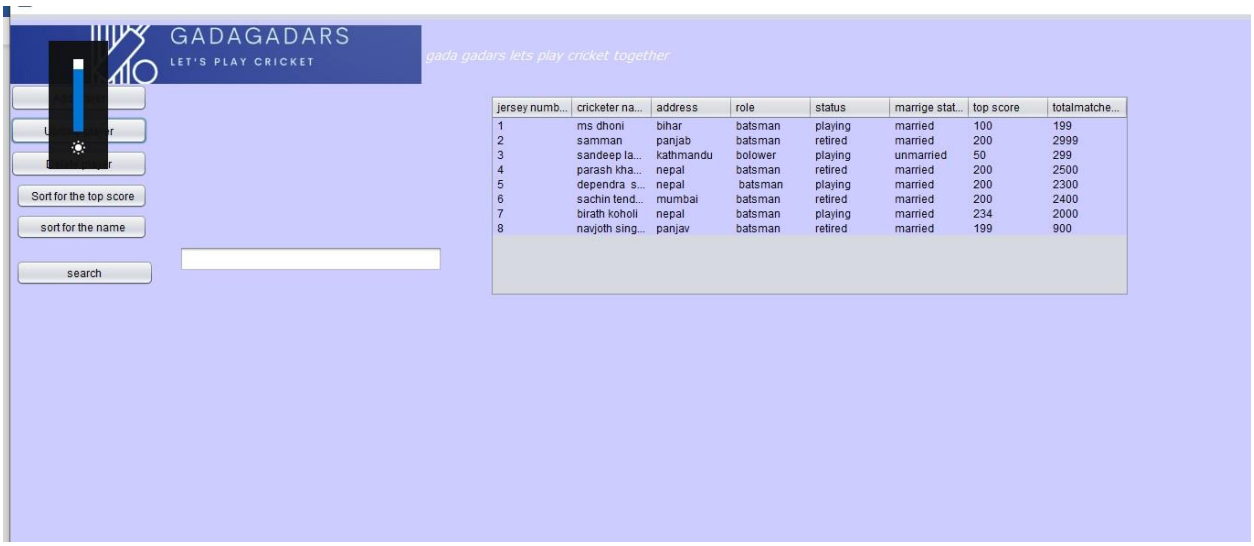

Figure 39 sucessful message after the validations

6.3 delete of player

Actions	Descriptions
For the delete of the player	After seleting the row the value will be deleted

For the exception	To check whether the row has been selected or not

Figure 40 for the update of the value

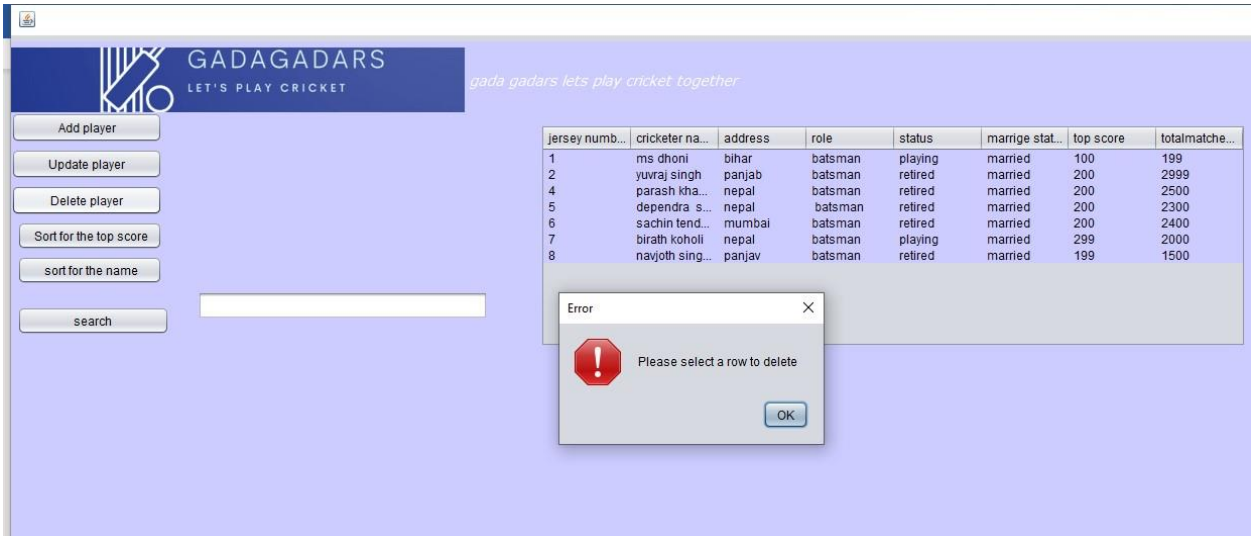

Figure 41 for the row to delete

7 development

7.1tools and technique implemented

there are different tools and technique that I used in order to complete the courser

one of the major tools that we used are

- apache netbeans

it is ide for the java in order to complete the coursework

Apache NetBeans is a free and open-source Integrated Development Environment (IDE) primarily focused on Java development. It's a powerful and versatile tool used by developers of all skill levels for creating various software applications.

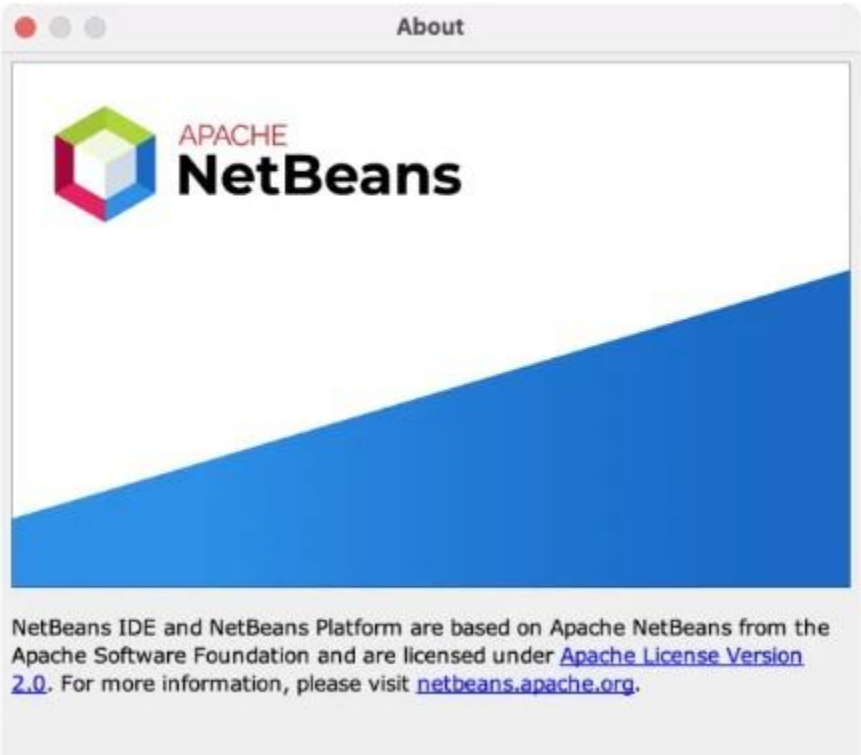

Figure 42 apache netbeans feature

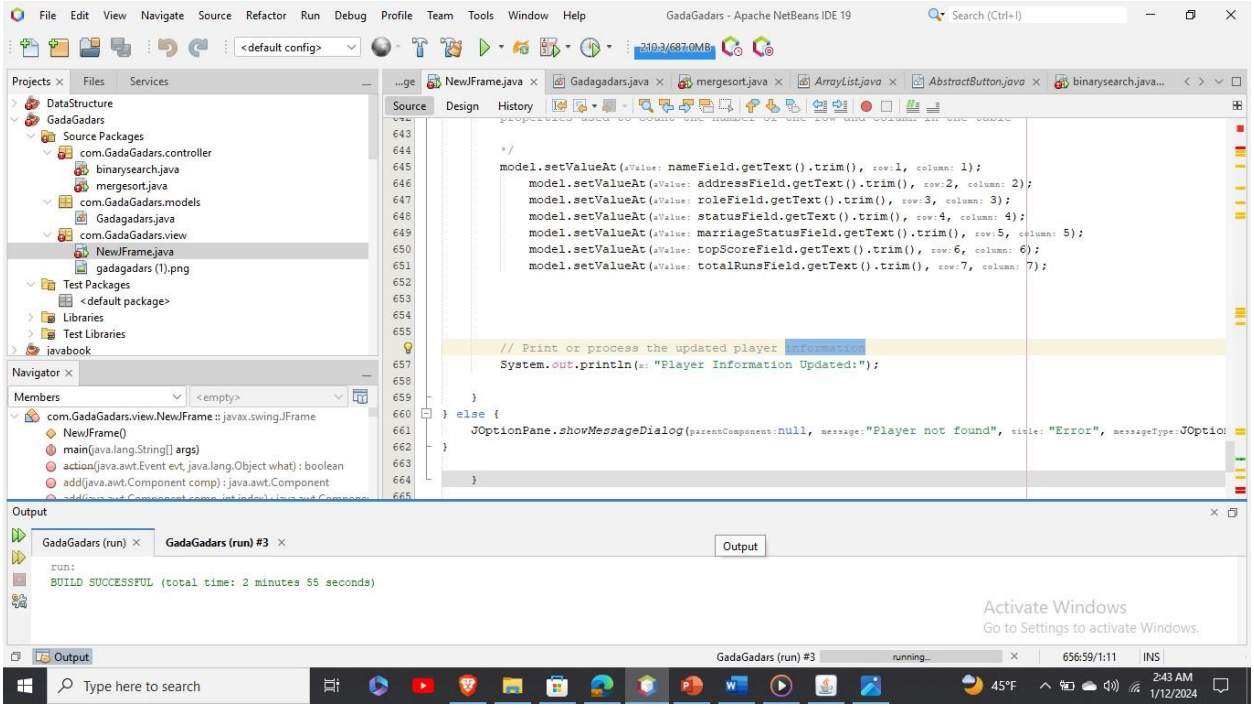

Figure 43 apache netbeans window

- ms words
  it's a popular word processing program developed by Microsoft and included in the Microsoft Office suite. the feature of the ms word are

- Creating and editing documents: From simple text documents to complex reports and presentations, Word offers a wide range of features for creating and formatting content.
- Collaboration: Multiple users can work on the same document simultaneously, making it ideal for team projects.
- Templates and layouts: A vast library of pre-designed templates and layouts helps you create professional-looking documents quickly and easily.
- Spell checking and grammar tools: Built-in spell checking, and grammar tools help you identify and correct errors in your writing.

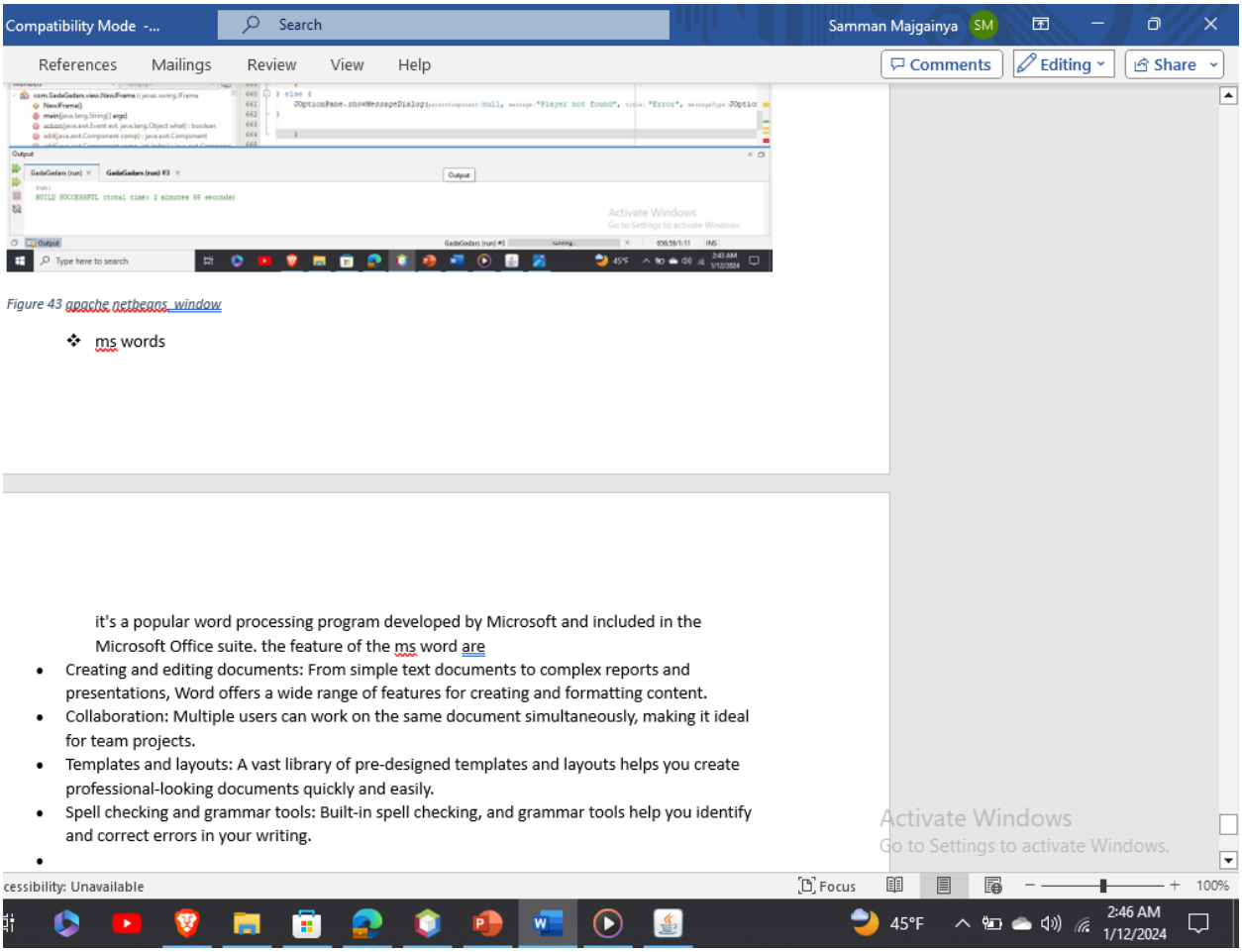

Figure 44 ms word internal system

- ❖ lucid chart

it is used for the creating of the different diagrams like class diagrams flowchart and so on

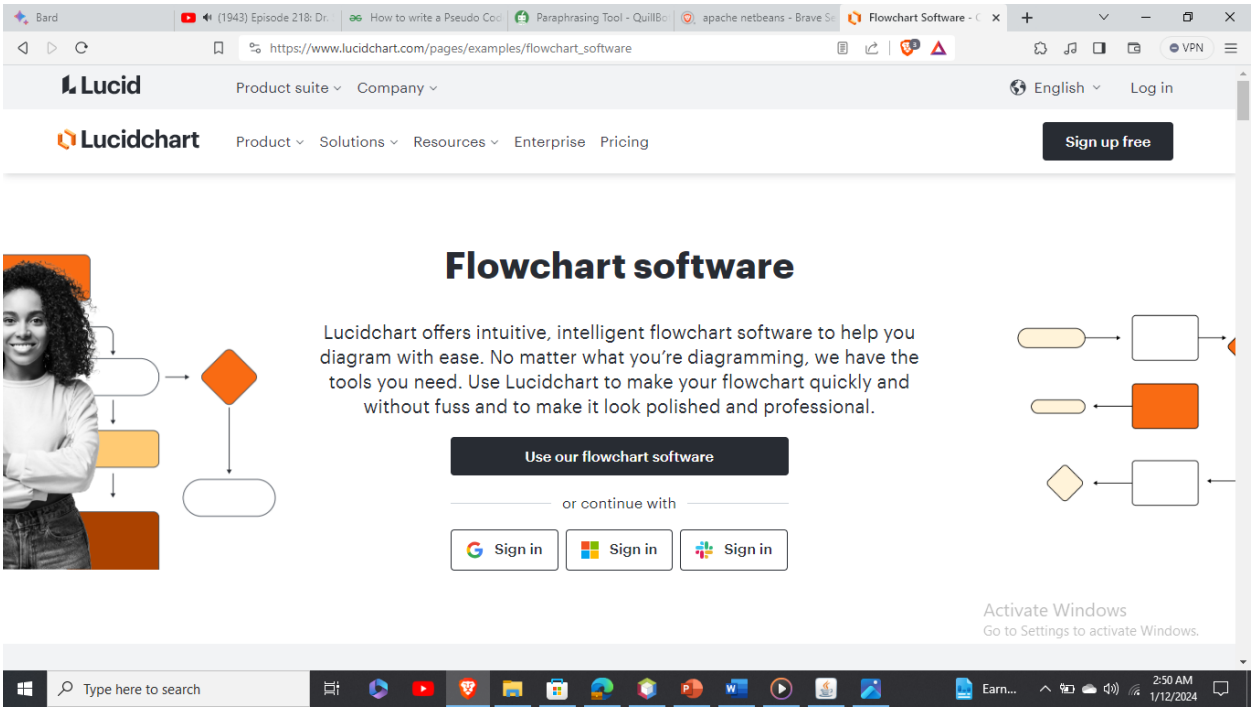

Figure 45 lucid chart

❖ stack overflow

I used for the research all the thing related to coursework.

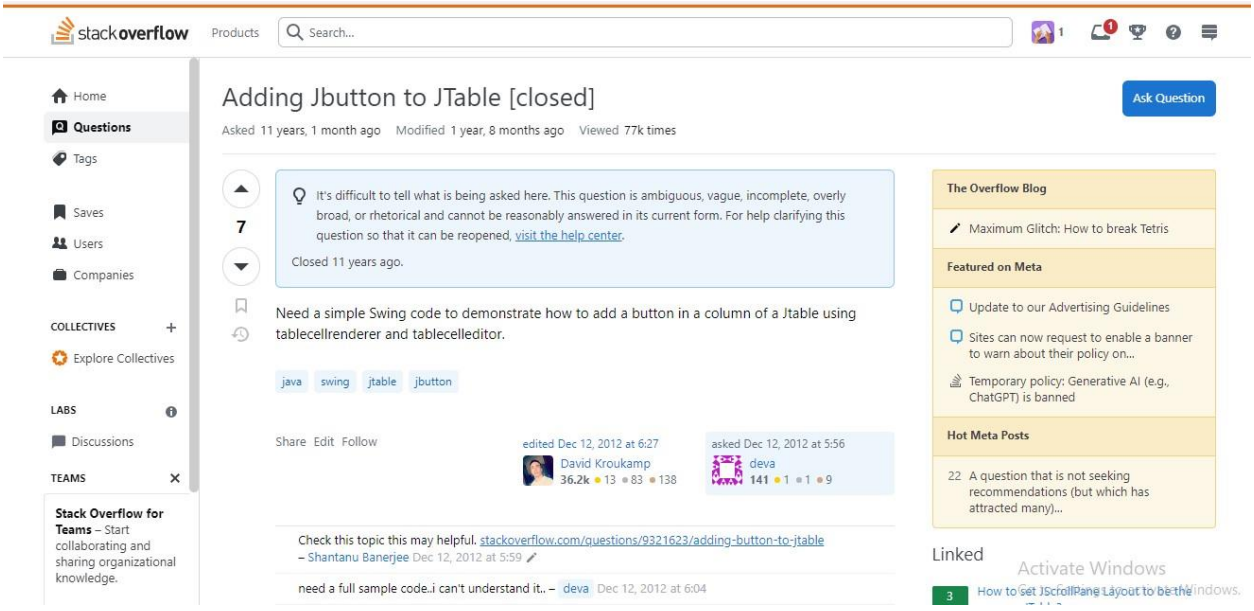

Figure 46 for the stack overflow

8 technique used

several techniques has been used in order to complete the coursework they have been mentioned here

- ❖ binary search

A searching method known as binary search halves the search period in half repeatedly when applied to a sorted array. Using the information that has been sorted into an array, binary search aims to minimize the time complexity to O(log N). (greek, 2024)

- ❖ recursion

Recursion is the process by which a function makes direct or indirect calls to itself; the equivalent function is known as a recursive function. A recursive method can be used to solve some problems quite quickly.((tuneja, 2023)

- ❖ data structure and algorithms

Several data striucture and algorithm concept has been used like merge sort and so on

9 challenges

several challenges has been identified in the coursework some of them are

- ❖ hard to know about the right data structure that has been used in the coursework
- ❖ hard for the know about the right knowledge.
- ❖ faced a lot of issues in the code and
- ❖ hard in the designing portions

10 conclusion

At last I would like to thanks the mr Prithivi maharjan sir in order to help me complete the coursework as well. I faced a lot of the issue while completions of the coursework as the system the outlined requirements and tasks for the system development project encompass a comprehensive approach to building a user-friendly and efficient address management system. The primary goals include providing a well-designed user interface, enabling data creation and deletion functionalities, and implementing robust data structures and algorithms for efficient data management.

Furthermore, the incorporation of a Binary Search algorithm for address information search adds an efficient and targeted search functionality, allowing users to refine their searches based on account type and status. The introduction of sorting functionality based on account balance contributes to the efficient organization of data, providing users with options for ascending, descending, or both orders.

Bibliography

Bhumika_Rani. (2018, aug 30). *greek for greek* . Retrieved from greek for greek : https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/

greek, g. f. (2024, jan 10). *binary search* . Retrieved from binary search: https://www.geeksforgeeks.org/binary-search/

programiz. (n.d.). *programiz*. Retrieved from https://www.programiz.com/dsa/algorithm

RishabhPrabhu. (n.d.). *greek to greek* . Retrieved from greek to greek: https://www.geeksforgeeks.org/introduction-to-algorithms/

tuneja, s. (2023, march 31). *greek for greek* . Retrieved from greek for greek: https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/

w3schools. (n.d.). *method* . Retrieved from w3schools.

code of appendix

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.GadaGadars.controller;

import com.GadaGadars.models.Gadagadars;
import java.util.ArrayList;



/**
 * this is used to preform the binary search in the method
 * @author Lenovo
 */
public class binarysearch {




    public static void main(String[] args) {
        // Create a list of Gadagadars objects
        ArrayList<Gadagadars> dataList = createDataList();

        // Create an instance of BinarySearch
        binarysearch binarySearch = new binarysearch();

        // Call the example method
```

```
        ArrayList<Gadagadars> result = binarySearch.binarySearchForAll(dataList);


        // Display the result
        System.out.println("Search result for all values:");
        for (Gadagadars gadagadars : result) {
            System.out.println(gadagadars.getJerseyNo() + ", " + gadagadars.getPlayerName() + ",
" + gadagadars.getAddressText() +
                    ", " + gadagadars.getRoleText() + ", " + gadagadars.getStatusText() + ", " +
                    gadagadars.getMarriageStatus() + ", " + gadagadars.getTopScoreText() + ", " +
gadagadars.getMatchesText());
        }
    }




    private void binarySearchHelper(ArrayList<Gadagadars> list, String searchValue, String
searchColumn, int left, int right, ArrayList<Gadagadars> result) {
        if (left <= right) {
            int mid = left + (right - left) / 2;
            Gadagadars midGadagadars = list.get(mid);


            // Compare the search value with the value in the specified column (case-insensitive)
            int comparisonResult = compareColumn(midGadagadars, searchValue, searchColumn);


            if (comparisonResult == 0) {
                // Element found, add it to the result list
                result.add(midGadagadars);


                // Recursively search left and right for more occurrences
                binarySearchHelper(list, searchValue, searchColumn, left, mid - 1, result);
```

```
            binarySearchHelper(list, searchValue, searchColumn, mid + 1, right, result);
        } else if (comparisonResult < 0) {
            binarySearchHelper(list, searchValue, searchColumn, mid + 1, right, result);
        } else {
            binarySearchHelper(list, searchValue, searchColumn, left, mid - 1, result);
        }
    }
}


private int compareColumn(Gadagadars gadagadars, String searchValue, String
searchColumn) {
    switch (searchColumn.toLowerCase()) {
        case "jerseyno":
            return gadagadars.getJerseyNo().compareTo(searchValue);
        case "playername":
            return gadagadars.getPlayerName().compareToIgnoreCase(searchValue);
        case "addresstext":
            return gadagadars.getAddressText().compareToIgnoreCase(searchValue);
        case "roletext":
            return gadagadars.getRoleText().compareToIgnoreCase(searchValue);
        case "statustext":
            return gadagadars.getStatusText().compareToIgnoreCase(searchValue);
        case "marriagestatus":
            return gadagadars.getMarriageStatus().compareToIgnoreCase(searchValue);
        case "topscoretext":
            return gadagadars.getTopScoreText().compareTo(searchValue);
        case "matchestext":
            return gadagadars.getMatchesText().compareTo(searchValue);
        default:
```

```
            throw new IllegalArgumentException("Unknown column: " + searchColumn);
        }
    }

    private static ArrayList<Gadagadars> createDataList() {
        ArrayList<Gadagadars> dataList = new ArrayList<>();
        dataList.add(new Gadagadars("1", "Player1", "Address1", "Role1", "Status1",
"MarriageStatus1", "100", "200"));
        dataList.add(new Gadagadars("2", "Player2", "Address2", "Role2", "Status2",
"MarriageStatus2", "150", "250"));
        dataList.add(new Gadagadars("3", "Player3", "Address3", "Role3", "Status3",
"MarriageStatus3", "120", "220"));
        // Add more data as needed
        return dataList;
    }
}
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.GadaGadars.controller;



import com.GadaGadars.models.Gadagadars;
import com.GadaGadars.models.Gadagadars.Gadagadars;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import javax.swing.JTable;
```

```
import javax.swing.table.DefaultTableModel;

//import java.util.Comparator;

//import java.util.List;

//import javax.swing.JTable;

//import javax.swing.table.DefaultTableModel;


/**

 *

 *  this is the controller class used for the merge sort

 */

public class mergesort {


/*

      it used for the return of the array list

    @param used list, string property , comparater and comparator

    */


   public <T> ArrayList<T> mergeSort(ArrayList<T> list, String property, Comparator<T>
comparator) {

        if (list.size() <= 1) {

            return list;

        }

        int mid = list.size() / 2;// used  for the mid point

        /*

         used to create the array list from index 0 to kid


        */

        ArrayList<T> left = new ArrayList<>(list.subList(0, mid));

       /*
```

```
 used to create the array from index from mid to list size
*/
ArrayList<T> right = new ArrayList<>(list.subList(mid, list.size()));
/*
 used for the left
@ paraam used left , property and comparator
for the right
@ param used right property and comparator
reutrn the  merge
@ param used left right property and comparator
*/
left = mergeSort(left, property, comparator);
right = mergeSort(right, property, comparator);
return merge(left, right, property, comparator);
}

private <T> ArrayList<T> merge(ArrayList<T> left, ArrayList<T> right, String property,
Comparator<T> comparator) {
ArrayList<T> result = new ArrayList<>();
int leftIndex = 0;
int rightIndex = 0;
// test for the index of the element

while (leftIndex < left.size() && rightIndex < right.size()) {
    if (comparator.compare(left.get(leftIndex), right.get(rightIndex)) < 0) {
        result.add(left.get(leftIndex));
        leftIndex++;
    } else {
        result.add(right.get(rightIndex));
```

```
            rightIndex++;
        }
    }
    //add all the element

    result.addAll(left.subList(leftIndex, left.size()));
    result.addAll(right.subList(rightIndex, right.size()));

    return result;
  }
  public static void main(String[] args) {
    // Create an instance of the mergesort class
    mergesort mergeSort = new mergesort();

    // Example: Create a list of Gadagadars
    ArrayList<Gadagadars> gadagadarsList = new ArrayList<>();
    gadagadarsList.add(new Gadagadars("1", "John", "Address1", "Role1", "Status1",
"Married", "100", "50"));
    gadagadarsList.add(new Gadagadars("2", "Jane", "Address2", "Role2", "Status2", "Single",
"90", "60"));

    // Define a comparator for sorting by player name
    Comparator<Gadagadars> playerNameComparator =
Comparator.comparing(Gadagadars::getPlayerName);

    // Perform the merge sort
    ArrayList<Gadagadars> sortedList = mergeSort.mergeSort(gadagadarsList, "playerName",
playerNameComparator);

    // Display the sorted list
```

```
        System.out.println("Sorted List:");
        for (Gadagadars gadagadars : sortedList) {
            System.out.println("Jersey No: " + gadagadars.getJerseyNo() +
                    ", Player Name: " + gadagadars.getPlayerName() +
                    ", Address: " + gadagadars.getAddressText() +
                    ", Role: " + gadagadars.getRoleText() +
                    ", Status: " + gadagadars.getStatusText() +
                    ", Marriage Status: " + gadagadars.getMarriageStatus() +
                    ", Top Score: " + gadagadars.getTopScoreText() +
                    ", Matches: " + gadagadars.getMatchesText());
        }
    }
}
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.GadaGadars.models;


/**
 *
 * @author Lenovo
 */
public class Gadagadars {
   String jerseyNo;
   String playerName;
    String addressText;
```

```
    public String getJerseyNo() {
        return jerseyNo;
    }

    public String getPlayerName() {
        return playerName;
    }

    public String getAddressText() {
        return addressText;
    }

    public String getRoleText() {
        return roleText;
    }

    public String getStatusText() {
        return StatusText;
    }

    public String getMarriageStatus() {
        return marriageStatus;
    }

    public String getTopScoreText() {
        return topScoreText;
    }
```

```
    public String getMatchesText() {
        return matchesText;
    }

    String roleText;
    String StatusText;
    String marriageStatus;

    public Gadagadars(String jerseyNo, String playerName, String addressText, String roleText,
String StatusText, String marriageStatus, String topScoreText, String matchesText) {
        this.jerseyNo = jerseyNo;
        this.playerName = playerName;
        this.addressText = addressText;
        this.roleText = roleText;
        this.StatusText = StatusText;
        this.marriageStatus = marriageStatus;
        this.topScoreText = topScoreText;
        this.matchesText = matchesText;
    }
     String topScoreText;
     String matchesText;

    public void setJerseyNo(String jerseyNo) {
        this.jerseyNo = jerseyNo;
    }

    public void setPlayerName(String playerName) {
        this.playerName = playerName;
    }
```

```
    public void setAddressText(String addressText) {
        this.addressText = addressText;
    }

    public void setRoleText(String roleText) {
        this.roleText = roleText;
    }

    public void setStatusText(String StatusText) {
        this.StatusText = StatusText;
    }

    public void setMarriageStatus(String marriageStatus) {
        this.marriageStatus = marriageStatus;
    }



    public void setTopScoreText(String topScoreText) {
        this.topScoreText = topScoreText;
    }

    public void setMatchesText(String matchesText) {
        this.matchesText = matchesText;
    }

    public Object getName() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
```

```
    }


        public Object gettopScore() {

            throw new UnsupportedOperationException("Not supported yet."); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody

        }




}


    /*

         it is used to get text  for the value recieved by the  user for the data recived

        parameter  used jersey no, player name , address , role status  marrigestatus top score ,
matches

         method used get text and trim

        */

     String jerseyNoText = jerseyno1.getText().trim();

String playerName = name.getText().trim();

String addressText = address.getText().trim();

String roleText = role.getText().trim();

String statusText = status.getText().trim();


String marriageStatus = (String) marriagestatus.getSelectedItem();

String topScoreText = topscore.getText().trim();

String matchesText = totalmatches.getText().trim();

DefaultTableModel tableModel = (DefaultTableModel) addplayertable1.getModel();// used to
updtate the value in the table


try {

    if (!jerseyNoText.isEmpty()) {
```

```
        int jerseyNo = Integer.parseInt(jerseyNoText);
        if (jerseyNo < 1 || jerseyNo > 15) {
            throw new IllegalArgumentException("Jersey number must be between 1 and 15");
        }// check for the jersey number is empty or not ot it should be in between the value of 1 and
15


        // Check if the jersey number is already in the table
        for (int row = 0; row < tableModel.getRowCount(); row++) {
            if (tableModel.getValueAt(row, 0).equals(jerseyNoText)) {
                throw new IllegalArgumentException("Jersey number must be unique");
            }
        }
    } else {
        throw new IllegalArgumentException("Jersey number cannot be empty");// display the
message
    }


    if (playerName.isEmpty()) {
        throw new IllegalArgumentException("Player name cannot be empty");// check for the
player name is empty or not
    }


    if (addressText.isEmpty()) {
        throw new IllegalArgumentException("Address cannot be empty"); // check for the address
should be empty or not
    }


    if (roleText.isEmpty()) {
        throw new IllegalArgumentException("Role cannot be empty");// check for thee  role is
empty
    }
```

```
    if (statusText.isEmpty()) {
        throw new IllegalArgumentException("Status cannot be empty");// check for the status is empty or not
    }

    // Additional checks for alphanumeric characters in playerName, addressText, roleText, and statusText
    if (!playerName.matches("[a-zA-Z ]+")) {
        throw new IllegalArgumentException("Player name should only contain alphabets and spaces");
    }

    if (!addressText.matches("[a-zA-Z0-9 ]+")) {
        throw new IllegalArgumentException("Address should only contain alphanumeric characters and spaces");
    }

    if (!roleText.matches("[a-zA-Z ]+")) {
        throw new IllegalArgumentException("Role should only contain alphabets and spaces");
    }

    if (!statusText.matches("[a-zA-Z ]+")) {
        throw new IllegalArgumentException("Status should only contain alphabets and spaces");
    }
// check for the top score should be in between the 300 and 800
    if (!topScoreText.isEmpty()) {
        int topScore = Integer.parseInt(topScoreText);
        if (topScore < 300 || topScore > 800) {
            throw new IllegalArgumentException("Top score must be between 300 and 800");
```

```
        }
    } else {
        throw new IllegalArgumentException("Top score cannot be empty");
    }
// checks for matche for the betweem the 300 and 500
    if (!matchesText.isEmpty()) {
        int totalMatches = Integer.parseInt(matchesText);
        if (totalMatches < 300 || totalMatches > 500) {
            throw new IllegalArgumentException("Total matches must be between 300 and 500");
        }
    } else {
        throw new IllegalArgumentException("Total matches cannot be empty");
    }
  /*
     create an new object for delete of table
    @param usedjerseyNoText, playerName, addressText, roleText, statusText, marriageStatus,
topScoreText, matchesText};
    tableModel.addRow(rowData
    */
    Object[] rowData = {jerseyNoText, playerName, addressText, roleText, statusText,
marriageStatus, topScoreText, matchesText};
    tableModel.addRow(rowData);


    // Display success message
    JOptionPane.showMessageDialog(null, "Player added successfully", "Success",
JOptionPane.INFORMATION_MESSAGE);


} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(null, "Invalid input for jersey number, top score, or total
matches", "Error", JOptionPane.ERROR_MESSAGE);
```

```
    e.printStackTrace(); // Handle or log the exception as needed
} catch (IllegalArgumentException e) {
    JOptionPane.showMessageDialog(null, e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    e.printStackTrace(); // Handle or log the exception as needed
} catch (Exception e) {
    JOptionPane.showMessageDialog(null, "An unexpected error occurred", "Error",
JOptionPane.ERROR_MESSAGE);
    e.printStackTrace(); // Handle or log the exception as needed
}




        }


    private void updateplayerActionPerformed(java.awt.event.ActionEvent evt) {


      String jerseyNoText = JOptionPane.showInputDialog("Enter Jersey Number:").trim();


// Check if jersey number is null or empty
if (jerseyNoText == null || jerseyNoText.isEmpty()) {
    JOptionPane.showMessageDialog(null, "Jersey number cannot be empty", "Error",
JOptionPane.ERROR_MESSAGE);
    return; // Exit the method if there's an error
}


// Validate the entered jersey number for alphanumeric characters
if (!jerseyNoText.matches("\\d+")) {
    JOptionPane.showMessageDialog(null, "Invalid Jersey number format (only digits are
allowed)", "Error", JOptionPane.ERROR_MESSAGE);
    return; // Exit the method if there's an error
```

```
}

// Parse the jersey number to an integer
int jerseyNo;
try {
    jerseyNo = Integer.parseInt(jerseyNoText);

    // Validate the range of the jersey number
    if (jerseyNo < 1 || jerseyNo > 15) {
        JOptionPane.showMessageDialog(null, "Jersey number must be between 1 and 15",
"Error", JOptionPane.ERROR_MESSAGE);
        return; // Exit the method if there's an error
    }
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(null, "Invalid Jersey number format", "Error",
JOptionPane.ERROR_MESSAGE);
    return; // Exit the method if there's an error
}

// Get the table model and row count
DefaultTableModel model = (DefaultTableModel) addplayertable1.getModel();
int rowCount = model.getRowCount();

// Search for the player with the specified jersey number
Object[] playerToUpdate = null;
for (int i = 0; i < rowCount; i++) {
    int currentJerseyNo = Integer.parseInt(model.getValueAt(i, 0).toString());

    if (currentJerseyNo == jerseyNo) {
        // Create a new PlayerData instance with existing data
```

```
        playerToUpdate = model.getDataVector().elementAt(i).toArray();
        break; // Exit the loop once the player is found
    }
}



if (playerToUpdate == null)
{
    JOptionPane.showMessageDialog(null, "Player with Jersey number " + jerseyNo + " does not exist in the table", "Error", JOptionPane.ERROR_MESSAGE);
    return; // Exit the method if the player does not exist
}


for (int i = 0; i < rowCount; i++) {
    int currentJerseyNo = Integer.parseInt(model.getValueAt(i, 0).toString());


    if (currentJerseyNo == jerseyNo) {


        playerToUpdate = model.getDataVector().elementAt(i).toArray();


        break;
    }
}
// check for the null value
if (playerToUpdate != null) {
    // Create the update panel
    JPanel updatePanel = new JPanel();
    updatePanel.setLayout(new GridLayout(4, 4));
/*
```

```
 used to get the textfiled
 @param used name, address, role , status, marrige status, topscore and total runs
 */
 JTextField nameField = new JTextField(8);
 JTextField addressField = new JTextField(8);
 JTextField roleField = new JTextField(8);
 JTextField statusField = new JTextField(8);
 JTextField marriageStatusField = new JTextField(8);
 JTextField topScoreField = new JTextField(8);
 JTextField totalRunsField = new JTextField(8);

/*
 used to create the jlabel
 @param used name, address, role status, marrige staatus , topscore, total matches played
 */
 updatePanel.add(new JLabel("Name:"));
 updatePanel.add(nameField);
 updatePanel.add(new JLabel("Address:"));
 updatePanel.add(addressField);
 updatePanel.add(new JLabel("Role:"));
 updatePanel.add(roleField);
 updatePanel.add(new JLabel("Status:"));
 updatePanel.add(statusField);
 updatePanel.add(new JLabel("Marriage Status:"));
 updatePanel.add(marriageStatusField);
 updatePanel.add(new JLabel("Top Score:"));
 updatePanel.add(topScoreField);
 updatePanel.add(new JLabel("Total matches played:"));
 updatePanel.add(totalRunsField);
```

```
    // Display the JOptionPane for updating player information
    int result = JOptionPane.showConfirmDialog(null, updatePanel, "Update Player Information",
            JOptionPane.OK_CANCEL_OPTION);
/*
     if matches the  result should have the   updated named and contains all the  value
    @param used upated name , updated address, updated role, updated status, updaated
marriage status  updated topscore
    total runs
    method used get text  and trim

    */
    if (result == JOptionPane.OK_OPTION) {

        String updatedName = nameField.getText().trim();
        String updatedAddress = addressField.getText().trim();
        String updatedRole = roleField.getText().trim();
        String updatedStatus = statusField.getText().trim();
        String updatedMarriageStatus = marriageStatusField.getText().trim();
        String updatedTopScore = topScoreField.getText().trim();
        String updatedTotalRuns = totalRunsField.getText().trim();

        /*
```

```
 used  to set the value based on the table
@ param useed name , address. role ,status , marriage status , top score and total runs


 method used set value  and get text
properties used to count the number of the row and column in the table


*/
model.setValueAt(nameField.getText().trim(), 1, 1);
    model.setValueAt(addressField.getText().trim(), 2, 2);
    model.setValueAt(roleField.getText().trim(), 3, 3);
    model.setValueAt(statusField.getText().trim(), 4, 4);
    model.setValueAt(marriageStatusField.getText().trim(), 5, 5);
    model.setValueAt(topScoreField.getText().trim(), 6, 6);
    model.setValueAt(totalRunsField.getText().trim(), 7, 7);





// Print or process the updated player information
System.out.println("Player Information Updated:");


    }
} else {
    JOptionPane.showMessageDialog(null, "Player not found", "Error",
JOptionPane.ERROR_MESSAGE);
}


    }
```

```
    private void statusActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

    }

    private void search1ActionPerformed(java.awt.event.ActionEvent evt) {
```

```
binarysearch algorithm = new binarysearch();
String playerName = tfSearchMain.getText();

if (!playerName.isEmpty()) {
    String searchColumn = "playerName";
    ArrayList<Gadagadars> searchResult = algorithm.binarySearch(gadagadarsList,
playerName, searchColumn);

    if (!searchResult.isEmpty()) {
        System.out.println("Search Results:");

        for (Gadagadars gadagadars : searchResult) {
```

```
            System.out.println("Jersey No: " + gadagadars.getJerseyNo());
            System.out.println("Player Name: " + gadagadars.getPlayerName());
            System.out.println("Address: " + gadagadars.getAddressText());
            System.out.println("Role: " + gadagadars.getRoleText());
            System.out.println("Status: " + gadagadars.getStatusText());
            System.out.println("Marriage Status: " + gadagadars.getMarriageStatus());
            System.out.println("Top Score: " + gadagadars.getTopScoreText());
            System.out.println("Matches Played: " + gadagadars.getMatchesText());

            // Display the information in your GUI as needed
        }
    } else {
        JOptionPane.showMessageDialog(this, "No results found for player: " + playerName,
"Search Results", JOptionPane.INFORMATION_MESSAGE);
    }
} else {
    JOptionPane.showMessageDialog(this, "No player Name entered", "Error",
JOptionPane.INFORMATION_MESSAGE);
}

        // TODO add your handling code here:
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        int option = JOptionPane.showConfirmDialog(
                this,
                "Are you sure you want to go back to the main panel?",
                "Confirmation",
                JOptionPane.YES_NO_OPTION);
```

```
        if (option == JOptionPane.YES_OPTION) {

            System.out.println("Going back to the main panel...");
            // You can replace this with the logic to switch to your main panel
            // or close the current panel, depending on your application structure.
        }
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// Create an instance of the mergesort class
mergesort mergeSort = new mergesort();

// Example: Create a list of Gadagadars
ArrayList<Gadagadars> gadagadarsList = new ArrayList<>();
gadagadarsList.add(new Gadagadars("1", "John", "Address1", "Role1", "Status1", "Married",
"100", "50"));
gadagadarsList.add(new Gadagadars("2", "Jane", "Address2", "Role2", "Status2", "Single", "90",
"60"));

// Define a comparator for sorting by player name (you can change this based on your
requirements)
Comparator<Gadagadars> playerNameComparator =
Comparator.comparing(Gadagadars::getPlayerName);

// Perform the merge sort using the correct property "topScore"
```

```
ArrayList<Gadagadars> sortedListByTopScore = mergeSort.mergeSort(gadagadarsList,
"topScore", playerNameComparator);


// Update the table with the sorted list

updateTable(sortedListByTopScore);
```