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

/**
 * A frame to display a bar chart representing allocation numbers of all the ref
erees
 */
public class BarChartViewer extends JFrame {

    /**
     * Creates a frame where the width is scaled to the size of the referee list
     * and paints a bar chart
     *
     * @param refereeList object which contains Referee instances
     */
    public BarChartViewer(RefList refereeList) {
        //Create the frame
        setTitle("Allocation numbers");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        getContentPane().setBackground(Color.WHITE);

        BarChart chart = new BarChart(refereeList);
        add(chart);

        // calculate a window width based on bar chart width.
        int width = refereeList.getRefereeCount() * (chart.BAR_WIDTH + chart.BAR
_GAP) + (chart.BAR_MARGIN * 2) − chart.BAR_GAP;
        setSize(width, 310);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    /**
     * The component that draws the bar chart
     */
    private class BarChart extends JComponent {
        private final int CHART_HEIGHT = 220;   // The height of the chart area
        private final int BAR_WIDTH = 30;   // The width of each bar
        private final int BAR_GAP = 5;  // The width of the gap between the bars
        private final int BAR_MARGIN = 20;  // Margin around the bar chart area
        private final RefList refereeList; // The list of the referee whose info
rmation will be displayed
        private int maxValue = 0;   // The highest number of referee allocations

        /**
         * Constructor for the BarChart class
         *
         * @param refereeList list of referees which contains allocation numbers
         */
        public BarChart(RefList refereeList) {
            this.refereeList = refereeList;

            // set the max allocation value, used to scale the bar chart
            for (Referee ref : this.refereeList) {
                if (ref.getNumAllocs() > maxValue) {
                    maxValue = ref.getNumAllocs();
                }
            }
        }

        /**
        * The paintComponent method draws a bar for each referee allocation
```

```
        */
        public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D) g;

            //scale the chart depending on the highest number of match allocatio
ns
            int unit = Math.round((float) CHART_HEIGHT / maxValue);

            //draw the grey unit lines
            drawAxis(g2, unit);

            //initial x coordinate to start drawing bars
            int barX = BAR_MARGIN;
            for (Referee ref : refereeList) {
                String id = ref.getRefID();
                int allocNum = ref.getNumAllocs();

                //calculate bar height relative to the chart area
                int barHeight = unit * allocNum;

                drawBar(g2, allocNum, barX, CHART_HEIGHT + 30 − barHeight, BAR_W
IDTH, barHeight, id);
                barX += BAR_WIDTH + BAR_GAP;
            }
        }

        /**
         * Draws a bar with the number of allocations and the referee id
         *
         * @param heading   displays number of allocations above the bar
         * @param x         the x coordinate
         * @param y         the y coordinate
         * @param BAR_WIDTH the width of the bar which is fixed
         * @param barHeight the height of the bar which depends on the number of
 allocations
         * @param id        Referee ID to displayed below the bar
         */
        private void drawBar(Graphics2D g, int heading, int x, int y, int BAR_WI
DTH, int barHeight, String id) {
            g.setColor(Color.MAGENTA);
            g.fillRect(x, y, BAR_WIDTH, barHeight);
            g.setColor(Color.BLACK);
            g.draw(new Rectangle(x, y, BAR_WIDTH, barHeight));
            g.drawString("" + heading, x + 5, y − 5);
            g.drawString(id, x, y + barHeight + 15);
        }

        /**
         * Draws gray lines behind the bar chart
         *
         * @param g    graphics component
         * @param unit the relative pixel distance between each unit
         */
        private void drawAxis(Graphics2D g, int unit) {
            int y = CHART_HEIGHT + 30;

            g.setColor(Color.LIGHT_GRAY);
            for (int i = 0; i <= maxValue; i++) {
                int x1 = BAR_MARGIN / 2;
                int x2 = refereeList.getRefereeCount() * (BAR_WIDTH + BAR_GAP) +
 BAR_GAP + BAR_MARGIN;
                g.drawLine(x1, y, x2, y);
                y -= unit;
            }
        }
    }
}
```

```
import javax.swing.*;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

/**
 * The file processor contains static methods to read and write files
 */
public class FileProcessor {

    private FileProcessor() {
    }

    /**
    * Retrieves information about the referees from a given files and creates a
referee which is added to the RefList object
    * @param refereesInFile the name of the file which contains the ref informat
ion
    * @param referees the list of all the referee objects that have been made
    */
    public static void readIn(String refereesInFile, RefList referees) {
        try (Scanner in = new Scanner(new FileReader(refereesInFile))){
            while (in.hasNextLine()) {
                referees.addRefFromFile(in.nextLine());
            }
        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Could not find the " + refereesInFile
 + " file.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Write referee and match details to a file
     *
     * @param fileOutName Output file
     * @param textToWrite String to be written to fileOutName
     * @return
     */
    public static boolean writeFileOut(String fileOutName, String textToWrite) {
        try (PrintWriter writer = new PrintWriter(fileOutName)) {
            writer.write(textToWrite);
            return true; //IO operations were successful
        }
        catch (IOException e) {
            return false; //IO operations were not successful
        }
    }
}
```

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

/**
 * LittleGUI is a view class which is responsible for displaying and informing t
he {@link RefList} controller class
 * about any inputs from the user. It has 3 distinct modes Add, Edit and Search 
which change the available options to the user
 * based on what inputs have been passed from the MainGUI. The class is also res
ponsible for validating any input for deleting,
 * adding or editing referees which is then passed to the controller.
 *
 * @see RefList
 */
public class LittleGUI extends JFrame implements ActionListener {
    //Reference variables for all the buttons
    private JButton editButton, saveButton, addButton, deleteButton, clearButton
, exitButton;
    //Reference variables for the two comboboxes
    private final String[] qualificationTypeList = {"Type", "NJB", "IJB"};
    private final String[] qualificationList = {"Level", "1", "2", "3", "4"};
    private JComboBox qualificationTypeCombo, qualificationsCombo;
    //Reference variables for all the text fields
    private JTextField fNameField, lNameField, idField, matchField;
    private JLabel idLabel;
    //Reference variables for the radio and check buttons.
    private JRadioButton northRadio, centralRadio, southRadio;
    private JCheckBox northCheck, centralCheck, southCheck;
    //Reference variables for the radio button group. Check buttons don’t have a
 group so an enumeration is used instead.
    private ButtonGroup homeGroup;
    /**
     * Reference variables for the {@link Referee}, {@link RefList}, {@link Main
GUI} classes
     */
    private final Referee referee;
    private final RefList refList;
    private final MainGUI mainGUI;
    //Reference variable to the bottom JPanel which changes in different modes o
f LittleGUI
    private JPanel bottom;

    /* following constants are used to set the GUI mode */
    public static final int ADD = 0;
    public static final int SEARCH = 1;

    /**
     * The constructor for LittleGUI.
     *
     * @param mode        use ADD or SEARCH to set the window mode. Search mode 
displays referee details, Add produces an empty window
     * @param ref         Referee object passed from the MainGUI. Can be null, u
sed for creating an add window
     * @param refereeList The referee list
     * @param refGUI      the main GUI window
     */
    public LittleGUI(int mode, Referee ref, RefList refereeList, MainGUI refGUI)
 {
        // assign the instance variables the values passed from MainGUI
```

```
        refList = refereeList;
        referee = ref;
        mainGUI = refGUI;

        // create the window
        setTitle("Referee Details");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setSize(350, 400);
        setLocationRelativeTo(null);

        //layout the components of the JFrame
        layoutCenter();
        layoutBottom();

        //based on the input from the MainGUI construct the corresponding Little
GUI mode
        if (mode == ADD) {
            showAdd();
        } else if (mode == SEARCH) {
            showSearch();
        }
    }

    /**
     * A method which lays out all the components in the center JPanel
     */
    private void layoutCenter() {
        JPanel center = new JPanel(new GridLayout(6, 1, 2, 2));

        fNameField = new JTextField(15);
        lNameField = new JTextField(15);
        idField = new JTextField(6);
        idField.setEditable(false);
        matchField = new JTextField(7);
        matchField.setEditable(false);
        qualificationTypeCombo = new JComboBox<>(qualificationTypeList);
        qualificationsCombo = new JComboBox<>(qualificationList);
        qualificationsCombo.setLightWeightPopupEnabled(false);

        JPanel fname = new JPanel();
        fname.add(new JLabel("First name:"));
        fname.add(fNameField);

        JPanel lname = new JPanel();
        lname.add(new JLabel("Last name:"));
        lname.add(lNameField);

        JPanel info = new JPanel();
        idLabel = new JLabel("ID: ");
        info.add(idLabel);
        info.add(idField);
        info.add(new JLabel("Matches:"));
        info.add(matchField);

        JPanel qualification = new JPanel();

        qualification.add(new JLabel("Qualification: "));
        qualification.add(qualificationTypeCombo);
        qualification.add(qualificationsCombo);

        JPanel home = new JPanel();
        northRadio = new JRadioButton("North");
```

```java
        northRadio.addActionListener(this);
        centralRadio = new JRadioButton("Central");
        centralRadio.addActionListener(this);
        southRadio = new JRadioButton("South");
        southRadio.addActionListener(this);

        homeGroup = new ButtonGroup();
        homeGroup.add(northRadio);
        homeGroup.add(centralRadio);
        homeGroup.add(southRadio);
        home.add(new JLabel("Home:"));
        home.add(northRadio);
        home.add(centralRadio);
        home.add(southRadio);

        JPanel preference = new JPanel();
        northCheck = new JCheckBox("North", false);
        centralCheck = new JCheckBox("Central", false);
        southCheck = new JCheckBox("South", false);

        //add all the preference buttons
        preference.add(new JLabel("Preference:"));
        preference.add(northCheck);
        preference.add(centralCheck);
        preference.add(southCheck);

        //add all the panels to the center JPanel
        center.add(fname);
        center.add(lname);
        center.add(info);
        center.add(qualification);
        center.add(home);
        center.add(preference);
        add(center, BorderLayout.CENTER);
    }

    /**
     * A method which lays down all the components from the bottom JPanel.
     */
    private void layoutBottom() {
        bottom = new JPanel();

        clearButton = new JButton("Clear");
        editButton = new JButton("Edit");
        saveButton = new JButton("Save");
        addButton = new JButton("Add");
        deleteButton = new JButton("Delete");
        exitButton = new JButton("Exit");

        clearButton.addActionListener(this);
        editButton.addActionListener(this);
        saveButton.addActionListener(this);
        addButton.addActionListener(this);
        deleteButton.addActionListener(this);
        exitButton.addActionListener(this);

        bottom.add(saveButton);
        bottom.add(addButton);
        bottom.add(clearButton);
        bottom.add(editButton);
        bottom.add(deleteButton);
        bottom.add(exitButton);
```

```java
        add(bottom, BorderLayout.SOUTH);
    }

    /**
     * A method which hides all bottom panel components
     */
    private void hideBottomComponents() {
        for (Component c : bottom.getComponents()) {
            c.setVisible(false);
        }
    }

    /**
     * A method which edits the visibility and the editable property of certain
components to show the appropriate Add mode.
     */
    private void showAdd() {
        hideBottomComponents();
        addButton.setVisible(true);
        exitButton.setVisible(true);
        matchField.setEditable(true);
        clearButton.setVisible(true);
        idField.setVisible(false);
        idLabel.setVisible(false);
    }

    /**
     * A method which edits the visibility and the editable property of certain
components to show the appropriate Search mode.
     */
    private void showSearch() {
        hideBottomComponents();
        // set up the relevant GUI components
        fNameField.setEditable(false);
        lNameField.setEditable(false);
        matchField.setEditable(false);
        qualificationTypeCombo.setEnabled(false);
        qualificationsCombo.setEnabled(false);
        editButton.setVisible(true);
        deleteButton.setVisible(true);
        exitButton.setVisible(true);

        for (Component homeRadio : Collections.list(homeGroup.getElements())) {
            homeRadio.setEnabled(false);
        }
        northCheck.setEnabled(false);
        centralCheck.setEnabled(false);
        southCheck.setEnabled(false);

        //set Referee details
        fNameField.setText(referee.getFName());
        lNameField.setText(referee.getLName());
        idField.setText(referee.getRefID());
        matchField.setText("" + referee.getNumAllocs());
        qualificationTypeCombo.setSelectedItem(referee.getQualificationType());
        qualificationsCombo.setSelectedItem(String.valueOf(referee.getQualificat
ionLevel()));

        //find home area
        if (referee.getHomeArea() == 0) {
            northRadio.setSelected(true);
        } else if (referee.getHomeArea() == 1) {
            centralRadio.setSelected(true);
        } else {
            southRadio.setSelected(true);
        }

        //find preferences
        northCheck.setSelected(referee.getTravelInfo(Referee.NORTH));
        centralCheck.setSelected(referee.getTravelInfo(Referee.CENTRAL));
        southCheck.setSelected(referee.getTravelInfo(Referee.SOUTH));
    }

    /**
     * A method which edits the visibility and the editable property of certain
components to show the appropriate Edit mode.
     */
    private void showEdit() {
        hideBottomComponents();

        saveButton.setVisible(true);
        exitButton.setVisible(true);
        matchField.setEditable(true);
        qualificationTypeCombo.setEnabled(true);
        qualificationsCombo.setEnabled(true);

        for (Component homeRadio : Collections.list(homeGroup.getElements())) {
            homeRadio.setEnabled(true);
        }

        northCheck.setEnabled(true);
        centralCheck.setEnabled(true);
        southCheck.setEnabled(true);
    }

    /**
     * A method which handles any events submitted from the action listeners
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == editButton) {
            showEdit();
        }
        if (e.getSource() == saveButton) {
            processSave();
        }
        if (e.getSource() == addButton) {
            processAdd();
        }
        if (e.getSource() == deleteButton) {
            processDelete();
        }
        if (e.getSource() == exitButton) {
            dispose();
        }
        if (e.getSource() == clearButton) {
            clearFields();
        }
        if (e.getSource() == northRadio) {
            northCheck.setSelected(true);
        }
        if (e.getSource() == centralRadio) {
            centralCheck.setSelected(true);
        }
```

```java
        if (e.getSource() == southRadio) {
            southCheck.setSelected(true);
        }
    }

    /**
     * A method which processes the editing of the details of a referee based on
 the input from LittleGUI.
     */
    private void processSave() {
        if (validateFields()) {
            setFields();
            mainGUI.updateTable();
            dispose();
            JOptionPane.showMessageDialog(this, "The referee details have been updated.",
                    "Success", JOptionPane.INFORMATION_MESSAGE);
        }
    }

    /**
     * A method which processes the addition of a new referee to RefList based o
n the input from LittleGUI. THe ID is calculated automatically in RefList.
     */
    private void processAdd() {
        if (validateFields()) {
            //If the referee with the same names has already been added to the d
atabase return an error
            if (refList.findRef(fNameField.getText(), lNameField.getText()) != n
ull) {
                errorPane("Adding referee failed. The referee already exists in the database.");
            }
            /**
             * If the {@link RefList} has already have 12 referees added return
an error
             */
            else if (!refList.checkForSpace()) {
                errorPane("Adding referee failed. There can’t be more than 12 referees in the database.")
;
            }
            /**
             * if all checks have been passed add a new referee to {@link RefLis
t}, update the MainGUI and display a message to feedback sucess
             */
            else {
                refList.addRefFromGui(fNameField.getText(), lNameField.getText()
, (String) qualificationTypeCombo.getSelectedItem() + Integer.parseInt((String)
(qualificationsCombo.getSelectedItem())),
                        Integer.parseInt(matchField.getText()), getHomeArea(), g
etPreferences());

                mainGUI.updateTable();
                clearFields();
                JOptionPane.showMessageDialog(this, "The referee has been added to the datab
ase.",
                        "Success", JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }

    /**
     * A method which processes the deletion of a referee from the reflist based
 on the input (first and last name) from the user from LittleGUI
```

```
     */
    private void processDelete() {
        //first check if ref has been allocated
        if (referee.isAllocated()) {
            errorPane("The referee cannot be deleted as he has been allocated to a match");
            return;
        }

        //Display a prompt to the user to confirm that they want to delete the r
eferee
        int dialogResult = JOptionPane.showConfirmDialog(this, "Would you like to delet
e this referee?", "Warning", JOptionPane.YES_NO_OPTION);

        //if yes call a method from RefList to delete the referee if the deletio
n was unsuccessful, return an error
        if (dialogResult == JOptionPane.YES_OPTION) {
            boolean deleted = refList.deleteRef(referee.getFName(), referee.getL
Name());
            if (deleted) {
                mainGUI.updateTable();
                dispose();
                JOptionPane.showMessageDialog(this, "The referee has been deleted from the d
atabase.",
                        "Success", JOptionPane.INFORMATION_MESSAGE);
            } else {
                dispose();
                errorPane("There was a problem deleting the referee. Please check if the referee still exists
 in the database.");
            }
        }
    }

    /**
     * A method which sets all the instance variables in referee based on the cu
rrent inputs in the JComponents of LittleGUI
     */
    private void setFields() {
        referee.setNumAllocs(Integer.parseInt(matchField.getText()));
        referee.setQualificationType((String) qualificationTypeCombo.getSelected
Item());
        referee.setQualificationLevel(Integer.parseInt((String) (qualificationsC
ombo.getSelectedItem())));
        referee.setHomeArea(getHomeArea());
        referee.setTravelInfo(getPreferences());
    }

    /**
     * A method which validates the input from the JComponents of LittleGUI and
returns an error if there is any discrepancy
     *
     * @return boolean
     */
    private boolean validateFields() {
        //Checks if either of the name fields are empty strings
        if (fNameField.getText().equals("") || lNameField.getText().equals(""))
{
            errorPane("The referee names cannot be empty strings.");
            return false;
        }

        //Checks that the name of referee contains only letters
        if (!(validName(fNameField.getText())) || !(validName(lNameField.getText
```

```
())))) {
            errorPane("The referee names should only contain letters.");
            return false;
        }

        //Catches any exceptions if the user enters something else than an integ
er
        try {
            if (Integer.parseInt(matchField.getText())<0)
                throw new NumberFormatException();
        } catch (NumberFormatException nfe) {
            errorPane("Please enter a positive integer number for the number of matches.");
            return false;
        }

        //Checks if a qualification type has been selected
        if (qualificationTypeCombo.getSelectedItem().equals("Type")) {
            errorPane("Please select a qualification type.");
            return false;
        }

        //Checks if a qualification level has been selected
        if (qualificationsCombo.getSelectedItem().equals("Level")) {
            errorPane("Please select a qualification level.");
            return false;
        }

        //Checks if a home area has been selected
        if (getHomeArea().equals("")) {
            errorPane("Please select a home area.");
            return false;
        }

        //Checks if at least one preference has been selected
        if (getPreferences().equals("NNN")) {
            errorPane("Please select at least one preference.");
            return false;
        }

        //Calls a method to check if the home are is reflected in the preference
s
        if (!checkHomePreference()) {
            errorPane("The preferences should include the home area of the referee.");
            return false;
        }

        return true;
    }

    /**
     * Returns true if name is alphabetic with an optional single hyphen only if
 it is not the first or last character
     *
     * @param name First or last name
     * @return true if name is valid
     */
    private boolean validName(String name) {
        return name.matches("^(?!-)[a-zA-Z]+-?[a-zA-Z]*$(?<!-)");
    }

    /**
     * A method which checks the current states of the Home Area Radio buttons i
```

```
n LittleGUI and returns a string suitable to be passed to {@link RefList}
     *
     * @return home area as string, empty if not selected
     */
    private String getHomeArea() {
        if (northRadio.isSelected())
            return "North";
        else if (centralRadio.isSelected())
            return "Central";
        else if (southRadio.isSelected())
            return ("South");
        else
            return "";
    }

    /**
     * A method which checks the current states of the Preference check buttons
in LittleGUI and returns a string suitable to be passed to {@link RefList}
     *
     * @return a three letter string representing preferred locations
     */
    private String getPreferences() {
        char[] travelInfo = {'N', 'N', 'N'};

        if (northCheck.isSelected()) {
            travelInfo[0] = 'Y';
        }
        if (centralCheck.isSelected()) {
            travelInfo[1] = 'Y';
        }
        if (southCheck.isSelected()) {
            travelInfo[2] = 'Y';
        }
        return new String(travelInfo);
    }

    /**
     * Check if home area and preferred area are both selected.
     *
     * @return true if home area and preference both match
     */
    private boolean checkHomePreference() {
        return (northRadio.isSelected() && northCheck.isSelected()) ||
                (centralRadio.isSelected() && centralCheck.isSelected()) ||
                (southRadio.isSelected() && southCheck.isSelected());
    }

    /**
     * A method which clears all the fields in LittleGUI
     */
    private void clearFields() {
        fNameField.setText("");
        lNameField.setText("");
        matchField.setText("");
        qualificationTypeCombo.setSelectedItem("Type");
        qualificationsCombo.setSelectedItem("Level");

        homeGroup.clearSelection();

        northCheck.setSelected(false);
        centralCheck.setSelected(false);
        southCheck.setSelected(false);
```

```
    }
    /**
     * Creates a JOption pane with a custom error message
     *
     * @param errorMessage Message to display on the JOptionPane
     */
    private void errorPane(String errorMessage) {
        JOptionPane.showMessageDialog(this, errorMessage, "Error", JOptionPane.ER
ROR_MESSAGE);
    }
}
```

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

/**
 * Main GUI which allows the user to enter match details, view information about
 the referees and search for a referee.
 */
public class MainGUI extends JFrame implements ActionListener {
        private final int INVALID_INFO = -1;
        private JButton allocateRefButton, barChartButton, addRefButton, searchR
efButton, saveExitButton;  // the buttons which allow the user to allocate a ref
, see the bar chart, add/view a ref and save and exit
        private JRadioButton northButton, centralButton, southButton, juniorButt
on, seniorButton;   // the radio buttons to select the match location and level
        private ButtonGroup locationGroup, levelGroup;  // the groups for the ra
dio buttons to ensure that they are mutually exclusive
        private JTextField weekField, firstNameField, lastNameField;    // the t
ext fields to enter the week in which a match takes place and the name of the re
f to be searched for
        private JTextArea centerText; // text area to display the referees which
 have been allocated to a match or displays an error message
        private DefaultTableModel model;    // the model to set the features of
the JTable
        private JTable centerTable;     // the JTable which displays the informa
tion about the referees
        private final JTabbedPane tabbedPane = new JTabbedPane();   // the tabbe
d pane which holds the table and the text area to display the allocated referees
        private final RefList refereeList;    // a RefList object which contains
 all the referees that have been entered so far
        private final MatchList matchList;    // a MatchList object which contai
ns all the matches that have been entered
        private final String matchAllocsFile = "MatchAllocs.txt";
        private final String refsOutFile = "RefereesOut.txt";

        /**
         * Constructs the main GUI window and creates the MatchList and RefList
objects
         */
        public MainGUI() {
                this.setTitle("Javaball Referee Selection");
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setSize(600, 400);
                this.setLocationRelativeTo(null);
                refereeList = new RefList();
                FileProcessor.readIn("RefereesIn.txt", refereeList);
                matchList = new MatchList();
                this.layoutComponents();
        }

        /**
         * Sets out the different GUI components within the JFrame
         */
        private void layoutComponents() {
                // Create allocRefsPanel JPanel which will contain the match all
ocation components
                JPanel allocRefsPanel = new JPanel();
                allocRefsPanel.setLayout(new BoxLayout(allocRefsPanel, BoxLayout
.Y_AXIS));
```

```
                allocRefsPanel.setBorder(BorderFactory.createTitledBorder("Allocat
e Referees"));

                // Create internal JPanels for each of the components
                JPanel weekPanel = new JPanel();
                JPanel locationPanel = new JPanel();
                JPanel levelPanel = new JPanel();
                JPanel allocateButtonPanel = new JPanel();

                // Create label and textField for match week number
                weekPanel.add(new JLabel("Week Number (1-52):"));
                weekField = new JTextField(2);
                weekPanel.add(weekField);

                //Create label and radio buttons for match location
                locationPanel.add(new JLabel("Match Location:"));
                northButton = new JRadioButton("North");
                centralButton = new JRadioButton("Central");
                southButton = new JRadioButton("South");

                //Group the location JRadioButtons so that they are mutually exc
lusive
                locationGroup = new ButtonGroup();
                locationGroup.add(northButton);
                locationGroup.add(centralButton);
                locationGroup.add(southButton);
                locationPanel.add(northButton);
                locationPanel.add(centralButton);
                locationPanel.add(southButton);


                // Create label and radio buttons for level
                levelPanel.add(new JLabel("Level:"));
                juniorButton = new JRadioButton("Junior");
                seniorButton = new JRadioButton("Senior");

                // Group the level JRadioButtons so they are mutually exclusive
                levelGroup = new ButtonGroup();
                levelGroup.add(juniorButton);
                levelGroup.add(seniorButton);
                levelPanel.add(juniorButton);
                levelPanel.add(seniorButton);

                //Create label and button for finding suitable referee
                allocateRefButton = new JButton("Allocate");
                allocateRefButton.addActionListener(this);
                allocateButtonPanel.add(allocateRefButton);

                // Add internal panels to the allocRefsPanel JPanel
                allocRefsPanel.add(weekPanel);
                allocRefsPanel.add(locationPanel);
                allocRefsPanel.add(levelPanel);
                allocRefsPanel.add(allocateButtonPanel);

                // Create searchPanel JPanel which will contain the search refer
ee components
                JPanel searchPanel = new JPanel(new GridLayout(3, 1));
                searchPanel.setBorder(BorderFactory.createTitledBorder("Search for
Referee"));

                // Create the internal JPanels for each of the components
                JPanel firstNamePanel = new JPanel();
```

```java
		JPanel lastNamePanel = new JPanel();
		JPanel searchButtonPanel = new JPanel();

		//Create label and button for first name
		firstNamePanel.add(new JLabel("First Name:"));
		firstNameField = new JTextField(10);
		firstNamePanel.add(firstNameField);

		// Create label and button for last name
		lastNamePanel.add(new JLabel("Last Name:"));
		lastNameField = new JTextField(10);
		lastNamePanel.add(lastNameField);

		// Create the button for searching for the referee
		searchRefButton = new JButton("Search");
		searchRefButton.addActionListener(this);
		searchButtonPanel.add(searchRefButton);

		// Add the internal panels to searchPanel JPanel
		searchPanel.add(firstNamePanel);
		searchPanel.add(lastNamePanel);
		searchPanel.add(searchButtonPanel);

		//Create the topSections JPanel which will contain both the allo
cRefsPanel and searchPanel JPanel so that they sit side by side
		JPanel topSections = new JPanel(new GridBagLayout());
		GridBagConstraints c = new GridBagConstraints();
		c.gridwidth = 2;
		c.gridheight = 1;
		c.fill = GridBagConstraints.VERTICAL;
		topSections.add(allocRefsPanel, c);
		topSections.add(searchPanel, c);

		// Use the setCenterTable method to populate the table and add i
t to the scrollpane
		setCenterTable();
		JScrollPane tableScroll = new JScrollPane(centerTable);
		centerTable.setFillsViewportHeight(true);

		// Create the text field which can be used to display informatio
n about the allocated referees
		centerText = new JTextArea();
		centerText.setFont(new Font("Monospaced", Font.PLAIN, 12));
		centerText.setEditable(false);
		JScrollPane textScroll = new JScrollPane(centerText);

		// add the table and text area to the CardLayout handler
		tabbedPane.addTab("All Referees", tableScroll);
		tabbedPane.addTab("Allocated Referees", textScroll);

		// Create the centerLayout GUI which will contain the main secti
ons and the table
		JPanel centerLayout = new JPanel(new GridLayout(2, 1));
		centerLayout.add(topSections);
		centerLayout.add(tabbedPane);
		this.add(centerLayout, BorderLayout.CENTER);

		// Create bottomButtons JPanel
		JPanel bottomButtons = new JPanel();
		this.add(bottomButtons, BorderLayout.SOUTH);

		//Create button for bar chart and add to internal JPanel
```

```java
		barChartButton = new JButton("View allocations charts");
		barChartButton.addActionListener(this);
		bottomButtons.add(barChartButton);

		//Create button for adding new ref
		addRefButton = new JButton("Add referee");
		addRefButton.addActionListener(this);
		bottomButtons.add(addRefButton);

		//Create button for saving and exiting
		saveExitButton = new JButton("Save and Exit");
		saveExitButton.addActionListener(this);
		bottomButtons.add(saveExitButton);
	}

	/**
	 * Decides which action will be taken depending on which input the user h
as given
	 * @param e the action event which results from the user pressing one of
the buttons
	 */
	public void actionPerformed(ActionEvent e) {
		if (e.getSource() == barChartButton) {
			BarChartViewer chart = new BarChartViewer(refereeList);
		}
		if (e.getSource() == addRefButton) {
			showLittleGui(LittleGUI.ADD, null);
		}
		if (e.getSource() == allocateRefButton) {
			checkForSuitableRefs();
			clearAllocComponents();
			updateTable();
		}
		if (e.getSource() == searchRefButton) {
			processSearch();
			clearNameFields();
		}
		if(e.getSource() == saveExitButton) {
			processSaveExit();
		}
	}

	/**
	 * Create the model for the JTable, ensuring it is non editable and the d
ata is displayed correctly
	 */
	private void setCenterTable() {
		final Object[] columnNames = {"ID", "Name", "Qualification", "Allocatio
ns", "Home", "North", "Central", "South"}; // the names for each of the columns in t
he JTable
		model = new DefaultTableModel(columnNames, 0) {
			public boolean isCellEditable(int row, int col) {
				return false;
			}
			public Class<?> getColumnClass(int colIndex) {
				return getValueAt(0, colIndex).getClass();
			}

		};
		populateTable();
	}
	/**
	* Add the information for each of the referees into a seperate row
	*/
	private void populateTable() {
		for (Referee ref : refereeList) {
			model.addRow(new Object[]{
					ref.getRefID(),
					ref.getFName() +" "+ ref.getLName(),
					ref.getQualificationType() + ref.getQual
ificationLevel(),
					ref.getNumAllocs(),
					ref.getHomeString(),
					ref.getTravelInfo(Referee.NORTH),
					ref.getTravelInfo(Referee.CENTRAL),
					ref.getTravelInfo(Referee.SOUTH),
			});
		}
		// Create JTable and add it to the scroll pane
		centerTable = new JTable(model);
		centerTable.setGridColor(Color.LIGHT_GRAY);
		centerTable.getColumnModel().getColumn(1).setPreferredWidth(150)
;
	}

	/**
	* Populates the table with the updated referee information
	*/
	public void updateTable()
	{
		model.setRowCount(0);
		populateTable();
	}

	/**
	 * Shows either a blank add referee window or displays search results
	 *
	 * @param mode sets whether the information about the referee is editabl
e or not
	 * @param ref  the referee object which will be displayed and can be edi
ted
	 */
	private void showLittleGui(int mode, Referee ref) {
		LittleGUI littleGUI = new LittleGUI(mode, ref, refereeList, this
);
		littleGUI.setVisible(true);
	}

	/**
	 * Retrieves the referees names from the textfields and checks if the re
feree exists;
	 * if so, opens LittleGUI to display info on searched ref
	 */
	private void processSearch() {
		//get ref’s first and last name from GUI
		String firstName = firstNameField.getText().trim();
		String lastName = lastNameField.getText().trim();

		if(!firstName.isEmpty() && !lastName.isEmpty()) {
			Referee ref = refereeList.findRef(firstName, lastName);
			if (ref != null)
				showLittleGui(LittleGUI.SEARCH, ref);
			else {
```

```
				errorPane("The referee " + firstName + " " + lastNam
e + " " + "was not found in the database.");
				clearNameFields();
			}
		}
		else {
			errorPane("First Name and Last Name fields cannot be empty.");
		}
	}

	/**
	 * Gets match info from GUI and checks for suitable referees for that ma
tch;
	 * calls methods to allocate 2 refs to the match and display information
	 * on the suitable refs in the GUI
	 */
	private void checkForSuitableRefs() {
		// First make sure there is room for another match
		if (matchList.getNoMatches() == 52) {
			errorPane("All the weeks in the year are allocated.");
			return; //if no room for more matches, exit method
		}

		// If there is room for another match, get match info input by u
ser
		int week = getValidWeekNum(weekField.getText());
		int loc = getLocationInfo();
		//check that all info has been input and is OK
		if (isLevelSelected() && week != INVALID_INFO && loc != INVALID_
INFO) {
			// Check if week does not already have a match scheduled
			if (!matchList.checkWeekAllocation(week)) {
				errorPane("Week " + week + " is already allocated.");
				return; // If week is already taken, exit method
			}
			boolean senMatch = seniorButton.isSelected(); // Check i
f match is senior or junior
			// After all match info has been checked, get list of su
itable refs
			List<Referee> suitableRefs = refereeList.getSuitableRefs
(loc, senMatch);
			if (suitableRefs.size() < 2) { // If not enough suitable
 refs found, display message
				displayNoSuitableRefs();
			}
	    // calls method to allocate 2 most suitable refs to match
	    else {
				allocateTwoRefs(suitableRefs, week, loc, senMatc
h);
		displayAllocatedRefs(suitableRefs);
			}
		}
	}

	/**
	 * Retrieves the two most suitable refs and passes them as a parameter w
hen creating the match object
	 * along with other match info
	 * @param suitRefs the full list of suitable refs
	 * @param weekNumber week number when the match is on
	 * @param place the location of the match
	 * @param senior True if match requires senior referee
```

```
         */
        private void allocateTwoRefs(List<Referee> suitRefs, int weekNumber, int
 place, boolean senior) {
                Referee ref1 = suitRefs.get(0); // Most suitable ref
                Referee ref2 = suitRefs.get(1); // Second most suitable ref

                String ref1Name = ref1.getFName() + " " + ref1.getLName();
                String ref2Name = ref2.getFName() + " " + ref2.getLName();
                // Create new match
                matchList.addMatch(weekNumber, place, senior, ref1Name, ref2Name
);
                // Increment the number of allocations of the 2 allocated refs
                ref1.incrementAllocs();
                ref2.incrementAllocs();
        }

        /**
         * Retrieves the week number from its respective text field and ensures
it is valid
         * @return INVALID_INFO constant if the week is invalid, week number oth
erwise
         */
        private int getValidWeekNum(String weekNum) {
                int week = INVALID_INFO;

                try {
                        week = Integer.parseInt(weekNum);
                        if (week < 1 || week > MatchList.MAX_MATCHES) {
                                week = INVALID_INFO;
                                throw new NumberFormatException();
                        }
                } catch (NumberFormatException nfx) {
                        errorPane("Please enter a week number between 1 and " + MatchList.M
AX_MATCHES);
                }
                return week;
        }

        /**
         * Retrieves the location of the match and returns it as a constant valu
e which is set down in the Referee class
         * @return the final int which is used as an indicator of the location
         */
        private int getLocationInfo() {
                if (northButton.isSelected()) {
                        return Referee.NORTH;
                } else if (centralButton.isSelected()) {
                        return Referee.CENTRAL;
                } else if (southButton.isSelected()) {
                        return Referee.SOUTH;
                } else {
                        errorPane("Please select the match location.");
                        return INVALID_INFO;
                }
        }

        /**
         * Checks if the match level has been selected
         * @return true if it has, false otherwise
         */
        private boolean isLevelSelected() {
                if (juniorButton.isSelected() || seniorButton.isSelected())
```

```
                        return true;
                else {
                        errorPane("Please select the match level.");
                        return false;
                }
        }

        /**
         * Inputs the error message that not enough referees were found into the
 text area.
         * Hides the JTable but makes the button to view the table visible
         */
        private void displayNoSuitableRefs()
        {
                tabbedPane.setSelectedIndex(1);
                centerText.setText("Not enough suitable refs found");
        }

        /**
         * Inputs the suitable referee list and selected referees into the text
area.
         * Hides the JTable but makes the button to view the table visible
         * @param suitableRefs List of suitable referees
         */
        private void displayAllocatedRefs(List<Referee> suitableRefs) {
                // switch to the text area in the second tab
                tabbedPane.setSelectedIndex(1);

                StringBuilder display = new StringBuilder();
                display.append("The referees allocated to the match are: \n")
                                .append(suitableRefs.get(0).getFName() + " ")
                                .append(suitableRefs.get(0).getLName())
                                .append(" and ")
                                .append(suitableRefs.get(1).getFName() + " ")
                                .append(suitableRefs.get(1).getLName())
                                .append("\n\nThe referees which are suitable for the match are: \n"
);

                for (Referee aSuitableRef : suitableRefs) {
                        display.append(String.format("%−35s%s %−4s%n",

aSuitableRef.getFName() + " " + aSuitableRef.getLName(),

"Allocations:",

aSuitableRef.getNumAllocs()));
                }

                centerText.setText(display.toString());
                centerText.setCaretPosition(0);
        }

        /**
         * Clears the components for inputting match info
         */
        private void clearAllocComponents() {
                weekField.setText("");
                locationGroup.clearSelection();
                levelGroup.clearSelection();
        }

        /**
```

```
         * Clears the name text fields in the search area of the GUI
         */
        private void clearNameFields() {
                firstNameField.setText("");
                lastNameField.setText("");
        }

        private void processSaveExit() {
                String matchAllocsText = matchList.getMatchAllocsText();
                String refReport = refereeList.getRefsOutText();
                boolean matchFileMade = FileProcessor.writeFileOut(matchAllocsFi
le, matchAllocsText);
                boolean refFileMade = FileProcessor.writeFileOut(refsOutFile, re
fReport);
                //if IO operations were successful, exit program
                if(matchFileMade && refFileMade)
                        System.exit(0);
        }

        /**
         * Creates a JOption pane with a custom error message
         *
         * @param errorMessage Message to display on the JOptionPane
         */
        private void errorPane(String errorMessage) {
                JOptionPane.showMessageDialog(this, errorMessage, "Error", JOptio
nPane.ERROR_MESSAGE);
        }
}
```

```
/**
 * Defines an object representing a single match
 */

public class Match {
    /**
     * instance variables
     */
    private final int weekNumber;
    private final String matchLevel;
    private String matchArea;
    private final String refOne;
    private final String refTwo;

    /**
     * Constructor for Match class
     *
     * @param week week number
     * @param area area code
     * @param isSenior boolean that represents referee level
     */
    public Match(int week, int area, boolean isSenior, String firstRef, String s
econdRef) {
        weekNumber = week;
        // area is being passed as an int so have to check which it is and set m
atchArea String accordingly
        if (area == Referee.NORTH)
            matchArea = "North";
        else if (area == Referee.CENTRAL)
            matchArea = "Central";
        else if (area == Referee.SOUTH)
            matchArea = "South";

        matchLevel = isSenior ? "Senior" : "Junior";

        refOne = firstRef;
        refTwo = secondRef;
    }

    /**
     * Accessor method for week number
     */
    public int getWeekNo() {
        return weekNumber;
    }

    /**
     * Accessor method for match level
     */
    public String getLevel() {
        return matchLevel;
    }

    /**
     * Accessor method for area of match
     */
    public String getArea() {
        return matchArea;
    }

    /**
     * Constructor and accessor method for refOne
     */
    public String getRefOne() {
        return refOne;
    }

    /**
     * Constructor and accessor method for refTwo
     */
    public String getRefTwo() {
        return refTwo;
    }

    /**
     * Method to get formatted line for match report document
     */
    public String getMatchLine() {
        return String.format("%−8d%−12s%−12s%−20s%−20s%n", weekNumber, matchLevel
, matchArea, refOne, refTwo);
    }
}
```

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 * Maintains a list of Match objects
 * The methods allow Matches to be added to the list
 */
public class MatchList implements Iterable<Match> {
    public static final int MAX_MATCHES = 52;
    private final List<Match> matchList;

    /**
     * Constructor for MatchList class
     */
    public MatchList() {
        matchList = new ArrayList<>();
    }

    /**
     * Checks if a given week can be allocated
     * @param week week number to check
     * @return true if week is not allocated, false when allocation exists
     */
    public boolean checkWeekAllocation(int week) {
        return (getMatch(week) == null);
    }

    //TODO temporary name
        /**
         * Takes match details, makes new match object and adds it to matchList
         * @param week the week the match is in
         * @param loc the match location
         * @param senior a boolean indicating whether match is senior or not
         * @param ref1Nm the full name of the first ref allocated to the match
         * @param ref2Nm the full name of the second ref allocated to the match
         */
    public void addMatch(int week, int loc, boolean senior, String ref1Nm, Strin
g ref2Nm) {
        Match newMatch = new Match(week, loc, senior, ref1Nm, ref2Nm);
        matchList.add(newMatch);
    }

    /**
     * get number of matches currently in the match list
     */
    public int getNoMatches() {
        return matchList.size();
    }

    private Match getMatch(int matchWeekNo) {
        for (Match match : matchList) {
            if (match.getWeekNo() == matchWeekNo) {
                return match;
            }
        }
        return null;
    }

        public String getMatchAllocsText() {
        String title = "Match details\r\n\r\n";
        String tableHeader = String.format("%−8s%−12s%−12s%−20s%−20s%n%n", "Week"
, "Level", "Area", "Referee 1", "Referee 2");

        StringBuilder matchesOutBuilder = new StringBuilder();
                for(Match match : matchList) {
                        matchesOutBuilder.append(match.getMatchLine());
                }

                return title + tableHeader + matchesOutBuilder;
        }

    /**
     * Returns an iterator over elements of type Match.
     *
     * @return an Iterator.
     */
    public Iterator<Match> iterator() {
        return matchList.iterator();
    }
}
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class RefList implements Iterable<Referee> {
        private static final int MAX_REFS = 12;
        private final List<Referee> refList;

        public RefList() {
                refList = new ArrayList<>();
        }

        /**
         * Checks if there is still room for another Referee, returns true if th
ere is.
         * @return boolean indicating whether there is room
         */
        public boolean checkForSpace() {
                return (refList.size() < MAX_REFS);
        }

        /**
         * Creates a Referee from info read from input file and adds it to the e
nd of refList.
         * @param refString the info line read in from the file
         */
        public void addRefFromFile(String refString) {
                Referee newRef = new Referee(refString);
                refList.add(newRef);
        }

        /**
         * Gets the current amount of Referees in refList
         * @return an int corresponding to the size of refList
         */
        public int getRefereeCount() {
                return refList.size();
        }

        /**
         * Used to add a Referee object fom information input to the GUI. First
calls method to create the ref’s ID then
         * creates a new Referee, checks where it should be placed in refList ba
sed on order of IDs and adds it
         * to refList in that position.
         * @param firstNm the first name input to the GUI
         * @param lastNm the last name input to the GUI
         * @param qual the ref’s qualification
         * @param allocs the ref’s number of match allocations
         * @param home the ref’s home area
         * @param travelInfo information on where the ref will travel
         */
        public void addRefFromGui(String firstNm, String lastNm, String qual, in
t allocs, String home, String travelInfo) {
                String newId = createId(firstNm, lastNm);
                //create string to pass to Referee constructor
                String refData = newId + " " + firstNm + " " + lastNm + " " + qua
l + " " + allocs + " " + home + " " + travelInfo;
                Referee newRef = new Referee(refData);

                //place new Referee in refList based on ID order
                boolean added = false;
                int i = 0;
                while (!added && i < refList.size()) { //loop through refList
                        String otherID = refList.get(i).getRefID(); //get ID to
compare
                        //if new ref's ID should be before compared ID, add new
ref at compared ref's position
                        if (newId.compareTo(otherID) < 0) {
                                refList.add(i, newRef);
                                added = true; //new ref has been added, exit loo
p
                        } else
                                i++; //if new ref should be placed after compare
d ref, check next position
                }
                //If new ID does not come before any existing ID, place new Ref
at end of refList
                if (!added)
                        refList.add(newRef);
        }

        /**
         * generates a unique ID for a ref added from the GUI based on first and
 last name
         * @param first the new ref's first name
         * @param last the new ref's last name
         * @return the new ref's ID
         */
        private String createId(String first, String last) {
                //get letter part of ID from initials
                String letterPart = ("" + first.charAt(0) + last.charAt(0)).toUp
perCase();
                int numPart = 1; // set number part to 1 initially

                for (Referee ref : refList) { //loop through refList
                        String otherInitials = ref.getRefID().substring(0, 2); /
/get compared ref's initials
                        if (letterPart.equals(otherInitials))
                                numPart++; //if matching initials found, increme
nt the number part of ID
                }
                //concatenate the letter part and number part
                return letterPart + numPart;
        }

        /**
         * used to delete a Referee from refList
         * @param first the ref's first name
         * @param last the ref's last name
         * @return true if the ref of that name was found and deleted, false if
not found
         */

        public boolean deleteRef(String first, String last) {
                Referee findRefResult = findRef(first, last); //look for ref in
refList

                if (findRefResult != null) { //if ref exists, remove from refLis
t
                        refList.remove(findRefResult);
                        return true;
                } else //ref not found, return false
```

```
                        return false;
        }

        /**
         * looks for a Referee in refList based on their name
         * @param first the ref's first name
         * @param last the ref's last name
         * @return Referee object if found, else null
         */
        public Referee findRef(String first, String last) {
                Referee ref = null;
                boolean found = false;
                int refIndex = 0;
                String firstToSearch = first.toLowerCase();
                String lastToSearch = last.toLowerCase();

                while(!found && refIndex < refList.size()) { //loop through refL
ist
                        ref = refList.get(refIndex);
                        String otherFirst = ref.getFName().toLowerCase();
                        String otherLast = ref.getLName().toLowerCase();
                        //compare name to name of ref at position i
                        if(otherFirst.equals(firstToSearch) && otherLast.equals(
lastToSearch))
                                found = true; //if found, exit loop
                        else
                                refIndex++; //if not found check against next re
f
                }

                if(!found) //if ref not in refList return null
                        return null;
                else //if ref was found, return the object
                        return ref;
        }

        /**
         * takes in information about a match and creates a list of suitable ref
s, in order of suitability.
         * makes an array of Referees from refList then sorts that array in orde
r of number of allocations.
         * then loops through the sorted array, checking each ref's suitability
for the match based on their
         * home area and willingness to travel and placing them in the appropria
te position in a new List:
         * first the local refs, then the adjacent refs, then the non-adjacent r
efs
         * @param matchLoc the location of the match
         * @param seniorMatch boolean indicating whether match is senior or not
         * @return List populated with suitable refs in order of suitability for
 the match
         */

        public List<Referee> getSuitableRefs(int matchLoc, boolean seniorMatch)
{
                //make an array of Referees from refList
                Referee[] arrayToSort = new Referee[refList.size()];
                arrayToSort = refList.toArray(arrayToSort);
                Arrays.sort(arrayToSort); //sort the array on number of allocati
ons

                //count of local and adjacent refs already in suitable refs list
                int numLocalRefs = 0;
                int numAdjRefs = 0;
                //arrayList to be populated with suitable referees
                List<Referee> suitableRefs = new ArrayList<>();

                for (Referee ref : arrayToSort) { //loop through array sorted on
 number of allocations
                        int home = ref.getHomeArea(); //get ref's home area
                        boolean refWillTravel = ref.getTravelInfo(matchLoc); //g
et whether ref will travel to match location

                        if (!seniorMatch || ref.checkIfQualified()) { //first ch
eck if ref is qualified for the match
                                //if ref lives in match location, add ref in pos
ition after last local ref
                                if (home == matchLoc) {
                                        suitableRefs.add(numLocalRefs, ref);
                                        numLocalRefs++; //increment number of lo
cal refs in list
                                        //if ref lives in an adjacent area, add
ref in position after last adjacent ref
                                } else if ((home == Referee.CENTRAL || matchLoc
== Referee.CENTRAL) && refWillTravel) {
                                        suitableRefs.add(numLocalRefs + numAdjRe
fs, ref);
                                        numAdjRefs++; //increment number of adja
cent refs in list
                                        //if ref lives in a non-adjacent area, a
dd ref to end of list
                                } else if (refWillTravel) {
                                        suitableRefs.add(ref);
                                }
                        }
                }
                return suitableRefs;
        }

        public String getRefsOutText() {
                StringBuilder refsOutBuilder = new StringBuilder();

                for(Referee ref : refList) {
                        refsOutBuilder.append(ref.getRefLine());
                }
                return refsOutBuilder.toString();
        }

        /**
         * Returns an iterator over elements of type Referee.
         * @return an Iterator.
         */
        public Iterator<Referee> iterator() {
                return refList.iterator();
        }
}
```

```
public class Referee implements Comparable<Referee> {

        private final String refID, fName, lName;
        private String qualificationType;
        private int homeArea, qualificationLevel;
        private final boolean[] travelInfo;
        private int numAllocations;
        //boolean to keep track if ref has been allocated to a match within the
program (to prevent deletion of ref if so)
        private boolean allocated = false;

        private static final int NUM_AREAS = 3;
        private static final String[] AREAS = {"North", "Central", "South"};
        //int constants to represent each area
        public static final int NORTH = 0;
        public static final int CENTRAL = 1;
        public static final int SOUTH = 2;

        /**
         * Referee constructor - takes line read from input file, tokenises and
sets
         * instance variables accordingly
         * @param refInfo the line read from the input file
         */
        public Referee(String refInfo) {
                String [] infoTokens = refInfo.split("[ ]+");

                refID = infoTokens[0];
                fName = infoTokens[1];
                lName = infoTokens[2];

                qualificationType = infoTokens[3].substring(0,3);
                qualificationLevel = Integer.parseInt(infoTokens[3].substring(3)
);

                numAllocations = Integer.parseInt(infoTokens[4].trim());

                setHomeArea(infoTokens[5]);

                travelInfo = new boolean [NUM_AREAS];
                setTravelInfo(infoTokens[6]);
        }

        //mutator methods

        /**
         * Mutator method for number of match allocations
         * @param numAlloc the number of allocations
         */
        public void setNumAllocs(int numAlloc)
        {
                 numAllocations = numAlloc;
        }

        /**
         * mutator for qualification type
         * @param qual the type of qualification
         */
        public void setQualificationType(String qual) {
                qualificationType = qual;
        }
```

```
	/**
	 * mutator for qualification level
	 * @param level the level of the ref’s qualification
	 */
	public void setQualificationLevel(int level)
	{
		qualificationLevel = level;
	}

	/**
	 * mutator for the ref’s home area. takes a string and sets homeArea as 
the corresponding
	 * int constant
	 * @param home the ref’s home area as a string
	 */
	public void setHomeArea(String home) {
		if(home.equals("North"))
			homeArea = NORTH;
		else if(home.equals("Central"))
			homeArea = CENTRAL;
		else
			homeArea = SOUTH;
	}

	/**
	 * mutator for ref’s travel info. takes a string indicating travel prefe
rences e.g. "YYN"
	 * checks each character − if ’Y’ sets corresponding position of travelI
nfo array to true
	 * if ’N’ sets to false
	 * @param travelStr string indicating ref’s travel preferences
	 */
	public void setTravelInfo(String travelStr) {
		for(int i=0; i < travelInfo.length; i++)
			travelInfo[i] = (travelStr.charAt(i) == ’Y’);
	}

	/**
	 * called whenever ref is allocated to a match
	 * besides incrementing numAllocs, checks if ref has been allocated to a
 match yet;
	 * if not, sets allocated to true
	 */
	public void incrementAllocs() {
		numAllocations++;

		if(!allocated)
			allocated = true;
	}

	//accessor methods

	/**
	 * gets info on whether ref will travel to a certain area. takes in the 
area as an integer
	 * and returns the boolean value of the travelInfo array at that positio
n of the array
	 * @param area the area as an integer constant
	 * @return true if ref is willing to travel to the area, false if not
	 */
	public boolean getTravelInfo(int area) {
```

```
		return travelInfo[area];
	}

	private String getTravelString() {
		String travelString = "";
		for (boolean aTravelInfo : travelInfo) {
			if (aTravelInfo)
				travelString += "Y";
			else
				travelString += "N";
		}
		return travelString;
	}

	/**
	 * accessor for refId
	 * @return the refID
	 */
	public String getRefID() {
		return refID;
	}

	/**
	 * accessor for ref’s first name
	 * @return the ref’s first name
	 */
	public String getFName() {
		return fName;
	}

	/**
	 * accessor the ref’s last name
	 * @return the ref’s last name
	 */
	public String getLName() {
		return lName;
	}

	/**
	 * accessor for the ref’s qualification type
	 * @return the the ref’s qualification type
	 */
	public String getQualificationType(){
		return qualificationType;
	}

	/**
	 * accessor for the ref’s qualification level
	 * @return the ref’s qualification level
	 */
	public int getQualificationLevel()
	{
		return qualificationLevel;
	}

	/**
	 * checks if the ref is qualified for senior matches
	 * @return true if ref is qualified, false if not
	 */
	public boolean checkIfQualified() {
		//ref is qualified if qualificationLevel is greater than 1
		return (qualificationLevel > 1);
```

```
        }

        /**
         * accessor for number of allocations
         * @return the ref’s number of allocations
         */
        public int getNumAllocs() {
                return numAllocations;
        }

        /**
         * accessor for the ref’s home area as an int
         * @return the ref’s home area as an int
         */
        public int getHomeArea() {
                return homeArea;
        }

        /**
         * accessor for the ref’s home area as a string
         * @return the ref’s home area as a string
         */
        public String getHomeString() {
                return AREAS[homeArea];
        }

        /**
         * checks if the ref has been allocated to a match in Hibernia
         * @return true if they have, false if not
         */
        public boolean isAllocated() {
                return allocated;
        }

        public String getRefLine() {
                String home = getHomeString();
                String travel = getTravelString();

                return String.format("%s %s %s %s%d %d %s %s %n", refID, fName, lNa
me, qualificationType, qualificationLevel, numAllocations, home, travel);
        }

        /**
         * compareTo method to be used in getSuitableRefs() in RefList.
         * compares Referee objects based on number of allocations
         * @param other the referee to be compared
         * @return the result of the comparison
         */
        public int compareTo(Referee other) {
                int thisAllocs = this.getNumAllocs();
                int otherAllocs = other.getNumAllocs();
                if(thisAllocs < otherAllocs)
                        return −1;
                else if(thisAllocs > otherAllocs)
                        return 1;
                else
                        return 0;
        }
}
```

```
/**
 * The main class
 */
public class TeamProject {
    public static void main(String[] args) {
        // Create the main GUI object and set it visible
                MainGUI main = new MainGUI();
                main.setVisible(true);
    }
}
```