### Logos - London Metropolitan UniversityCourse Submission Cover Sheet

### Module: CS4001 Programming

### Assignment no: 003

### Weighting: 60% of module mark

### Deadline: Friday 7th of May 2025

Module Leaders: Dr Sandra Fernando & Dr Sahar Al-Sudani

Student ID: 24012278

Github repository: https://github.com/KHZ0044/CS4001-CW-repository

Please note that there are specific regulations concerning **the use of AI tools and Academic Misconduct**. Below are extracts from these regulations. By signing, you acknowledge that you have read and understood these extracts.

(signature:) Khuseyn Alikovitch Zarmayev Date: 25/04/2025

This header sheet should be attached to the work you submit.

Academic Integrity means being honest in your academic work and your studies and making sure that you acknowledge the work of others and giving credit where you have used other people's ideas as part of presenting your arguments. Your assessment submissions must therefore always be entirely your own work, based on your own learning and appropriately referenced including how you have used Generative AI. The University regards the use of Generative AI applications by students to deceive to gain unfair advantage as **academic misconduct**. This usage includes:

* **Plagiarism**, where AI tools are used to generate output and ideas that are presented or submitted as if they were the student's own work, without proper citation or references.
* Where a complete assignment is created using Generative AI and represented as a student's own work, this will be regarded as contract cheating in the same way as commissioning an 'Essay Mill' or other third party to complete your work. Further information can be found on : [Guidance on the use of Artificial Intelligence.](https://student.londonmet.ac.uk/your-studies/student-administration/guidance-on-the-use-of-artificial-intelligence/)

**Academic misconduct:** The University takes academic misconduct very seriously and seeks at all times to rigorously protect its academic standards. Plagiarism, collusion and other forms of cheating constitute academic misconduct, for which there is an explicit range of graduated penalties depending on the particular type of academic misconduct. The penalties that can be applied if academic misconduct is substantiated range from a reprimand to expulsion in very serious cases and for repeated instances of misconduct. You are also responsible for ensuring that all work submitted is your own and that it is appropriately referenced. The University does not tolerate cheating of any kind. You are strongly advised to familiarise yourself with the Academic Misconduct Policy and Procedure ([Academic Misconduct](https://student.londonmet.ac.uk/your-studies/student-administration/rules-and-regulations/academic-misconduct/)), which list a range of categories of academic misconduct and associated penalties, covering instances of academic misconduct (plagiarism, collusion, exam cheating).

# 1. Introduction

In this report, I’m excited to share the design and implementation of a Recruitment System that I developed using Java, specifically with Java Swing as my IDE through BlueJ. This application is designed to manage both full-time and part-time staff, featuring a user-friendly graphical user interface (GUI). I built it on an object-oriented framework, keeping in mind the key principles of inheritance, encapsulation, and modularity. The system offers various functions, such as adding staff, updating salary and shift details, terminating part-time employees when necessary, displaying all staff members, and resetting the interface. Throughout the design process, I aimed to create a system that ensures clarity in handling user input while also maintaining code that’s easy to manage. I made sure to validate user inputs and apply appropriate exception handling techniques. This report will delve into the methods used, the logic behind the user interface, and provide practical evidence of testing for system implementation along with effective error management.

# 2. Class Diagrams

The project consists of four main classes: StaffHire, FullTimeStaffHire, PartTimeStaffHire, and RecruitmentSystem.

• StaffHire (Superclass):  
 - Contains attributes common to all staff types: vacancyNumber, designation, jobType, staffName, joiningDate, qualification, appointedBy, and joined.  
 - Provides basic accessor and mutator methods, as well as a display method.

• FullTimeStaffHire (Subclass of StaffHire):  
 - Adds attributes: salary and weeklyFractionalHours.  
 - Implements logic to restrict salary modification only after staff have joined.  
 - Overrides the display method to include additional attributes.

• PartTimeStaffHire (Subclass of StaffHire):  
 - Adds attributes: workingHour, wagesPerHour, shifts, and terminated.  
 - Includes methods to update shifts and terminate the staff member.  
 - Calculates and displays income per day.

• RecruitmentSystem (GUI Controller):  
 - Handles user interaction using Java Swing.  
 - Stores all staff objects in an ArrayList<StaffHire>.  
 - Contains logic for all button actions, data validation, and user feedback dialogs.

# 3. Method Descriptions

Each class includes methods appropriate to its role. In StaffHire, accessor and mutator methods manage the encapsulated data. The display() method is common to all classes and outputs relevant information.

In FullTimeStaffHire:  
• setSalary(double salary): Only updates salary if joined == true.  
• setWeeklyFractionalHours(int hours): Updates working hours.  
• display(): Extends parent display method with salary and hours.

In PartTimeStaffHire:  
• setShifts(String shift): Updates shift only if joined is true.  
• terminateStaff(): Clears personal details and marks staff as terminated.  
• display(): Extends parent display and calculates income per day.  
In RecruitmentSystem:  
• actionPerformed(ActionEvent e): Handles all GUI events.  
• main(): Entry point launching the application.

# 4. Pseudocode for Button Actions

Add Full-Time Staff:  
 Read input fields  
 Create FullTimeStaffHire object  
 Add to staffList  
 Show confirmation dialog

Add Part-Time Staff:  
 Read input fields  
 Create PartTimeStaffHire object  
 Add to staffList  
 Show confirmation dialog

Set Salary:  
 Get vacancy number and new salary  
 Loop staffList to find matching FullTimeStaffHire  
 Call setSalary()

Set Shifts:  
 Get vacancy number and new shift  
 Loop staffList to find matching PartTimeStaffHire  
 Call setShifts()

Terminate Staff:  
 Get vacancy number  
 Find matching PartTimeStaffHire  
 Call terminateStaff()

Display Staff:  
 Loop through staffList  
 Call display() for each

# 5. Input Validation (Try/Catch)

Robust input validation was achieved using try/catch blocks around parsing functions such as Integer.parseInt() and Double.parseDouble(). This prevents the program from crashing due to invalid numeric entries. Informative messages are shown using JOptionPane when user inputs are invalid or missing. For example, if a user enters "abc" in a numeric field, the catch block detects the error and notifies the user without stopping the application.

# 6. Button Functionality Overview

• Add Full-Time Staff: Adds a new full-time staff member to the list using the provided input fields.  
• Add Part-Time Staff: Adds a new part-time staff member and sets their working hours, wage, and shift.  
• Set Salary: Updates salary for an existing full-time staff member, only if they have joined.  
• Set Shifts: Updates the shift of an existing part-time staff member.  
• Terminate Staff: Marks a part-time staff member as terminated and clears their data.  
• Display Staff: Outputs all staff details in the BlueJ terminal using the display() method.  
• Clear Fields: Resets all text fields in the interface.

# 7. Screenshot Evidence of Testing

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1: GUI successfully launched in BlueJ.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2: Adding Full-Time Staff (fields filled).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Confirmation after adding Full-Time Staff.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Adding Part-Time Staff (fields filled).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5: Confirmation after adding Part-Time Staff.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6: Updating Full-Time Staff salary.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7: Updating Part-Time Staff shift.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: Staff details before termination.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Confirmation of Part-Time Staff termination.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10: Terminal output from 'Display Staff'.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11: Error message for invalid salary input.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12: Error message for empty field input.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13: Fields filled before clicking 'Clear Fields'.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14: All fields cleared after clicking 'Clear Fields'.

# 8. Error Detection and Correction

1. Syntax Error – You might run into a duplicate variable declaration issue (like declaring displayButton twice). This can be fixed by making sure each component is declared just once.

2. Runtime Error – A NumberFormatException can pop up when you're trying to parse salary and hours. The solution? Wrap all your input parsing in try/catch blocks to handle any hiccups.

3. Logical Error – The setSalary method was allowing salary changes even before joining. I fixed this by checking if the user has joined before permitting any changes.

# 9. Conclusion and Reflection

The Recruitment System was successfully built using object-oriented programming and Java Swing. It allows for the addition and management of both full-time and part-time staff, demonstrating the principles of inheritance and encapsulation. The user interface enhances the overall experience and features solid input validation and error handling. This coursework really helped me grasp the ins and outs of GUI applications, class relationships, and exception handling. I tackled challenges like interface layout, error detection, and event management by employing modular code and structured logic. Looking ahead, there are some exciting improvements to consider, such as saving staff data to a file, adding a search feature, and upgrading the interface with dropdown menus and date pickers.

# Appendix: Full Java Code Listings

## StaffHire.java

public class StaffHire {  
 private int vacancyNumber;  
 private String designation;  
 private String jobType;  
 private String staffName;  
 private String joiningDate;  
 private String qualification;  
 private String appointedBy;  
 private boolean joined;  
  
 public StaffHire(int vacancyNumber, String designation, String jobType, String staffName,  
 String joiningDate, String qualification, String appointedBy, boolean joined) {  
 this.vacancyNumber = vacancyNumber;  
 this.designation = designation;  
 this.jobType = jobType;  
 this.staffName = staffName;  
 this.joiningDate = joiningDate;  
 this.qualification = qualification;  
 this.appointedBy = appointedBy;  
 this.joined = joined;  
 }  
  
 public int getVacancyNumber() {  
 return vacancyNumber;  
 }  
  
 public String getDesignation() {  
 return designation;  
 }  
  
 public String getJobType() {  
 return jobType;  
 }  
  
 public String getStaffName() {  
 return staffName;  
 }  
  
 public String getJoiningDate() {  
 return joiningDate;  
 }  
  
 public String getQualification() {  
 return qualification;  
 }  
  
 public String getAppointedBy() {  
 return appointedBy;  
 }  
  
 public boolean isJoined() {  
 return joined;  
 }  
  
 public void setVacancyNumber(int vacancyNumber) {  
 this.vacancyNumber = vacancyNumber;  
 }  
  
 public void setDesignation(String designation) {  
 this.designation = designation;  
 }  
  
 public void setJobType(String jobType) {  
 this.jobType = jobType;  
 }  
  
 public void setStaffName(String staffName) {  
 this.staffName = staffName;  
 }  
  
 public void setJoiningDate(String joiningDate) {  
 this.joiningDate = joiningDate;  
 }  
  
 public void setQualification(String qualification) {  
 this.qualification = qualification;  
 }  
  
 public void setAppointedBy(String appointedBy) {  
 this.appointedBy = appointedBy;  
 }  
  
 public void setJoined(boolean joined) {  
 this.joined = joined;  
 }  
  
 public void display() {  
 System.out.println("Vacancy Number: " + vacancyNumber);  
 System.out.println("Designation: " + designation);  
 System.out.println("Job Type: " + jobType);  
 System.out.println("Staff Name: " + staffName);  
 System.out.println("Joining Date: " + joiningDate);  
 System.out.println("Qualification: " + qualification);  
 System.out.println("Appointed By: " + appointedBy);  
 System.out.println("Joined: " + joined);  
 }  
  
 public void setJoinedStatus(boolean joined) {  
 this.joined = joined;  
 }  
}

## FullTimeStaffHire.java

public class FullTimeStaffHire extends StaffHire {  
 private double salary;  
 private int weeklyFractionalHours;  
  
 // Constructor  
 public FullTimeStaffHire(int vacancyNumber, String designation, String jobType, String staffName,  
 String joiningDate, String qualification, String appointedBy, boolean joined,  
 double salary, int weeklyFractionalHours) {  
 super(vacancyNumber, designation, jobType, staffName, joiningDate, qualification, appointedBy, joined);  
 this.salary = salary;  
 this.weeklyFractionalHours = weeklyFractionalHours;  
 }  
  
 // Getters  
 public double getSalary() {  
 return salary;  
 }  
  
 public int getWeeklyFractionalHours() {  
 return weeklyFractionalHours;  
 }  
  
 // Setters  
 public void setSalary(double salary) {  
 if (isJoined()) {  
 this.salary = salary;  
 } else {  
 System.out.println("Salary cannot be set — staff not yet joined.");  
 }  
 }  
  
 public void setWeeklyFractionalHours(int weeklyFractionalHours) {  
 this.weeklyFractionalHours = weeklyFractionalHours;  
 }  
  
 // Display  
 public void display() {  
 super.display(); // call the display method of StaffHire  
 System.out.println("Salary: " + salary);  
 System.out.println("Weekly Fractional Hours: " + weeklyFractionalHours);  
 }  
}

## PartTimeStaffHire.java

public class PartTimeStaffHire extends StaffHire {  
 private int workingHour;  
 private double wagesPerHour;  
 private String shifts;  
 private boolean terminated;  
  
 // Constructor  
 public PartTimeStaffHire(int vacancyNumber, String designation, String jobType, String staffName,  
 String joiningDate, String qualification, String appointedBy, boolean joined,  
 int workingHour, double wagesPerHour, String shifts) {  
 super(vacancyNumber, designation, jobType, staffName, joiningDate, qualification, appointedBy, joined);  
 this.workingHour = workingHour;  
 this.wagesPerHour = wagesPerHour;  
 this.shifts = shifts;  
 this.terminated = false; // initially false  
 }  
  
 // Getters  
 public int getWorkingHour() {  
 return workingHour;  
 }  
  
 public double getWagesPerHour() {  
 return wagesPerHour;  
 }  
  
 public String getShifts() {  
 return shifts;  
 }  
  
 public boolean isTerminated() {  
 return terminated;  
 }  
  
 // Setters  
 public void setWorkingHour(int workingHour) {  
 this.workingHour = workingHour;  
 }  
  
 public void setWagesPerHour(double wagesPerHour) {  
 this.wagesPerHour = wagesPerHour;  
 }  
  
 public void setShifts(String shifts) {  
 if (isJoined()) {  
 this.shifts = shifts;  
 } else {  
 System.out.println("Cannot change shift — staff has not joined.");  
 }  
 }  
  
 // Terminate Staff  
 public void terminateStaff() {  
 if (terminated) {  
 System.out.println("Staff is already terminated.");  
 } else {  
 setStaffName("");  
 setJoiningDate("");  
 setQualification("");  
 setAppointedBy("");  
 setJoined(false);  
 terminated = true;  
 System.out.println("Staff has been terminated.");  
 }  
 }  
  
 // Display Method  
 public void display() {  
 super.display(); // display common fields  
 System.out.println("Working Hour: " + workingHour);  
 System.out.println("Wages Per Hour: " + wagesPerHour);  
 System.out.println("Shifts: " + shifts);  
 System.out.println("Terminated: " + terminated);  
 System.out.println("Income Per Day: " + (wagesPerHour \* workingHour));  
 }  
}

## RecruitmentSystem.java

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.util.ArrayList;  
  
public class RecruitmentSystem extends JFrame implements ActionListener {  
 // Input fields  
 private JTextField vacancyField, designationField, jobTypeField, staffNameField, joiningDateField,  
 qualificationField, appointedByField, salaryField, hoursField,  
 workingHoursField, wagesPerHourField, shiftsField,  
 vacancyForSalaryField, newSalaryField,  
 vacancyForShiftsField, newShiftsField,  
 terminateVacancyField;  
  
 // Buttons  
 private JButton addFullTimeButton, displayButton, addPartTimeButton,  
 setSalaryButton, setShiftsButton, terminateButton, clearButton;  
  
 private ArrayList<StaffHire> staffList;  
  
 public RecruitmentSystem() {  
 setTitle("Recruitment System");  
 setSize(600, 850);  
 setLayout(null);  
 setDefaultCloseOperation(EXIT\_ON\_CLOSE);  
  
 staffList = new ArrayList<>();  
  
 // Labels & Inputs  
 JLabel vacancyLabel = new JLabel("Vacancy Number:");  
 vacancyLabel.setBounds(20, 20, 150, 25);  
 add(vacancyLabel);  
  
 vacancyField = new JTextField();  
 vacancyField.setBounds(180, 20, 200, 25);  
 add(vacancyField);  
  
 JLabel designationLabel = new JLabel("Designation:");  
 designationLabel.setBounds(20, 60, 150, 25);  
 add(designationLabel);  
  
 designationField = new JTextField();  
 designationField.setBounds(180, 60, 200, 25);  
 add(designationField);  
  
 JLabel jobTypeLabel = new JLabel("Job Type:");  
 jobTypeLabel.setBounds(20, 100, 150, 25);  
 add(jobTypeLabel);  
  
 jobTypeField = new JTextField();  
 jobTypeField.setBounds(180, 100, 200, 25);  
 add(jobTypeField);  
  
 JLabel staffNameLabel = new JLabel("Staff Name:");  
 staffNameLabel.setBounds(20, 140, 150, 25);  
 add(staffNameLabel);  
  
 staffNameField = new JTextField();  
 staffNameField.setBounds(180, 140, 200, 25);  
 add(staffNameField);  
  
 JLabel joiningDateLabel = new JLabel("Joining Date:");  
 joiningDateLabel.setBounds(20, 180, 150, 25);  
 add(joiningDateLabel);  
  
 joiningDateField = new JTextField();  
 joiningDateField.setBounds(180, 180, 200, 25);  
 add(joiningDateField);  
  
 JLabel qualificationLabel = new JLabel("Qualification:");  
 qualificationLabel.setBounds(20, 220, 150, 25);  
 add(qualificationLabel);  
  
 qualificationField = new JTextField();  
 qualificationField.setBounds(180, 220, 200, 25);  
 add(qualificationField);  
  
 JLabel appointedByLabel = new JLabel("Appointed By:");  
 appointedByLabel.setBounds(20, 260, 150, 25);  
 add(appointedByLabel);  
  
 appointedByField = new JTextField();  
 appointedByField.setBounds(180, 260, 200, 25);  
 add(appointedByField);  
  
 JLabel salaryLabel = new JLabel("Salary:");  
 salaryLabel.setBounds(20, 300, 150, 25);  
 add(salaryLabel);  
  
 salaryField = new JTextField();  
 salaryField.setBounds(180, 300, 200, 25);  
 add(salaryField);  
  
 JLabel hoursLabel = new JLabel("Weekly Hours:");  
 hoursLabel.setBounds(20, 340, 150, 25);  
 add(hoursLabel);  
  
 hoursField = new JTextField();  
 hoursField.setBounds(180, 340, 200, 25);  
 add(hoursField);  
  
 JLabel workingHoursLabel = new JLabel("Working Hours:");  
 workingHoursLabel.setBounds(20, 380, 150, 25);  
 add(workingHoursLabel);  
  
 workingHoursField = new JTextField();  
 workingHoursField.setBounds(180, 380, 200, 25);  
 add(workingHoursField);  
  
 JLabel wagesPerHourLabel = new JLabel("Wages Per Hour:");  
 wagesPerHourLabel.setBounds(20, 420, 150, 25);  
 add(wagesPerHourLabel);  
  
 wagesPerHourField = new JTextField();  
 wagesPerHourField.setBounds(180, 420, 200, 25);  
 add(wagesPerHourField);  
  
 JLabel shiftsLabel = new JLabel("Shifts:");  
 shiftsLabel.setBounds(20, 460, 150, 25);  
 add(shiftsLabel);  
  
 shiftsField = new JTextField();  
 shiftsField.setBounds(180, 460, 200, 25);  
 add(shiftsField);  
  
 // Buttons  
 addFullTimeButton = new JButton("Add Full-Time Staff");  
 addFullTimeButton.setBounds(400, 20, 170, 30);  
 addFullTimeButton.addActionListener(this);  
 add(addFullTimeButton);  
  
 addPartTimeButton = new JButton("Add Part-Time Staff");  
 addPartTimeButton.setBounds(400, 60, 170, 30);  
 addPartTimeButton.addActionListener(this);  
 add(addPartTimeButton);  
  
 displayButton = new JButton("Display Staff");  
 displayButton.setBounds(400, 100, 170, 30);  
 displayButton.addActionListener(this);  
 add(displayButton);  
   
 displayButton.addActionListener(this);  
 add(displayButton);  
   
 JLabel vacancySalaryLabel = new JLabel("Vacancy # (Set Salary):");  
 vacancySalaryLabel.setBounds(20, 520, 170, 25);  
 add(vacancySalaryLabel);  
  
 vacancyForSalaryField = new JTextField();  
 vacancyForSalaryField.setBounds(200, 520, 180, 25);  
 add(vacancyForSalaryField);  
  
 JLabel newSalaryLabel = new JLabel("New Salary:");  
 newSalaryLabel.setBounds(20, 560, 170, 25);  
 add(newSalaryLabel);  
  
 newSalaryField = new JTextField();  
 newSalaryField.setBounds(200, 560, 180, 25);  
 add(newSalaryField);  
  
 setSalaryButton = new JButton("Set Salary");  
 setSalaryButton.setBounds(400, 140, 170, 30);  
 setSalaryButton.addActionListener(this);  
 add(setSalaryButton);  
  
 JLabel vacancyShiftLabel = new JLabel("Vacancy # (Set Shift):");  
 vacancyShiftLabel.setBounds(20, 600, 170, 25);  
 add(vacancyShiftLabel);  
  
 vacancyForShiftsField = new JTextField();  
 vacancyForShiftsField.setBounds(200, 600, 180, 25);  
 add(vacancyForShiftsField);  
  
 JLabel newShiftsLabel = new JLabel("New Shifts:");  
 newShiftsLabel.setBounds(20, 640, 170, 25);  
 add(newShiftsLabel);  
  
 newShiftsField = new JTextField();  
 newShiftsField.setBounds(200, 640, 180, 25);  
 add(newShiftsField);  
  
 setShiftsButton = new JButton("Set Shifts");  
 setShiftsButton.setBounds(400, 180, 170, 30);  
 setShiftsButton.addActionListener(this);  
 add(setShiftsButton);  
  
 JLabel terminateVacancyLabel = new JLabel("Vacancy # to Terminate:");  
 terminateVacancyLabel.setBounds(20, 680, 170, 25);  
 add(terminateVacancyLabel);  
  
 terminateVacancyField = new JTextField();  
 terminateVacancyField.setBounds(200, 680, 180, 25);  
 add(terminateVacancyField);  
  
 terminateButton = new JButton("Terminate Staff");  
 terminateButton.setBounds(400, 220, 170, 30);  
 terminateButton.addActionListener(this);  
 add(terminateButton);  
  
 clearButton = new JButton("Clear Fields");  
 clearButton.setBounds(400, 260, 170, 30);  
 clearButton.addActionListener(this);  
 add(clearButton);  
  
 setVisible(true);  
 }  
  
 public void actionPerformed(ActionEvent e) {  
 try {  
 if (e.getSource() == addFullTimeButton) {  
 int vacancyNumber = Integer.parseInt(vacancyField.getText());  
 String designation = designationField.getText();  
 String jobType = jobTypeField.getText();  
 String staffName = staffNameField.getText();  
 String joiningDate = joiningDateField.getText();  
 String qualification = qualificationField.getText();  
 String appointedBy = appointedByField.getText();  
 double salary = Double.parseDouble(salaryField.getText());  
 int hours = Integer.parseInt(hoursField.getText());  
  
 FullTimeStaffHire staff = new FullTimeStaffHire(vacancyNumber, designation, jobType, staffName,  
 joiningDate, qualification, appointedBy, true, salary, hours);  
 staffList.add(staff);  
 JOptionPane.showMessageDialog(this, "Full-Time Staff Added!");  
 }  
  
 if (e.getSource() == addPartTimeButton) {  
 int vacancyNumber = Integer.parseInt(vacancyField.getText());  
 String designation = designationField.getText();  
 String jobType = jobTypeField.getText();  
 String staffName = staffNameField.getText();  
 String joiningDate = joiningDateField.getText();  
 String qualification = qualificationField.getText();  
 String appointedBy = appointedByField.getText();  
 int workingHours = Integer.parseInt(workingHoursField.getText());  
 double wagesPerHour = Double.parseDouble(wagesPerHourField.getText());  
 String shifts = shiftsField.getText();  
  
 PartTimeStaffHire staff = new PartTimeStaffHire(vacancyNumber, designation, jobType, staffName,  
 joiningDate, qualification, appointedBy, true, workingHours, wagesPerHour, shifts);  
 staffList.add(staff);  
 JOptionPane.showMessageDialog(this, "Part-Time Staff Added!");  
 }  
  
 if (e.getSource() == displayButton) {  
 for (StaffHire s : staffList) {  
 s.display();  
 }  
 }  
  
 if (e.getSource() == setSalaryButton) {  
 int vacancy = Integer.parseInt(vacancyForSalaryField.getText());  
 double newSalary = Double.parseDouble(newSalaryField.getText());  
 boolean found = false;  
 for (StaffHire s : staffList) {  
 if (s instanceof FullTimeStaffHire && s.getVacancyNumber() == vacancy) {  
 ((FullTimeStaffHire) s).setSalary(newSalary);  
 JOptionPane.showMessageDialog(this, "Salary Updated.");  
 found = true;  
 break;  
 }  
 }  
 if (!found)  
 JOptionPane.showMessageDialog(this, "Full-Time Staff not found.");  
 }  
  
 if (e.getSource() == setShiftsButton) {  
 int vacancy = Integer.parseInt(vacancyForShiftsField.getText());  
 String newShift = newShiftsField.getText();  
 boolean found = false;  
 for (StaffHire s : staffList) {  
 if (s instanceof PartTimeStaffHire && s.getVacancyNumber() == vacancy) {  
 ((PartTimeStaffHire) s).setShifts(newShift);  
 JOptionPane.showMessageDialog(this, "Shifts Updated.");  
 found = true;  
 break;  
 }  
 }  
 if (!found)  
 JOptionPane.showMessageDialog(this, "Part-Time Staff not found.");  
 }  
  
 if (e.getSource() == terminateButton) {  
 int vacancy = Integer.parseInt(terminateVacancyField.getText());  
 boolean found = false;  
 for (StaffHire s : staffList) {  
 if (s instanceof PartTimeStaffHire && s.getVacancyNumber() == vacancy) {  
 ((PartTimeStaffHire) s).terminateStaff();  
 JOptionPane.showMessageDialog(this, "Part-Time Staff Terminated.");  
 found = true;  
 break;  
 }  
 }  
 if (!found)  
 JOptionPane.showMessageDialog(this, "Part-Time Staff not found.");  
 }  
  
 if (e.getSource() == clearButton) {  
 vacancyField.setText("");  
 designationField.setText("");  
 jobTypeField.setText("");  
 staffNameField.setText("");  
 joiningDateField.setText("");  
 qualificationField.setText("");  
 appointedByField.setText("");  
 salaryField.setText("");  
 hoursField.setText("");  
 workingHoursField.setText("");  
 wagesPerHourField.setText("");  
 shiftsField.setText("");  
 vacancyForSalaryField.setText("");  
 newSalaryField.setText("");  
 vacancyForShiftsField.setText("");  
 newShiftsField.setText("");  
 terminateVacancyField.setText("");  
 }  
 } catch (Exception ex) {  
 JOptionPane.showMessageDialog(this, "Invalid Input: " + ex.getMessage());  
 }  
 }  
  
 public static void main(String[] args) {  
 new RecruitmentSystem();  
 }  
}