CC4001 Programming

Year -01 (StaffHire.java)
BSc (Hons) Computer Science

AAVASH SAPKOTA
ID- 22081679

Course Submission Cover Sheet

LONDON METROPOLITAN UNIVERSITY

Module: CC4001 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:22081679

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:) aavash sapkota Date: 07-05-25

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.

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), which list a range of categories of academic misconduct and associated penalties, covering instances of academic misconduct (plagiarism, collusion, exam cheating).

Contents

1. Introduction

This report is designed to show the design and implementation of the staff hire process in Java that manages the staff hire for full-time and part-time positions. The system comprises four different classes: StaffHire (superclass), FullTimeStaffHire (subclass), PartTimeStaffHire(subclass), and StaffHireGUI (GUI and arraylist logic) as shown in the E-R diagram below. This assignment will showcase a deep understanding of Java related to the array, use of GitHub, ER diagram, and the relationship between different classes.

2. Aim

The program aims to add the information to the particular class, such as Part-Time, Full-Time, while along with that set the salary, and setting the working shift. all in all, basic understanding of how the hiring system works in the real-life world. This report will break down the system to understand the system by the readers and record the process and test the method. This report will also describe each class with attributes that as used in the class. While explaining the attributes, we also go through the pseudocode to guide me in writing and have a plan to write the code.

3. GitHub

The link provide is the repository of the program code in the java.

https://github.com/aavash15/Staff-hire.java/

4. Class Diagram

StaffHire (Superclass

FullTimeStaffHire (Inherits StaffHire)

PartTimeStaffHire (Inherits StaffHire)

StaffHireGUI (Handles GUI and ArrayList logic)

Figure –1.1: Class Diagram showing the relation between different classes.

Figure –1.1: Entity Relationship (ER) Diagram showing the relation between different classes.

The diagram shows the relationship between StaffHire, FullTimeStaffHire, PartTimeStaffHire, and StaffHireGUI. The primary key is the ID in each class. The diagram helps us to provide a visual representation in the system's architecture.

5. Method Descriptions

5.1 StaffHire class methods

5.1.1. Constructor : It initializes the class new staffhire object with the details and staff information.

"Public StaffHire(int vacancyNumber, String designation, String jobType, String staffName,

String joiningDate, String qualification, String appointedBy, boolean joined)"

```
/**
 * Constructor for objects of class StaffHire
 * Initializes the staff member's details
 */
public StaffHire(int vacancyNumber, String designation, String jobType, String staffName,
                 String joiningDate, String qualification, String appointedBy, boolean joined)
```

Figure -2: Construction of the staff hire

This code allows us to set the values in the different variable to use them later for the staff hire process.

5.1.2: Accessor Methods :

```
{
    this.vacancyNumber = vacancyNumber; // Set vacancy number
    this.designation = designation; // Set designation
    this.jobType = jobType; // Set job type
    this.staffName = staffName; // Set staff name
    this.joiningDate = joiningDate; // Set joining date
    this.qualification = qualification; // Set qualification
    this.appointedBy = appointedBy; // Set appointed by
    this.joined = joined; // Set joined status
}
```

Figure -3: Accessor method of the staff hires

This code and the comment help class staff hire to access and modify the induvial value.

5.1.3: Display :

Outputs formatted information about the vacancy and staff, including vacancy number, designation, job type, staff name, joining date, qualification, appointed by, and joined status.

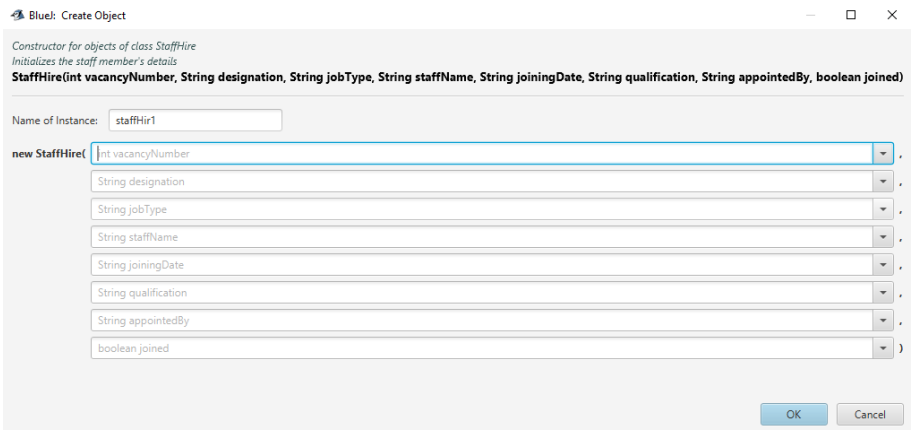

Figure -4: Display of staffHire

5.2 PartTimeStaffHire Class Methods

5.2.1. Constructor : It initializes the class new PartTimeStaffhire object with the details and staff information.

```
public class PartTimeStaffHire extends StaffHire
 {
    // Instance variables specific to part-time staff
    private int workingHour; // Number of working hours
    private double wagesPerHour; // Wages per hour
    private String shifts; // Working shifts (e.g., morning, evening)
    private boolean terminated; // Status indicating if the staff is terminated

    // Constructor for PartTimeStaffHire
    public PartTimeStaffHire(int vacancyNumber, String designation, String jobType, String staffName,
                             String joiningDate, String qualification, String appointedBy,
                             boolean joined, int workingHour, double wagesPerHour, String shifts)
```

Figure -5: Construction of the part timer staff hire

Unlink, class staff hire this class is the subclass of superclass staffhire therefore, it calls for the superclass and set the additional attributes.

5.2.2: Accessor Methods:

```
{
    // Call the constructor of the superclass (StaffHire) to initialize common fields
    super(vacancyNumber, designation, jobType, staffName, joiningDate, qualification, appointedBy, joined);
    this.workingHour = workingHour; // Set working hours
    this.wagesPerHour = wagesPerHour; // Set wages per hour
    this.shifts = shifts; // Set working shifts
    this.terminated = false; // Default to not terminated
}
```

Figure -6: Accessor method of the part time staff hires

Terminates a part-time staff member by clearing their details and updating their status.

Formatted: Centred

Formatted: Font:

Formatted: Font:

5.2.3: Display:

Overrides the parent class method to display part-time staff details, including working hours, wages, shifts, and calculates income per day.

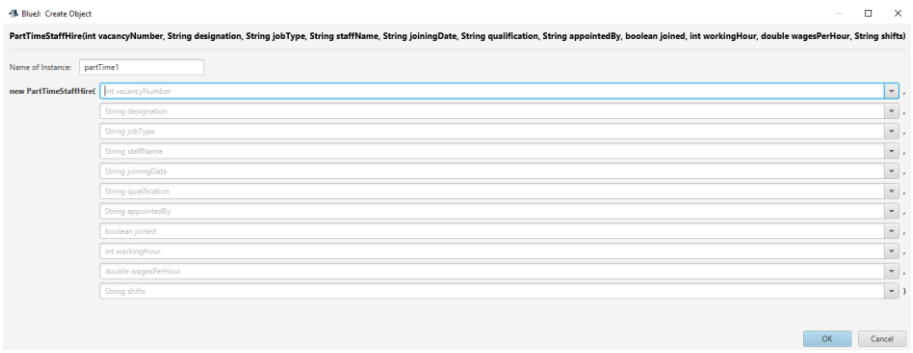

Figure -7: Display of the part time staff hire

5.3 FullTimeStaffHire Class Methods

5.3.1. Constructor: It initializes the class new PartTimeStaffhire object with the details and staff information.

Similarly to class parttime staff hire this class is the subclass of superclass staff hire therefore, it calls for the superclass and set the additional attributes.

```
public class FullTimeStaffHire extends StaffHire
{
    // Instance variables specific to full-time staff
    private double salary; // Salary of the full-time staff
    private int weeklyFractionalHours; // Weekly working hours in fractional format

    // Constructor for FullTimeStaffHire
    public FullTimeStaffHire(int vacancyNumber, String designation, String jobType, String staffName,
                             String joiningDate, String qualification, String appointedBy,
                             boolean joined, double salary, int weeklyFractionalHours)
                             {
```

Figure -8: Construction of the full time staff hire

5.3.2: Accessor Methods:

```
// Call the constructor of the superclass (StaffHire) to initialize common fields
super(vacancyNumber, designation, jobType, staffName, joiningDate, qualification, appointedBy, joine
this.salary = salary; // Set the salary
this.weeklyFractionalHours = weeklyFractionalHours; // Set weekly fractional hours
```

Figure -9: Accessor method of the full-time staff hire

This help to access and modify the value of the salary, etc.

5.3.3: Display:

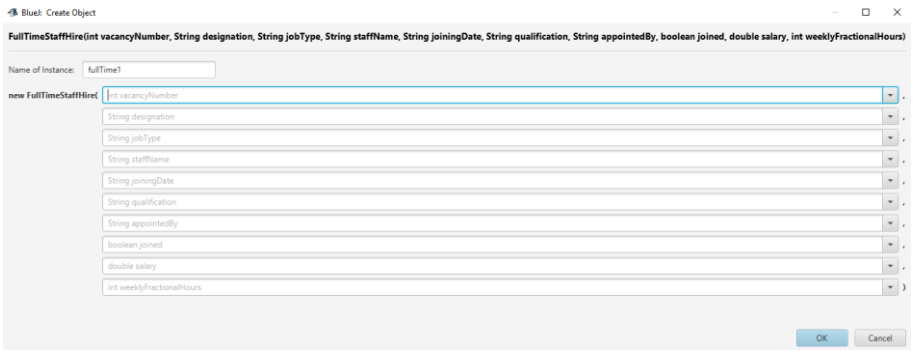

Figure -10: display of the full-time staff hire

Here user can see and the data for the staff hire and modify the value for the one.

6. Pseudocode for classes

6.1. Add Staff hire class

```
// This class keeps track of hiring info for a staff member
CLASS StaffHire

 // Stuff this class remembers about each staff hire:
 PRIVATE
  designation     // Job title, like "Teacher" or "Manager"
  jobType         // Type of job: permanent, contract, or temporary
  staffName       // Name of the person hired
  joiningDate     // When they started working
  qualification   // What qualifications they have
  appointedBy     // Who hired them
  vacancyNumber   // The job opening number
```

6.2. Add parttimeStaff hire class

```
// This class represents part-time staff hiring details, extending from StaffHire
CLASS PartTimeStaffHire EXTENDS StaffHire

 // Extra details specific to part-time staff
 PRIVATE
  workingHour      // How many hours they work in a day
  wagesPerHour     // How much they earn for each hour worked
  shifts           // What shift they work (morning, evening, etc.)
  terminated       // Is this staff member terminated? True or False

 // Constructor: sets up a new part-time staff member with all their details
```

```
  joined        // Have they actually joined yet? True or False

 // When we create a StaffHire, we need to give all this info at once
 FUNCTION StaffHire(vacancyNumber, designation, jobType, staffName, joiningDate,
                    qualification, appointedBy, joined)
   SET this.vacancyNumber = vacancyNumber
   SET this.designation = designation
   SET this.jobType = jobType
   SET this.staffName = staffName
   SET this.joiningDate = joiningDate
   SET this.qualification = qualification
   SET this.appointedBy = appointedBy
   SET this.joined = joined
 END FUNCTION

 // Methods to get info from the object or change it if needed:

 FUNCTION getVacancyNumber()
   RETURN vacancyNumber
 END FUNCTION

 FUNCTION setVacancyNumber(newNumber)
   SET this.vacancyNumber = newNumber
 END FUNCTION

 FUNCTION getDesignation()
   RETURN designation
 END FUNCTION

 FUNCTION setDesignation(newDesignation)
   SET this.designation = newDesignation
 END FUNCTION

 FUNCTION getJobType()
   RETURN jobType
 END FUNCTION

 FUNCTION setJobType(newJobType)
```

```
 FUNCTION PartTimeStaffHire(vacancyNumber, designation, jobType, staffName,
                            joiningDate, qualification, appointedBy,
                            joined, workingHour, wagesPerHour, shifts)
   // Call the parent class (StaffHire) constructor to set common details
   CALL super(vacancyNumber, designation, jobType, staffName, joiningDate,
              qualification, appointedBy, joined)

   // Now set the part-time specific details
   SET this.workingHour = workingHour // Set how many hours they work
   SET this.wagesPerHour = wagesPerHour // Set their pay rate
   SET this.shifts = shifts // Set their working shift
   SET this.terminated = false // By default, they are not terminated
 END FUNCTION

 // Getters: methods to access the private details of the part-time staff

 FUNCTION getWorkingHour()
   RETURN workingHour // Return the number of working hours
 END FUNCTION

 FUNCTION getWagesPerHour()
   RETURN wagesPerHour // Return the pay rate per hour
 END FUNCTION

 FUNCTION getShifts()
   RETURN shifts // Return the working shift
 END FUNCTION

 FUNCTION isTerminated()
   RETURN terminated // Check if the staff member is terminated
 END FUNCTION
```

```
  SET this.jobType = newJobType
 END FUNCTION

 FUNCTION getStaffName()
  RETURN staffName
 END FUNCTION

 FUNCTION setStaffName(newName)
  SET this.staffName = newName
 END FUNCTION

 FUNCTION getJoiningDate()
  RETURN joiningDate
 END FUNCTION

 FUNCTION setJoiningDate(newDate)
  SET this.joiningDate = newDate
 END FUNCTION

 FUNCTION getQualification()
  RETURN qualification
 END FUNCTION

 FUNCTION
setQualification(newQualification)
  SET this.qualification =
newQualification
 END FUNCTION

 FUNCTION getAppointedBy()
  RETURN appointedBy
 END FUNCTION

 FUNCTION
setAppointedBy(newAppointedBy)
  SET this.appointedBy =
newAppointedBy
 END FUNCTION

 FUNCTION isJoined()
  // Returns whether the staff member
has joined yet
  RETURN joined
 END FUNCTION

 FUNCTION setJoined(joinStatus)
  SET this.joined = joinStatus
```

```
 // Method to set the working shifts if the
staff member is still employed
 FUNCTION setShifts(newShifts)
  IF isJoined() THEN // Only change shifts
if they are still with us
   SET this.shifts = newShifts // Update
the shifts
  END IF
 END FUNCTION

 // Method to terminate the part-time
staff member
 FUNCTION terminate()
  IF NOT terminated THEN // Check if
they are not already terminated
   setJoined(false) // Mark them as not
joined anymore
   SET this.terminated = true // Now they
are officially terminated
  ELSE
   PRINT "Staff is already terminated." //
Let us know if they were already
terminated
  END IF
 END FUNCTION

 // Override the display method to show
part-time staff details
 FUNCTION display()
  CALL super.display() // Show the
common details from the parent class
  IF isJoined() THEN // Only show details
if they are still employed
   PRINT "Working Hour: " + workingHour
// Show working hours
   PRINT "Wages Per Hour: " +
wagesPerHour // Show pay rate
   PRINT "Shifts: " + shifts // Show
working shift
   PRINT "Terminated: " + terminated //
Show termination status
   // Calculate and show daily income
   PRINT "Income Per Day: " +
(wagesPerHour * workingHour) //
Calculate daily earnings
  END IF
```

END FUNCTION
 	
// Show all the important info about this staff member
FUNCTION display()
PRINT "Vacancy Number: " + vacancyNumber
PRINT "Designation: " + designation
PRINT "Job Type: " + jobType
PRINT "Staff Name: " + staffName
PRINT "Joining Date: " + joiningDate
PRINT "Qualification: " + qualification
PRINT "Appointed By: " + appointedBy
PRINT "Joined: " + joined
END FUNCTION
 	
// Change if the staff member has joined or not
FUNCTION amendJoinedStatus(newStatus)
SET this.joined = newStatus
END FUNCTION
 	
END CLASS
	END FUNCTION
	
	END CLASS
	
	Pseudocode Explanation:
	
	Class Definition: PartTimeStaffHire class is defined as a subclass of StaffHire, which implies that it acquires the properties and methods from the class StaffHire.
	
	Attributes A class that has certain attributes such as part-time staff, working hours, wages, shifts and termination status.
	
	Constructor: The constructor sets all the inherited and new members. It invokes the parent class constructor, which establishes all common attributes, and it also initializes the part-time superclass characteristics.
	
	Getters: These functions expose the private data elements of

Explanation This is a pseudocode that describes the class structure/attributes/methods and what they do in a straightforward way, so that you can have the logic in your head without going into the Java syntax.	

6.3. Add full time Staff hire class

CLASS FullTimeStaffHire EXTENDS StaffHire // Instance variables DECLARE salary AS DOUBLE

6.4. Add Staff hire class GUI

CLASS StaffHireGUI EXTENDS JFrame DECLARE staffList AS ArrayList<StaffHire>

```
  DECLARE weeklyFractionalHours AS INTEGER

  // Constructor
  FUNCTION FullTimeStaffHire(vacancyNumber, designation, jobType, staffName,
                   joiningDate, qualification, appointedBy,
                   joined, salary, weeklyFractionalHours)
    CALL super constructor with (vacancyNumber, designation, jobType,
                     staffName, joiningDate, qualification,
                     appointedBy, joined)
    SET this.salary TO salary
    SET this.weeklyFractionalHours TO weeklyFractionalHours
  END FUNCTION

  // Accessor methods
  FUNCTION getSalary()
    RETURN salary
  END FUNCTION

  FUNCTION getWeeklyFractionalHours()
    RETURN weeklyFractionalHours
  END FUNCTION

  // Method to set salary
  FUNCTION setSalary(salary)
    IF isJoined() THEN
      SET this.salary TO salary
    ELSE
      PRINT "No staff appointed to set the salary."
    END IF
  END FUNCTION

  // Method to set weekly fractional hours
  FUNCTION setWeeklyFractionalHours(hours)
    SET this.weeklyFractionalHours TO hours
  END FUNCTION
```

```
  FUNCTION StaffHireGUI()
    SET window title and size
    SET layout to GridLayout
    INITIALIZE input fields for staff details
    ADD labels and input fields to window
    ADD buttons for actions (add staff, set salary, etc.)
    SET window visibility to true
  END FUNCTION

  FUNCTION addFullTimeStaff()
    TRY
      GET input values
      CREATE FullTimeStaffHire object
      ADD staff to staffList
      SHOW success message
    CATCH error
      SHOW error message
    END TRY
  END FUNCTION

  FUNCTION addPartTimeStaff()
    TRY
      GET input values
      CREATE PartTimeStaffHire object
      ADD staff to staffList
      SHOW success message
    CATCH error
      SHOW error message
    END TRY
  END FUNCTION

  FUNCTION setSalary()
    TRY
      GET vacancy number and staff name
      SEARCH staffList for staff
      IF found THEN SET salary
      ELSE SHOW not found message
    CATCH error
      SHOW error message
    END TRY
  END FUNCTION

  FUNCTION setWorkingShifts()
    TRY
```

```
  // Display method
  FUNCTION display()
    CALL super display method
    PRINT "Salary: " + salary
    PRINT "Weekly Fractional Hours: " +
weeklyFractionalHours
  END FUNCTION
END CLASS
```

```
      GET vacancy number and staff
name
      SEARCH staffList for staff
      IF found THEN SET shifts
      ELSE SHOW not found message
    CATCH error
      SHOW error message
    END TRY
  END FUNCTION

  FUNCTION terminatePartTimeStaff()
    TRY
      GET vacancy number
      SEARCH staffList for staff
      IF found THEN TERMINATE staff
      ELSE SHOW not found message
    CATCH error
      SHOW error message
    END TRY
  END FUNCTION

  FUNCTION displayStaff()
    TRY
      GET vacancy number
      SEARCH staffList for staff
      IF found THEN DISPLAY details
      ELSE SHOW not found message
    CATCH error
      SHOW error message
    END TRY
  END FUNCTION

  FUNCTION clearFields()
    SET all input fields to empty
  END FUNCTION

  FUNCTION main()
    CREATE StaffHireGUI instance
  END FUNCTION
END CLASS
```

Explanation

The class FullTimeStaffHire is a part of a system to recruit full time staff. It records certain information — such as how much an employee is paid and how many hours they work a week — that employers are

Explanation

StaffHireGUI class: This will provide a staff hiring interface in GUI form. It gives the facility to add full-time and part-time staff details, including vacancy number, post, salary, and working hours of staff.

legally required to document. When you add someone new to your full-time staff, you can fill in all their details, from salary and hours to contact information. There are ways to discover this information and to update the salary, but it only will allow changes if the staff member is working for you at the present time. Class also has a function to out output all detailed information of the staff.	The user interface has some buttons for adding staff, setting salaries, scheduling shifts, cancelling staff, and viewing staff details. When users click the buttons, their respective methods are invoked, and either a new staff member is added, or the staff details are updated. And the array list makes the information to add and retrieve the information easy and make the GUI look simple and clean.

7. Pseudocode for Button-Handling Methods

7.1. Add Full-Time Staff ()	*FUNCTION addFullTimeStaff()* *Get info from the user input boxes* *Make a full-time staff with that info* *Add that staff to the list* *END FUNCTION*	Backend Explanation: When the "Add Full-Time Staff" button is clicked by the user, the method collects the user inputs from GUI fields. A new FullTimeStaffHire object will be created with the given values. We can see that the constructor of the FullTimeStaffHire takes the parameters and calls the inherited constructor (StaffHire) to set the common attributes. The new staff object is then added to an ArrayList in the RecruitmentSystem class that stores all staff. Finally, the final message to the user to inform them staff member has been successfully added.
7.2. Add Part-Time Staff ()	*FUNCTION addPartTimeStaff()* *Get info from the user input boxes* *Make a part-time staff with that info* *Add that staff to the list* *Show message "Staff added!"* *END FUNCTION*	Explanation: This code is the same as we had to add the full-time staff, except for part-time staff. A new PartTimeStaffHire class is created, given the supplied values. PartTimeStaffHire constructor also call the StaffHire constructor to define the common attributes this the class. Under the RecruitmentSystem class, we add the new part-time staff object to the same ArrayList. The user is shown a confirmation message that the part-time staff was added.

7.3. Set Salary for Full-Time Staff ()	*FUNCTION setSalary()*
	Get vacancy number and new salary from user
	If vacancy number is okay
	Look for full-time staff with that number
	If staff has joined
	Set salary
	Show "Salary updated"
	Else
	Show "Staff hasn't joined"
	End if
	Else
	Show "Invalid vacancy number"
	End if
	END FUNCTION
		Backend Explanation:
		
		When the "Set Salary" button is pressed, it gets the vacancy number and the new salary from the GUI.
		
		It verifies if the vacancy number is valid (i.e., search the vacancy number in the ArrayList). If not null it finds the associated FullTimeStaffHire object.
		
		Then in the method section, it verifies, if the staff member has signed in by accessing the joined respectively. If they are also joined, it calls the setSalary method on the FullTimeStaffHire object, to set the new salary.
		
		The user is notified of success. Otherwise, a good message is displayed for the staff who has not yet joined.
		
		Error Message is displayed if vacancy number is not valid.

7.3. Set Working Shifts for Part-Time Staff ()	*FUNCTION setWorkingShifts()*
	Get vacancy number and new shifts from user
	If vacancy number is okay
	Look for part-time staff with that number
	If staff has joined
	Update shifts
	Show "Shifts updated"
	Else
	Show "Staff hasn't joined"
	End if
	Else
	Show "Invalid vacancy number"
	End if
	END FUNCTION
		Backend Explanation:
		
		It works much the same as the "Set Salary" method. It gets the vacancy number and new shift from the GUI.
		
		It validates if the vacancy number is valid and retrieves the PartTimeStaffHire object associate with the vacancy number.
		
		The strategy determines whether the staff has been added. If it has, it invokes the setShifts method of the PartTimeStaffHire object to change the shifts.
		
		If the update is successful then a success message is printed.

7.4. terminate PartTime Staff ()	*METHOD terminatePartTimeStaff()*
	TRY
	// First, get the number the user typed in for the vacancy
	
	// Then, check all the staff one by one
	FOR each staff in the list
	// See if this staff is a part-timer and has that vacancy number
	IF yes, then
	// Tell that staff to stop working
	do the terminate job for that staff
	// Let the user know it worked
	show a message saying it was successful
	// Then stop looking because we're done
	RETURN
	END IF
	END FOR
	
	// If we didn't find anyone, tell the user
	show a message saying, "Staff not found!"
	
	CATCH any mistakes that happen
	// Tell the user there was a problem and what it was
	show a message with the error
	END TRY
	END METHOD
		Explanation
		
		This process will assist you in terminating a part-time employee by the vacancy number. It validates input, searches staff listing and keeps the user informed along the way. If all goes well, it acknowledges the termination; otherwise, it lets the user know about the problems.
		
		Handle any errors:
		
		If the input is bad (there wasn't a number entered), show an alert box with the message "Please enter valid numeric values".
		
		For any other unhandled errors, print a message with their details.

7.5. displayStaff()	*START displayStaff* *TRY to do the following:* *- Get the vacancy number entered by the user (from a text field).* *- Convert that input into an integer.* *- Go through each staff in the staff list:* *- IF the staff's vacancy number matches the input:* *- Show that staff's details by calling their display method.* *- STOP the method (no need to check others).* *- IF no matching staff was found:* *- Show a message saying, "Staff not found!"* *CATCH any errors that happen during this process:* *- Show an error message that includes the specific error details.* *END displayStaff*	**Explanation** The function displayStaff begins by attempting to retrieve the vacancy number that the user entered. It then casts that number to an integer. Then, for each entry in the list of staff, it compares to determine if the vacancy number is found. If it does here something it finds, then it will display that staff member details and stop looking further. (If it doesn't find anyone with that vacancy number, it tells the user by using puts "Staff not found".
7.6. clearFields()	*When the user clicks "Clear" or we want to reset the form:* *1. Go through every input field on the screen — the ones where the user typed stuff.* *2. For each of those fields (like vacancy number, name, salary, hours, etc.),* *just wipe them clean — set them back to empty so it looks like a fresh form.* *3. Act like you're handing the user a brand-new form to fill again.*	**Explanation** Just whacking away all those text boxes one by one. No loops or conditions. Straightforward cleanup. Just think of the counter being reset when clicking on a "Reset" button, that is what is going on behind the curtains.

8. Button Method Implementations

A method per button action to organize and maintain it well. Those functions which you mentioned in pseudo code is the actual implementation that you want these buttons to do:

addFullTimeStaff(): Verifies input fields, create a new FullTimeStaffHire object and adds it to the staff list.

addPartTimeStaff(): Checks whether user input fields are valid and then adds a new PartTimeStaffHire object to the staff list.

setSalary(): It gets a full-time staff by vacancy number and change a salary for his/her.

setShifts(): Looks for a part-time staff by vacancy number and changes its working shifts.

terminateStaff(): search for a part time staff based on vacancy number and dismiss from service.

displayStaffDetails(): Shows the details of a staff by index.

clearFields(): Resets the input fields to their initial states.

9. GUI

Graphical user interfaces (GUIs) assist new users in getting work done quickly. Chen à Zhang (2007) state that beginners achieve more efficient use of GUIs and Rauterberg (1992) claims that beginners and experts work 51% faster in a GUI than in a character-based environment. Similarly, decreased cognitive load and error rates are reported as common user benefits in e.g. Dillon and Song (1997) and Staggers and Kobus (2000).

The GUI includes: Fields to input all required staff details. Add staff, add salary, remove staff buttons. A screen area for displaying staff information according to the user input and many more.

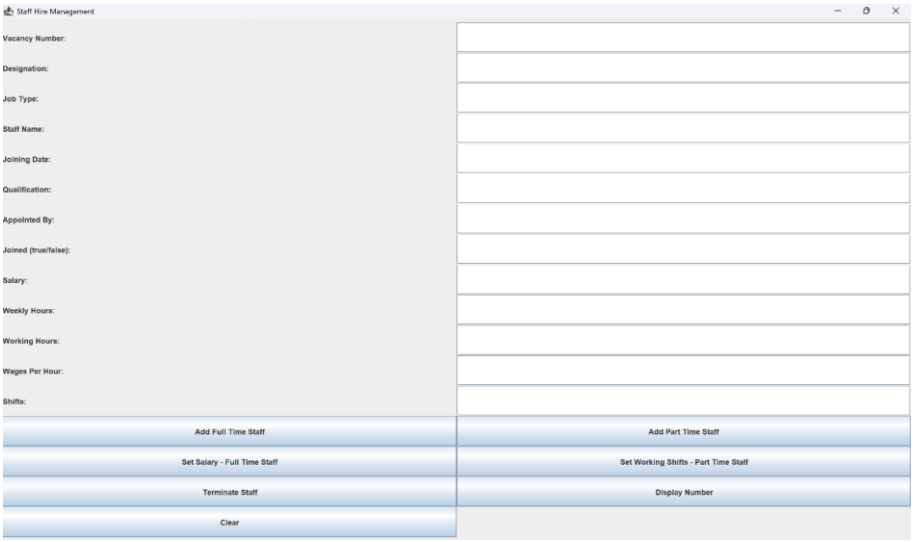

Figure -11: GUI of the staff hire management

10. Textboxes and Input Validation

We use GUI for RecruitmentSystem (java), and we validate input in text fields thoroughly using try-catch blocks. This means that the users cannot enter the wrong data that will lead to runtime or logic errors in the application. Chen and Zhang (2007)

10.1 EXPERIMENT

10.1.1. Testing the empty string.

As per the logic, when user try to click a button without adding the details. Error pop up text will be displayed.

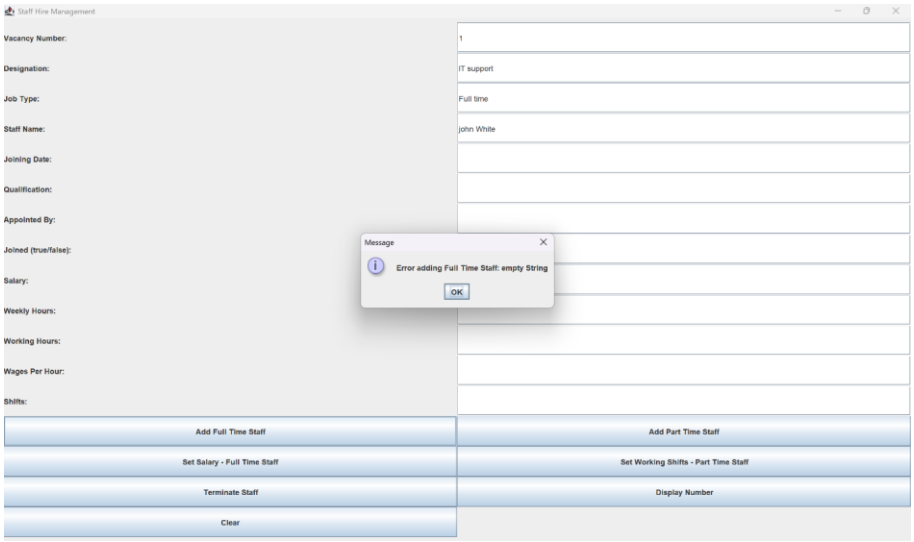

Figure -12: Testing the empty string.

10.2.1: ADDING A FULL-TIME STAFF MEMBER

We first present an example of the results using Test 1.

Description: Entered legitimate full-time staff details Pressed the "Add fulltime staff" button.

Positive Test: The staff should be added to staffHireList and a success message should be printed.

Actual Output:

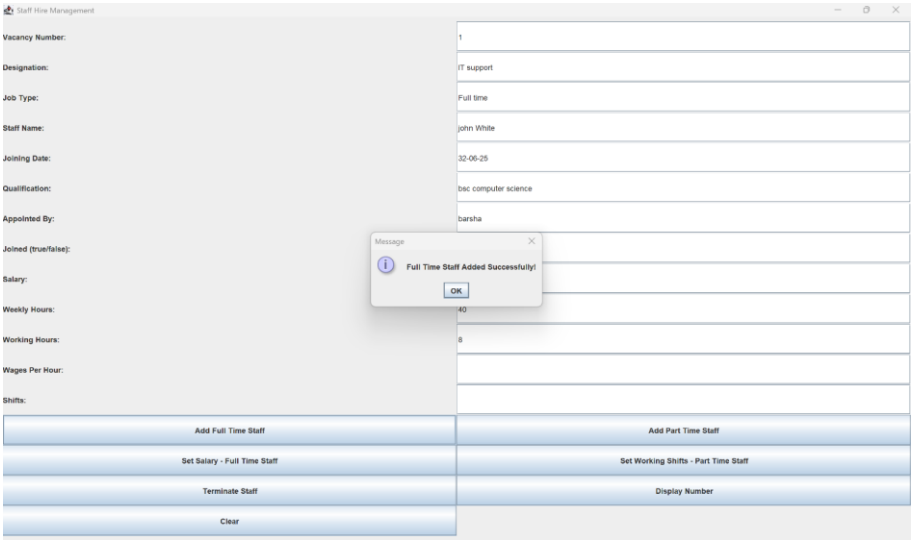

Figure -13: Adding full time

10.2.2: ADDING A FULL-TIME STAFF MEMBER salary with an error.

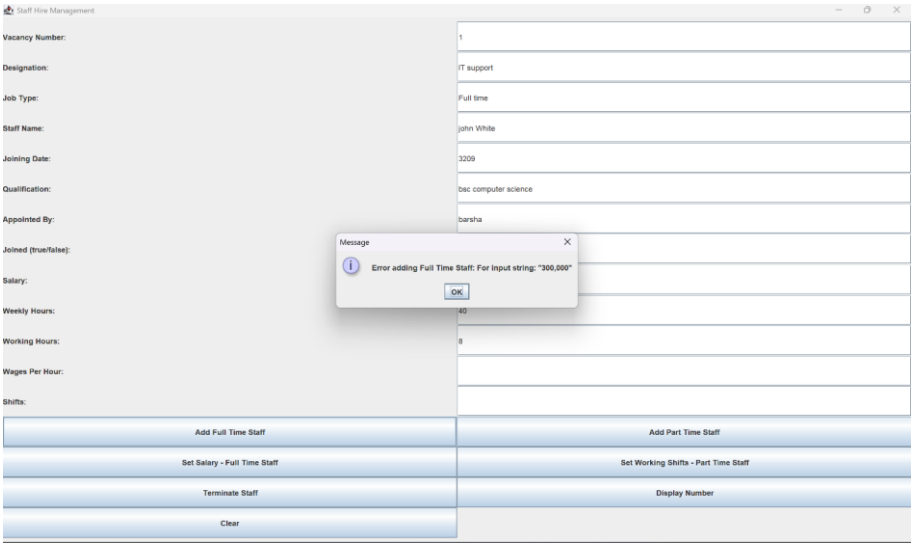

Figure -14: adding salary as string

10.3.1 Hire a Part-Time Staff Member

Description: I entered some legit data for part-time staff and clicked that Add Parttime Staff button.

Output Expected: Staff should be added to staffHireList, with success message.

Actual Output:

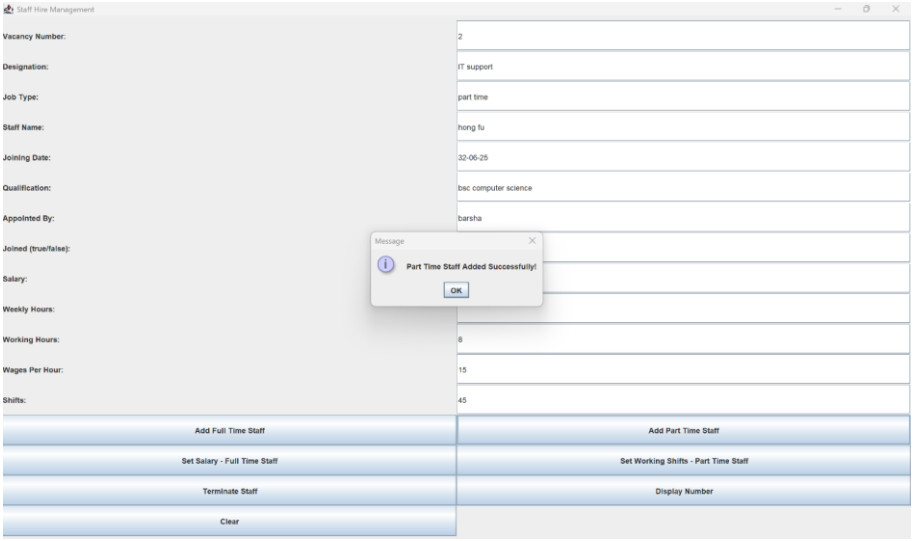

Figure -15: Added part time

10.3.2. Hire a Part-Time Staff Member without salary and hours

Even though we added all the details including working hours and weekly hours it will still show the error since for the part timer we can check the validation of the wages per hours and shifts.

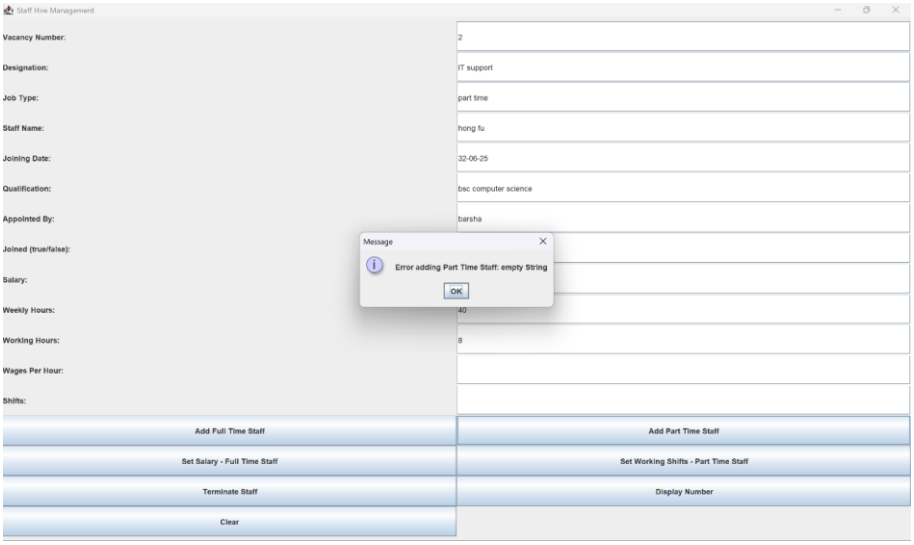

Figure -16: testing with empty section for wages per hour and shifts

10.4. Fixed Pay - Full-Time Salaried Member of fulltime Staff

Description: I entered a valid vacancy number and salary in the salary field for a full-time staff member and clicked the “Set Salary” button.

Expected Result: The salary of this employee gets updated, and Flash message will be shown.

Actual Output:

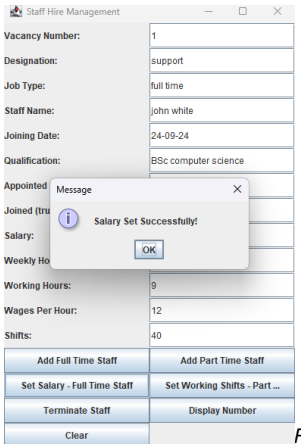

Figure -17: Testing adding salary.

10.4.1. Fixed Pay - Full-Time Salaried Member of fulltime Staff error

Although, the successful pop-up text in the backend data was yet not changed.

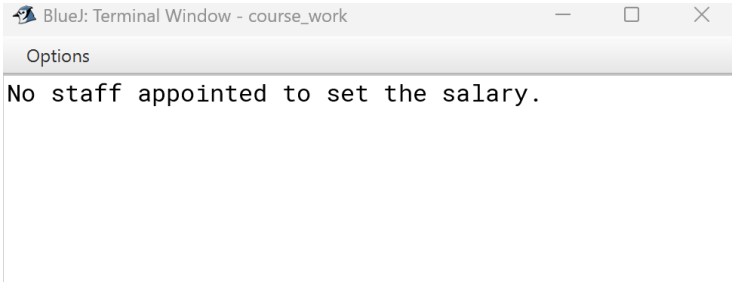

Figure -14: Error while adding salary

To change that I added clickable joined so staff would be added and set salary would properly.

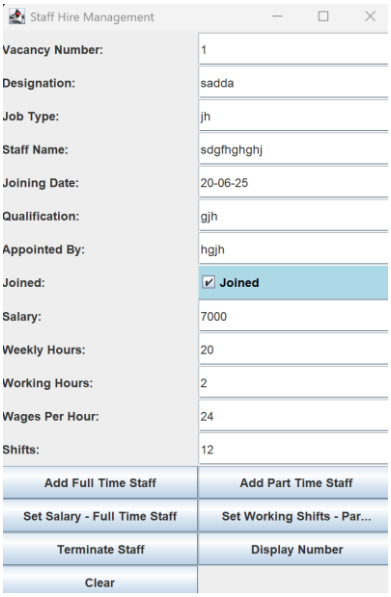

Figure -15: Update in the joined section to remove error

10.5. Display the record of added staff

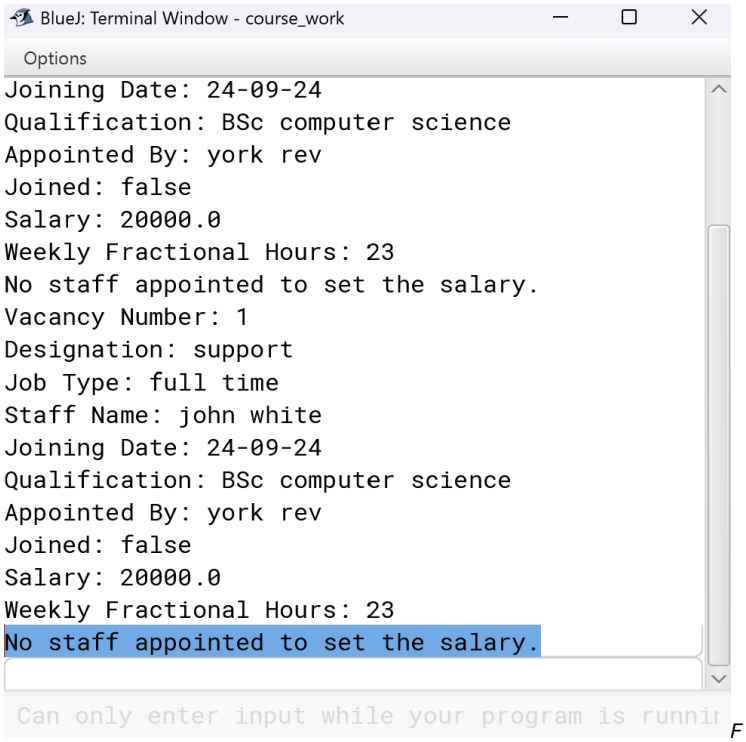

Figure -16: Display of output

Displayed the record of the hired staff. *error fix in section 10.4.1*

10.5.1 Display the record of added staff in the pop-up GUI style (additional features)

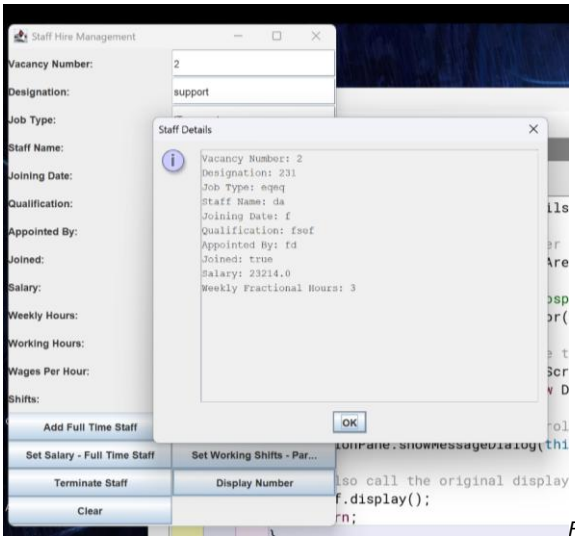

Figure -17: Display for full time

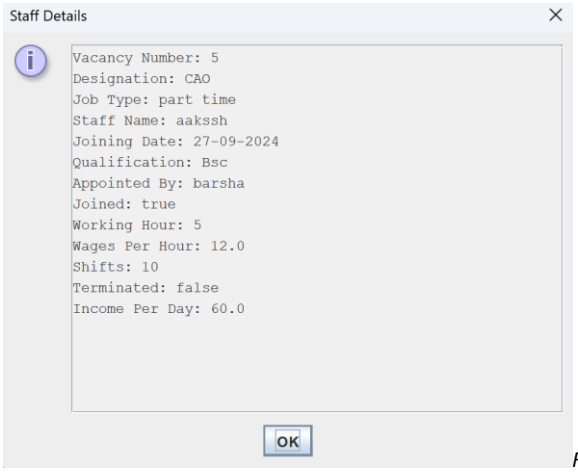

Figure -18: Display for part time

In order to simply show the details as a staff that should be created as previous, the following change will be applied to print the staff details in a popup dialog message instead to the console I modified the method displayStaff() to keep all the staff details into StringBuilder instead of using the display() method that shows the output using System. out. println()`. The method... takes the vacancy number, position, etc, etc, etc and creates a nicely formatted string with the appropriate labels and line breaks: For details about staff type, I was using instanceof to check whether the staff is a part-time person or a full-timer, and then I was casting the object to the appropriate type to access marks or working hours etc. At last, I showed this information I collected with JOptionPane. showMessageDialog(null, staffs.toString()); I am using the JOptionPane.showMessageDialog() and passing the StringBuilder toString(), and then it displays all staff details in a neat, readable format. (Pericles, B., 2019) investigates advanced strategies in utilizing JOptionPane in Java, that could facilitate development of dialog programming, especially for novices.

This way you keep my old code structure, and just reroute the output from the console to a user

11. Set Shifts - Part-Time Staff Member

Desired Output: The shifts should be updated for the provided employee and show success message.

The actual result:

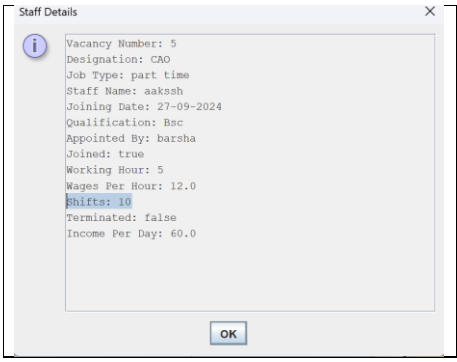

Figure -19: Initial shifts

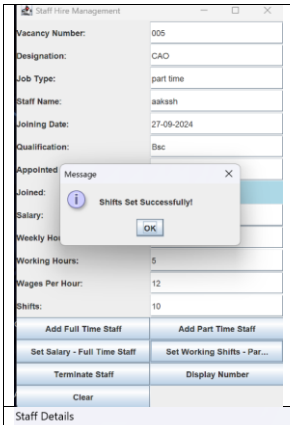

Figure -20: Setting the new shift

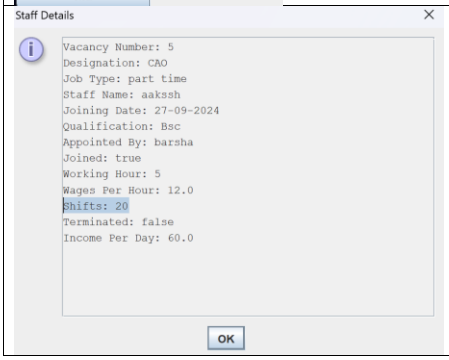

Figure -21: shift changed after update through set working shifts.

12.Terminate a Part-Time Staff Member

Explanation: The user submitted a valid vacancy number (of a part-time employee) and activated the "Terminate" button.

Expected Output: Staff member fields should be cleared, joined should be set to false and a success message should be shown.

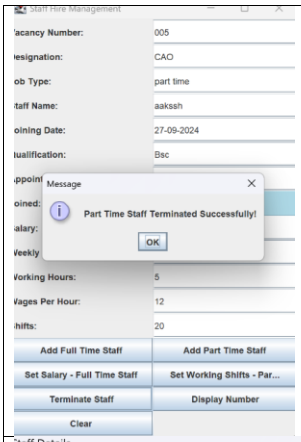

Figure -22: Terminated successfully.

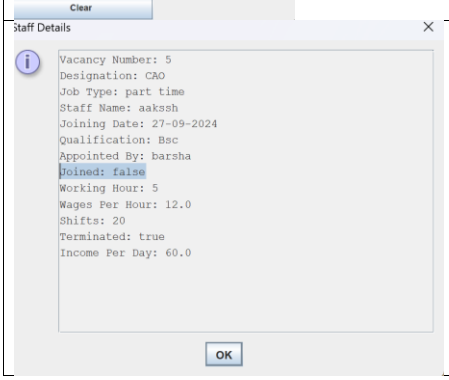

Figure -23: Joined statue to false.

13. Compile and Run from Command Prompt

Description: Downloaded and ran the staffhireGUI. java file from the CMD command prompt. Used JDK and changed the path in the pc manually to add the java path. Opened the cmd and through the path of the java we compile the java file (javac

filename.name)

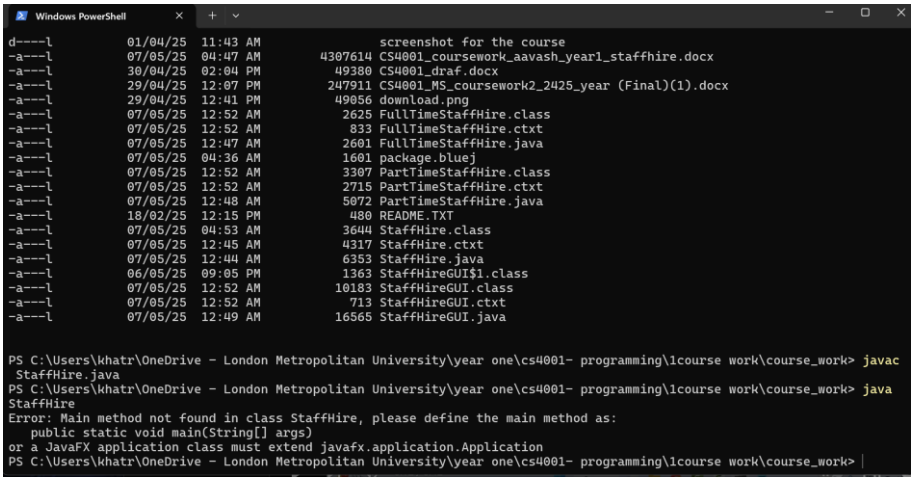

Figure -24: cmd accessing the path and file.

Expected Output: Compiler output will compile and run with no errors, GUI displayed.

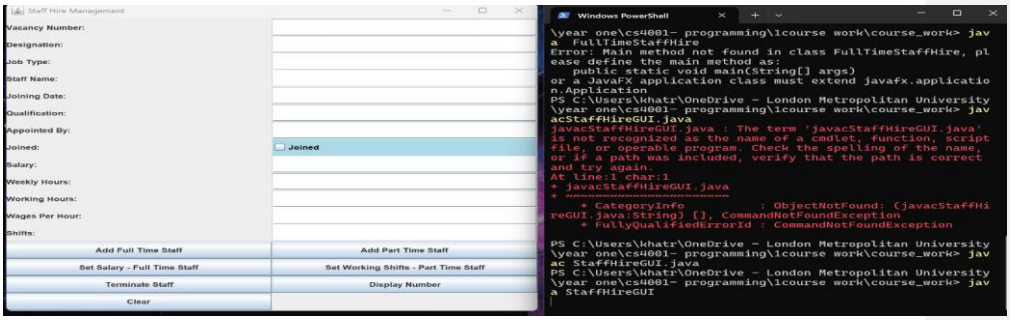

Figure -24: opened the GUI

While complying the java file it showed error but upon trying again with simple change opened the GUI and made user able to access the GUI and add/ modify the staff hiring process.

14. Challenges Faced

The biggest challenges of this project were:

Type Casting: The following image might help explain some of the consideration that had to be taken not to get ClassCastException errors when down/super-casting between parent and child classes. (Lee et al., 2015).

Complex GUI Design: The development of a GUI with all the fields that also was user intuitive proved to be difficult. I tried with other layout managers for it and found out that the following worked out the best. (Shneiderman et al.1993).

Data Validation: Developing rigorous validation for all these varieties and ensuring clear error messages when it is not processable. (Freeman,2011).

15. Final Thoughts

I learned more in this project about array list, GUI, etc. The development and testing Staff Recruitment System from design has improved my programming knowledge and problem-solving skills. Any knowledge learned from this endeavours will prove to be invaluable when I embark on other software projects.

16. Reference

Chen, J.-W. and Zhang, J., 2007. Comparing text-based and graphic user interfaces for novice and expert users. Proceedings of the American Medical Informatics Association (AMIA) Annual Symposium, pp.125–129. https://www.academia.edu/26791138/Comparing_Text_based_and_Graphic_User_Interfaces_for_novice_and_expert_users [Accessed 5 May 2025]

Dillon, A. and Song, M., 1997. An empirical comparison of the usability for novice and expert searchers of a textual and a graphic interface to an art-resource database. *Journal of Digital Information*, [online] Available at: https://www.researchgate.net/publication/2561770_An_Empirical_Comparison_of_the_Usability_for_Novice_and_Expert_Searchers_of_a_Textual_and_a_Graphic_Interface_to_an_Art-Resource_Database [Accessed 6 May 2025].

Freeman, Adam. "Validating Form Data." (2011). [online] Available at: https://www.semanticscholar.org/paper/Validating-Form-Data-Freeman/1cc183469f7396f4ac4f86a053a65a2602fcf49b [Accessed 6 May 2025].

Lee, Byoungyoung et al. "Type Casting Verification: Stopping an Emerging Attack Vector." USENIX Security Symposium (2015). https://www.semanticscholar.org/paper/Type-Casting-Verification%3A-Stopping-an-Emerging-Lee-Song/c68aa444565e10897b21e33a67e4643a322a2bfa [Accessed 7 May 2025].

Pericles, B., 2019. Core Java APIs. In: *Core Java APIs*. [online] Hoboken, NJ: Wiley, pp.161–221. Available at: https://doi.org/10.1002/9781119584773.ch5 [Accessed 7 May 2025].

Shneiderman, Ben and Andrew Sears. "Layout appropriateness: guiding user interface design with simple task descriptions." (1993). Available at: https://www.semanticscholar.org/paper/Layout-appropriateness%3A-guiding-user-interface-with-Shneiderman-Sears/1d1180842ed4de84d8bca9e7b2cc5504fafe1b57 [Accessed 7 May 2025].

Staggers, N. and Kobus, D., 2000. Comparing response time, errors, and satisfaction between text-based and graphical user interfaces during nursing order tasks. *Journal of the American Medical Informatics Association*, 7(2), pp.164–176. Available at: https://www.researchgate.net/publication/12584811_Comparing_Response_Time_Errors_and_Satisfaction_Between_Text-based_and_Graphical_User_Interfaces_During_Nursing_Order_Tasks

[Accessed 6 May 2025].

Westerman, S.J., 1997. Individual differences in the use of command line and menu computer interfaces. *International Journal of Human-Computer Interaction*, 9(2), pp.133–150.