CS4001NI Programming

30% Individual Coursework

2023-24 Autumn

Student Name: Bachan Timalsina

London Met ID: 23047401

College ID: NP01NT4A230100

Group: N6

Assignment Due Date: Friday, May 10, 2024

Assignment Submission Date: 9 May, 2024

ACKNOWLEDGMENT

Throughout this coursework, Mrs. Astha Sharma and Mr. Ujjwal Subedi has provided me with invaluable guidance, support, and mentorship. For that, I want to sincerely thank them. Their knowledge and support have been crucial in helping me gain a greater understanding of my programming abilities. I am also grateful for my classmate's helpful attitudes and intelligent conversations, which enhanced my educational experience. Finally, I would like to express sincere thanks to all the libraries, tools, and resources that helped me finish this coursework.

Contents

1. INTRODUCTION 6
1.1 AIM AND OBJECTIVES 6
1.2 TERMINOLOGIES 7
2. CLASS DIAGRAM 8
3. METHOD DESCRIPTION 13
4. PSEUDOCODE 16
5. TESTING 18
TEST 1 18
TEST 2 19
2.1 19
2.2 20
2.3 21
2.4 22
2.5 23
TEST 3 24
3.1 24
3.2 25
6. ERROR DETECTION 26
7. CONCLUSION 28
8. APPENDIX 29

Table of Figures

Figure 1 BlueJ logo 7
Figure 2 MS Word logo 7
Figure 3 Draw.io logo 7
Figure 4 Teacher Class Diagram 8
Figure 5 Lecturer Class Diagram 9
Figure 6 Tutor Class Diagram 10
Figure 7 TeacherGUI Class Diagram 11
Figure 8 Relation between all classes 12
Figure 9 compiled and run from cmd 18
Figure 10 Adding to Lecturer 19
Figure 11 Adding to tutor 20
Figure 12 Grading Assignment 21
Figure 13 setting salary 22
Figure 14 removing tutor 23
Figure 15 displaying wrong error message 24
Figure 16 error message at tutor 25
Figure 17 Syntax error 26
Figure 18 Semantic error 27

Table of tables

Table 1 Compile and run from Command Prompt 18
Table 2 Adding to Lecturer 19
Table 3 Adding to tutor 20
Table 4 Grading Assignment 21
Table 5 Setting Salary 22
Table 6 Removing Tutor 23
Table 7 showing error after entering invalid value in Lecturer part................................. 24
Table 8 showing error after entering invalid value in Tutor part 25

1. INTRODUCTION

The task of this graphical user interface (GUI) design project is to develop a simple interface for an educational institution's teacher data management system. Users will be able to grade assignments, set tutor salaries, remove tutors, add lecturers, and display important data through the GUI. The interface will have buttons to carry out particular tasks in addition to text fields for entering different teacher details. Try and catch blocks for error handling will also guarantee smooth data input and processing, giving users a perfect experience.

1.1 AIM AND OBJECTIVES

AIMS:

- To create a Graphical User Interface (GUI) application with the ability to add, grade, remove, and display teacher information to manage teachers, particularly lecturers and tutors.

OBJECTIVES:

- Create a simple graphical user interface (GUI) with text fields for teacher information and buttons for different actions.
- Make a new Lecturer object and add it to the list of teachers to implement the function to add a lecturer.
- Create a new Tutor object and add it to the list of teachers to implement the function to add a tutor.
- Optimize the GUI layout to make it look better and user-friendly, which will enhance the user experience in general.

1.2 TERMINOLOGIES

BlueJ is an integrated development environment (IDE) designed specifically for Java programming that is easy to use. It was created especially with beginners in mind, with the goal of making object-oriented programming concepts easier to learn and teach. BlueJ stands out due to its focus on visual representation, which enables users to engage with objects and classes directly. Understanding basic concepts like classes, objects, inheritance, and polymorphism is made easier with the help of this visual method.

Figure 1 BlueJ logo

Microsoft Word is a strong word-processing program that can be used for creating, editing, and formatting documents. It has an easy-to-use interface. Word is an essential tool for both personal and professional use thanks to its huge formatting options, advanced editing features, and seamless integration with other Microsoft Office applications.

Figure 2 MS Word logo

With the help of the adaptable and user-friendly drawing tool Draw.io, users can create a wide variety of flowcharts, diagrams, and other visual representations. Draw.io's wide range of shapes and elements, coupled with its user-friendly interface, enable users to effectively collaborate and visually express ideas. Draw.io is a web-based application that allows users to create and edit diagrams from any device with an internet connection. It is flexible and accessible.

Figure 3 Draw.io logo

2. CLASS DIAGRAM

A class diagram is a diagram that shows classes and their connections and is used in software design and modelling. The software can be modelled at a high level of abstraction and without requiring us to view the source code thanks to class diagrams.

A class diagram's classes and the source code's classes line up. The diagram displays the class names, their attributes, the relationships between them, and occasionally even their methods.

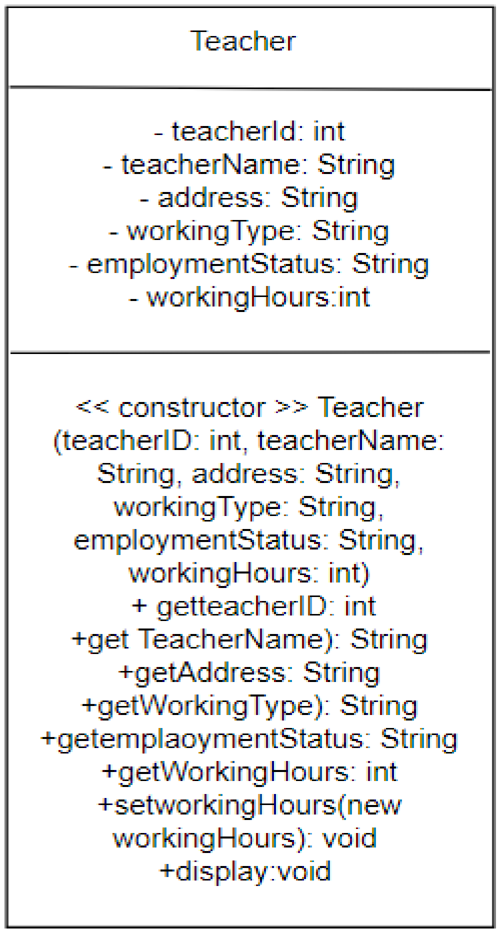

Teacher
- teacherId: int
- teacherName: String
- address: String
- workingType: String
- employmentStatus: String
- workingHours:int
<< constructor >> Teacher (teacherID: int, teacherName: String, address: String, workingType: String, employmentStatus: String, workingHours: int)
+ getteacherID: int
+get TeacherName): String
+getAddress: String
+getWorkingType): String
+getemplaoymentStatus: String
+getWorkingHours: int
+setworkingHours(new workingHours): void
+display:void

Figure 4 Teacher Class Diagram

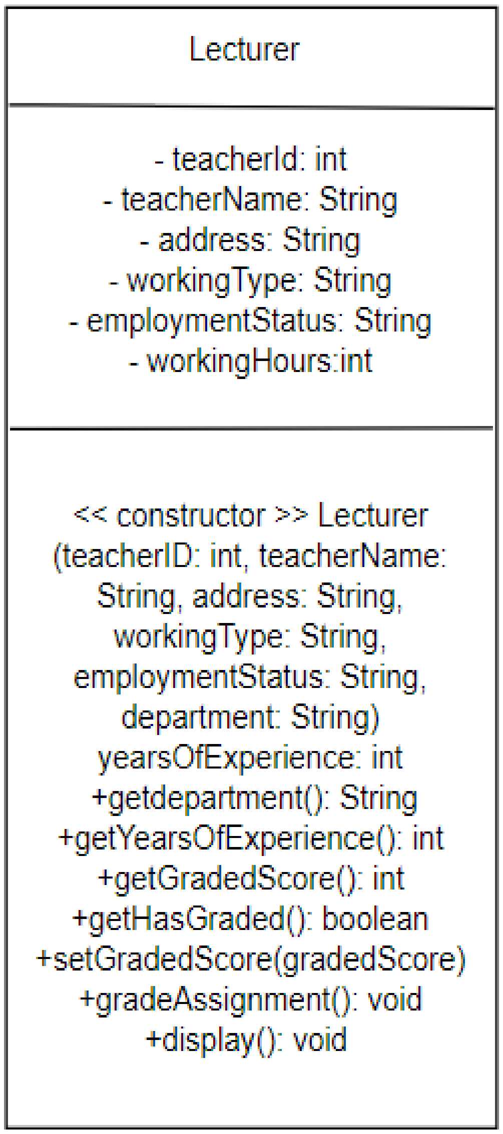
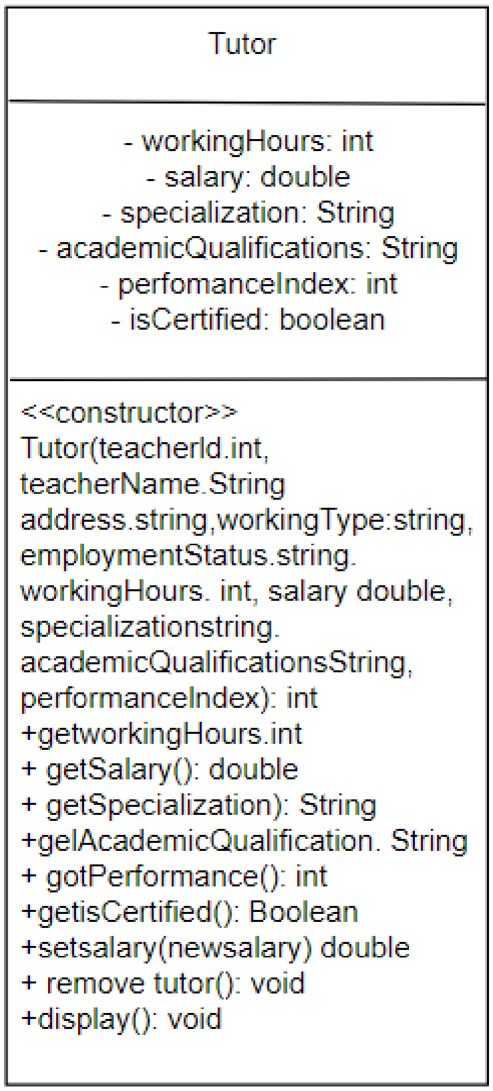


Figure 5 Lecturer Class Diagram


Tutor

- workingHours: int
- salary: double
- specialization: String
- academicQualifications: String
- perfomanceIndex: int
- isCertified: boolean

<<constructor>>
Tutor(teacherId.int,
teacherName.String
address.string,workingType:string,
employmentStatus.string.
workingHours. int, salary double,
specializationstring.
academicQualificationsString,
performanceIndex): int
+getworkingHours.int
+ getSalary(): double
+ getSpecialization): String
+gelAcademicQualification. String
+ gotPerformance(): int
+getisCertified(): Boolean
+setsalary(newsalary) double
+ remove tutor(): void
+display(): void

Figure 6 Tutor Class Diagram

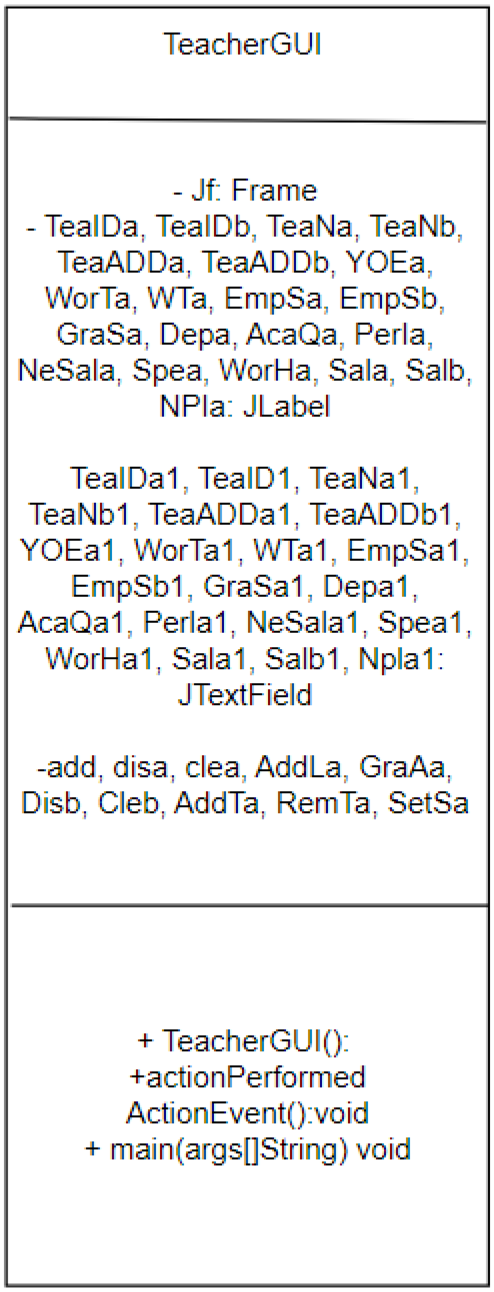

TeacherGUI
- Jf: Frame
- TeaIDa, TeaIDb, TeaNa, TeaNb, TeaADDa, TeaADDb, YOEa, WorTa, WTa, EmpSa, EmpSb, GraSa, Depa, AcaQa, Perla, NeSala, Spea, WorHa, Sala, Salb, NPla: JLabel
TeaIDa1, TeaID1, TeaNa1, TeaNb1, TeaADDa1, TeaADDb1, YOEa1, WorTa1, WTa1, EmpSa1, EmpSb1, GraSa1, Depa1, AcaQa1, Perla1, NeSala1, Spea1, WorHa1, Sala1, Salb1, Npla1: JTextField
-add, disa, clea, AddLa, GraAa, Disb, Cleb, AddTa, RemTa, SetSa
+ TeacherGUI():
+actionPerformed ActionEvent():void
+ main(args[]String) void

Figure 7 TeacherGUI Class Diagram

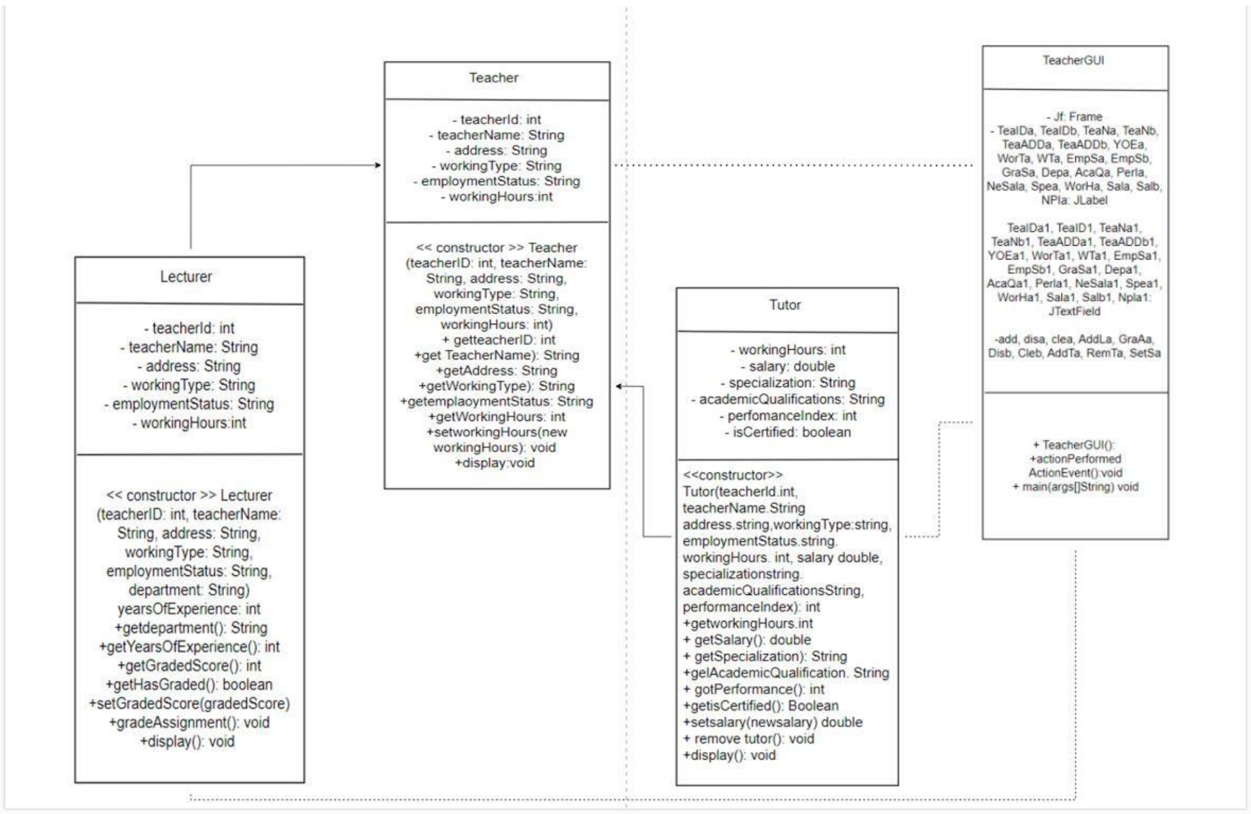

Teacher
- teacherId: int
- teacherName: String
- address: String
- workingType: String
- employmentStatus: String
- workingHours:int
<< constructor >> Teacher (teacherID: int, teacherName: String, address: String, workingType: String, employmentStatus: String, workingHours: int)
+ getteacherID: int
+get TeacherName): String
+getAddress: String
+getWorkingType): String
+getemplaoymentStatus: String
+getWorkingHours: int
+setworkingHours(new workingHours): void
+display:void
Lecturer
- teacherId: int
- teacherName: String
- address: String
- workingType: String
- employmentStatus: String
- workingHours:int
<< constructor >> Lecturer (teacherID: int, teacherName: String, address: String, workingType: String, employmentStatus: String, department: String)
yearsOfExperience: int
+getdepartment(): String
+getYearsOfExperience(): int
+getGradedScore(): int
+getHasGraded(): boolean
+setGradedScore(gradedScore)
+gradeAssignment(): void
+display(): void
Tutor
- workingHours: int
- salary: double
- specialization: String
- academicQualifications: String
- perfomanceIndex: int
- isCertified: boolean
<<constructor>>
Tutor(teacherId.int, teacherName.String address.string,workingType:string, employmentStatus.string. workingHours. int, salary double, specializationstring. academicQualificationsString, performanceIndex): int
+getworkingHours.int
+ getSalary(): double
+ getSpecialization): String
+gelAcademicQualification. String
+ gotPerformance(): int
+getisCertified(): Boolean
+setsalary(newsalary) double
+ remove tutor(): void
+display(): void
TeacherGUI
- Jf: Frame
- TealDa, TealDb, TeaNa, TeaNb, TeaADDa, TeaADDb, YOEa, WorTa, WTa, EmpSa, EmpSb, GraSa, Depa, AcaQa, Perla, NeSala, Spea, WorHa, Sala, Salb, NPla: JLabel
TealDa1, TealD1, TeaNa1, TeaNb1, TeaADDa1, TeaADDb1, YOEa1, WorTa1, WTa1, EmpSa1, EmpSb1, GraSa1, Depa1, AcaQa1, Perla1, NeSala1, Spea1, WorHa1, Sala1, Salb1, Npla1: JTextField
-add, disa, clea, AddLa, GraAa, Disb, Cleb, AddTa, RemTa, SetSa
+ TeacherGUI():
+actionPerformed ActionEvent():void
+ main(args[]String) void

Figure 8 Relation between all classes

3. METHOD DESCRIPTION

1. **Constructor - public TeacherGUI():**
    - **Description:** Initializes the TeacherGUI class by setting up the graphical user interface (GUI) components and creating an empty ArrayList to store Teacher objects.
    - **Parameters:** None.
    - **Return Type:** Void.
2. **GUI method - public void GUI():**
    - **Description:** Sets up the GUI components, including labels, text fields, buttons, and action listeners, for managing teacher information.
    - **Parameters:** None.
    - **Return Type:** Void.
3. **Action Listener (addL and addT) - public void actionPerformed(ActionEvent ae):**
    - **Description:** Handles the "Add Lecturer" or "Add Tutor" button click events by validating input fields, creating a new Lecturer or Tutor object with provided information, and adding it to the ArrayList.
    - **Parameters:** ActionEvent ae - the event object representing the user's action.
    - **Return Type:** Void.
4. **Action Listener (clr and clr1) - public void actionPerformed(ActionEvent ae):**
    - **Description:** Handles the "Clear" button click events for either lecturer or tutor by clearing input fields to allow users to enter new data.
    - **Parameters:** ActionEvent ae - the event object representing the user's action.
    - **Return Type:** Void.
5. **Action Listener (disp and disp1) - public void actionPerformed(ActionEvent ae):**

- **Description**: Handles the "Display" button click events for either lecturer or tutor by retrieving input data, validating it, and displaying the information of the respective teacher.
- **Parameters:** ActionEvent ae - the event object representing the user's action.
- **Return Type:** Void.

6. **Action Listener (SetSal) - public void actionPerformed(ActionEvent ae):**
    - **Description:** Handles the "Set Salary" button click event by retrieving input values for teacher ID, new performance index, and new salary, updating the tutor's salary and performance index based on certain conditions, and displaying the outcome message.
    - **Parameters:** ActionEvent ae - the event object representing the user's action.
    - **Return Type:** Void.
7. **Action Listener (RemT) - public void actionPerformed(ActionEvent ae):**
    - **Description:** Handles the "Remove Tutor" button click event by retrieving input value for teacher ID, removing the tutor from the list of teachers if found, resetting tutor's attributes to default values, and displaying the outcome message.
    - **Parameters:** ActionEvent ae - the event object representing the user's action.
    - **Return Type:** Void.
8. **Action Listener (Grade) - public void actionPerformed(ActionEvent ae):**
    - **Description:** Grades an assignment for a lecturer based on input parameters such as graded score, department, and years of experience, updates the lecturer's attributes accordingly, and returns a string representing the assigned grade.
    - **Parameters**: Graded score (int), department (String), years of experience (int).
    - **Return Type:** A string representing the grade assigned to the assignment.
9. **main method - public static void main(String[] args):**

- **Description**: Serves as the entry point for the program by creating an instance of the teacherGUI class and calling the GUI() method to start the GUI application.
- **Output:** Starting point for the GUI application.
- **Return Type:** Void.

4. PSEUDOCODE

Pseudocode is an informal method of describing programming that doesn't depend on technological factors or accurate programming language syntax. It is used in the preparation of program initial drafts or outlines. Pseudocode eliminates important details while summarizing the operation of a program. To make sure that programmers understand the requirements of a software project and align code appropriately, system designers create pseudocode.

PSEUDOCODE OF TeacherGUI

```
CREATE class named TeacherGUI
DO
  DECLARE teachers as ArrayList of Teacher

  CREATE method GUI()
  DO
    CREATE JFrame with title "Coursework GUI"
    SET layout manager of JFrame to null
    MAKE JFrame visible

    ADD JLabel "Lecturer" to JFrame
    ADD JLabel "Teacher ID:" to JFrame
    ADD JTextField for teacher ID to JFrame
    ADD JButton "Add Lecturer" to JFrame
    ADD JButton "Grade Assignments" to JFrame
```

```
        ADD JLabel "Tutor" to JFrame
        ADD JLabel "Teacher ID:" to JFrame
        ADD JTextField for teacher ID to JFrame
        ADD JButton "Add Tutor" to JFrame
        ADD JButton "Remove Tutor" to JFrame
        ADD ActionListener for "Add Lecturer" button
        ADD ActionListener for "Grade Assignments" button
    END DO

    CREATE ActionListener for "Add Lecturer" button
    DO
        RETRIEVE data from input fields for lecturer information
        VALIDATE input data
        IF data is valid THEN
            CREATE new Lecturer object with retrieved data
            ADD new Lecturer to teachers list
            DISPLAY success message
        ELSE
            DISPLAY error message indicating missing or invalid data
        END IF
    END DO

END DO
```

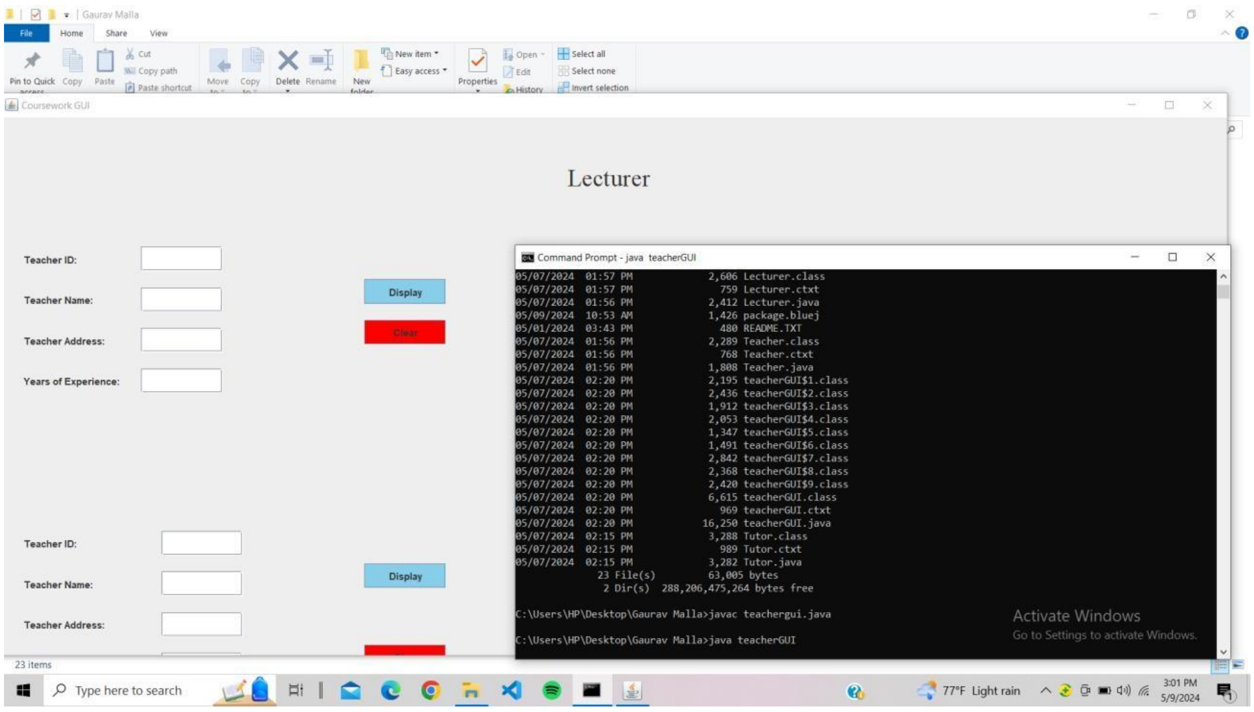
5. TESTING

TEST 1

Test that the program can be compiled and run using the command prompt.

Table 1 Compile and run from Command Prompt

Objective	To check if the program can be compile and run from command prompt.
Action	Opening bluej and command prompt and adding the file location on command prompt
Expected Result	While entering the file name in the command prompt it should open the program.
Actual Result	The program runs from the command prompt.
Conclusion	Successful



```
05/07/2024  01:57 PM             2,606 Lecturer.class
05/07/2024  01:57 PM               759 Lecturer.ctxt
05/07/2024  01:56 PM             2,412 Lecturer.java
05/09/2024  10:53 AM             1,426 package.bluej
05/01/2024  03:43 PM               480 README.TXT
05/07/2024  01:56 PM             2,289 Teacher.class
05/07/2024  01:56 PM               768 Teacher.ctxt
05/07/2024  01:56 PM             1,808 Teacher.java
05/07/2024  02:20 PM             2,195 teacherGUI$1.class
05/07/2024  02:20 PM             2,436 teacherGUI$2.class
05/07/2024  02:20 PM             1,912 teacherGUI$3.class
05/07/2024  02:20 PM             2,053 teacherGUI$4.class
05/07/2024  02:20 PM             1,347 teacherGUI$5.class
05/07/2024  02:20 PM             1,491 teacherGUI$6.class
05/07/2024  02:20 PM             2,842 teacherGUI$7.class
05/07/2024  02:20 PM             2,368 teacherGUI$8.class
05/07/2024  02:20 PM             2,420 teacherGUI$9.class
05/07/2024  02:20 PM             6,615 teacherGUI.class
05/07/2024  02:20 PM               969 teacherGUI.ctxt
05/07/2024  02:20 PM            16,250 teacherGUI.java
05/07/2024  02:15 PM             3,288 Tutor.class
05/07/2024  02:15 PM               989 Tutor.ctxt
05/07/2024  02:15 PM             3,282 Tutor.java
              23 File(s)         63,005 bytes
               2 Dir(s)  288,206,475,264 bytes free

C:\Users\HP\Desktop\Gaurav Malla>javac teachergui.java

C:\Users\HP\Desktop\Gaurav Malla>java teacherGUI
```

Figure 9 compiled and run from cmd

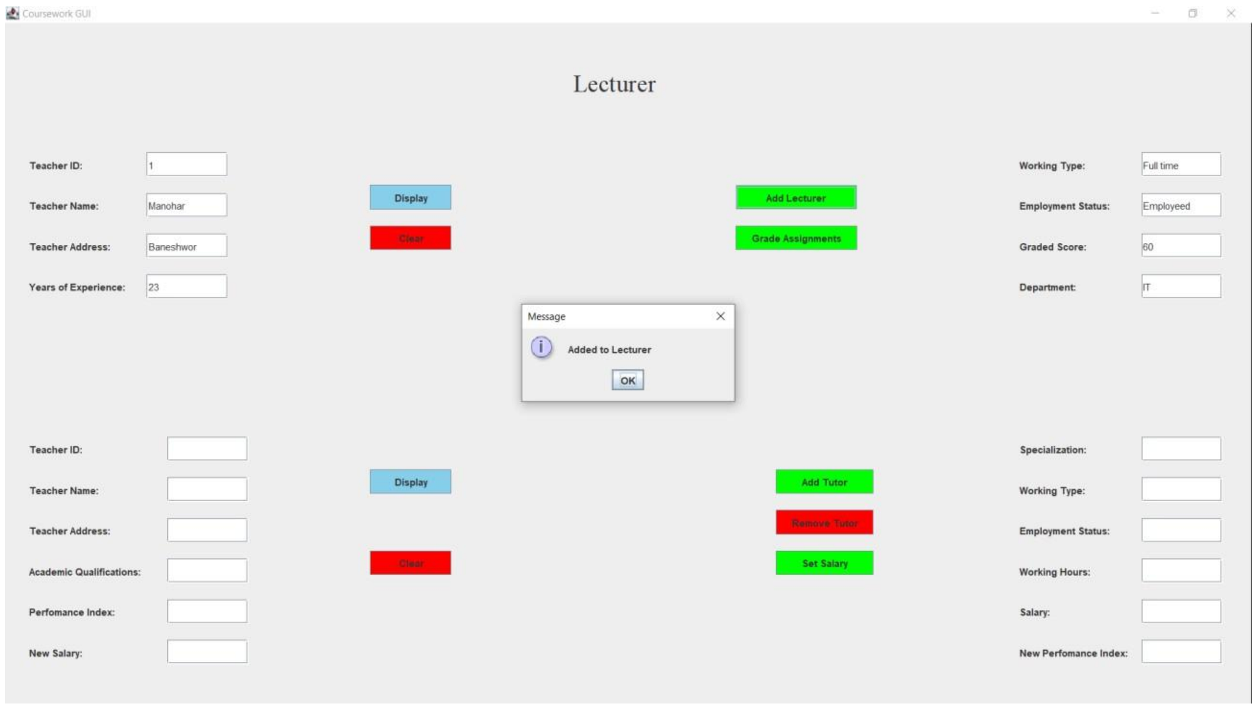
TEST 2

2.1

To show the evidence of Adding to Lecturer:

Table 2 Adding to Lecturer

Objective	To show the evidence of adding to lecturer
Action	Added the values on GUI for Lecturer.
Expected Result	While clicking on add lecturer button the lecturer should be added.
Actual Result	While clicking on add lecturer button the lecturer was added.
Conclusion	Successful



Teacher ID: 1
Teacher Name: Manohar
Teacher Address: Baneshwor
Years of Experience: 23
Working Type: Full time
Employment Status: Employeed
Graded Score: 60
Department: IT

Added to Lecturer

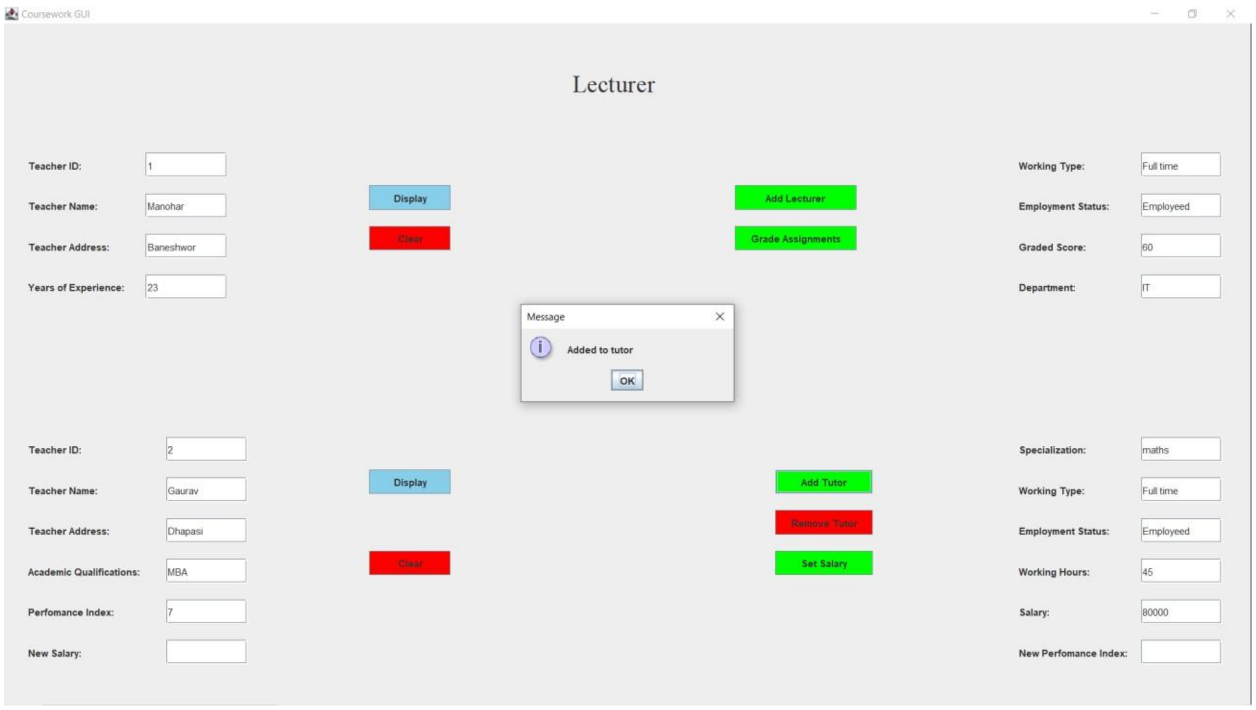
Figure 10 Adding to Lecturer

2.2

To show evidence of adding to tutor:

Table 3 Adding to tutor

Objective	Showing the evidence of adding to tutor
Action	Added the values on GUI for tutor.
Expected Result	While clicking on add tutor button the tutor should be added.
Actual Result	While clicking on add tutor button the tutor was added.
Conclusion	Successful



Lecturer

Teacher ID:	1
Teacher Name:	Manohar
Teacher Address:	Baneshwor
Years of Experience:	23
Working Type:	Full time
Employment Status:	Employeed
Graded Score:	60
Department:	IT

Message

Added to tutor

OK

Teacher ID:	2
Teacher Name:	Gaurav
Teacher Address:	Dhapasi
Academic Qualifications:	MBA
Perfomance Index:	7
New Salary:	
Specialization:	maths
Working Type:	Full time
Employment Status:	Employeed
Working Hours:	45
Salary:	80000
New Perfomance Index:	

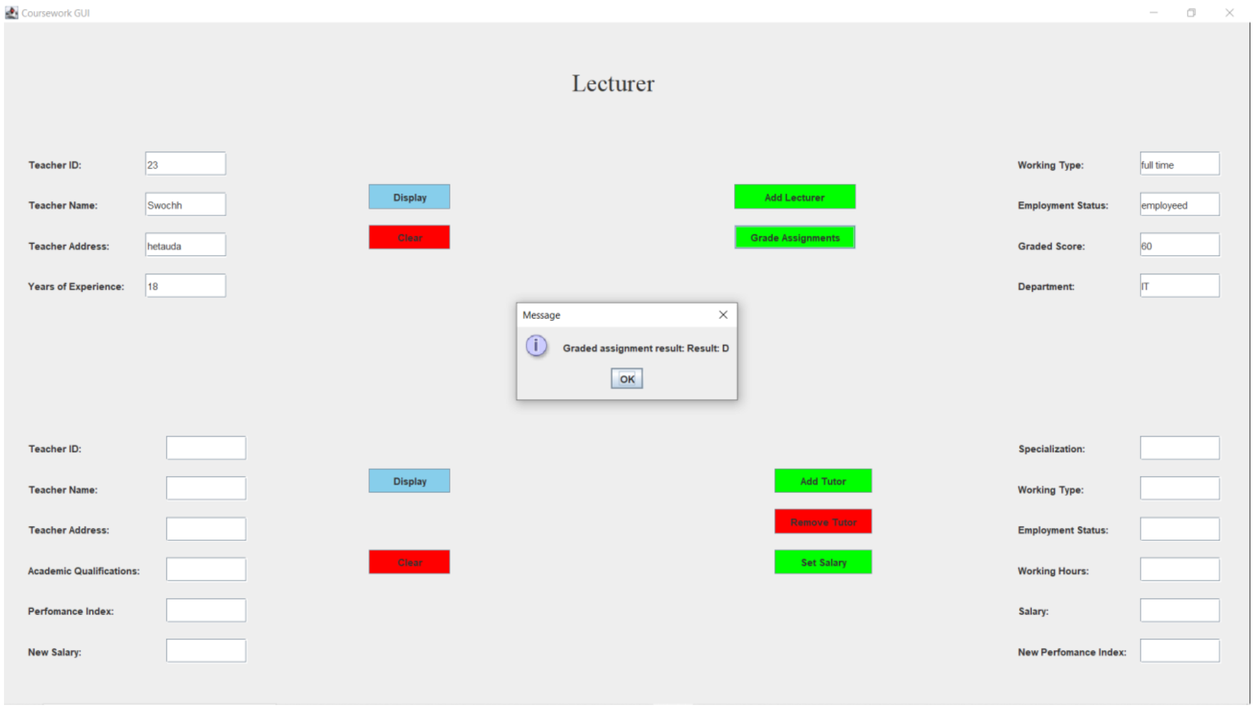
Figure 11 Adding to tutor

2.3

To show evidence of grading assignment:

Table 4 Grading Assignment

Objective	Showing the evidence of grading assignment
Action	Added the graded score on GUI for Grading Assignment.
Expected Result	While clicking on Grade Assignment button the grade result should be displayed.
Actual Result	While clicking on Grade Assignment button the grade result was displayed.
Conclusion	Successful



Teacher ID: 23
Teacher Name: Swochh
Teacher Address: hetauda
Years of Experience: 18
Working Type: full time
Employment Status: employeed
Graded Score: 60
Department: IT

Graded assignment result: Result: D

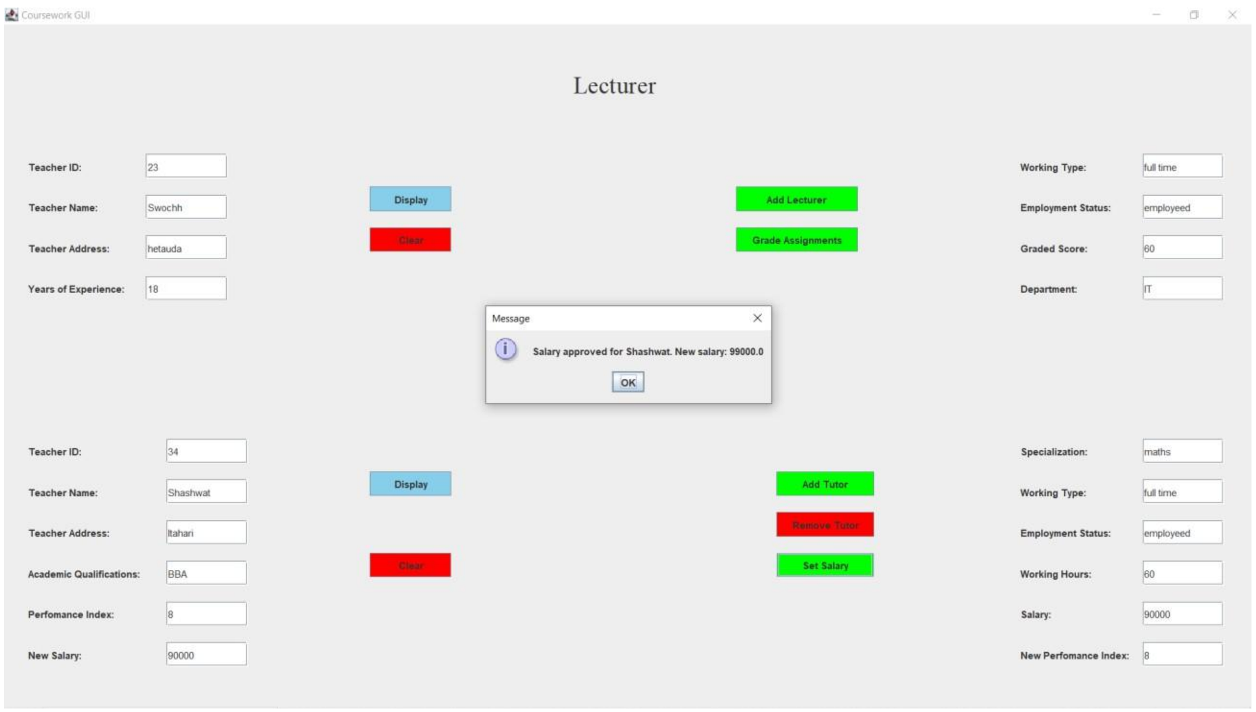
Figure 12 Grading Assignment

2.4

To show evidence of setting salary:

Table 5 Setting Salary

Objective	Showing the evidence of setting salary
Action	Added the new salary on GUI for Setting Salary.
Expected Result	While clicking on Set Salary button the salary of the tutor should be updated.
Actual Result	While clicking on Set Salary button the salary of the tutor was updated.
Conclusion	Successful



Lecturer

Teacher ID: 23
Teacher Name: Swochh
Teacher Address: hetauda
Years of Experience: 18

Working Type: full time
Employment Status: employeed
Graded Score: 60
Department: IT

Message

Salary approved for Shashwat. New salary: 99000.0

Teacher ID: 34
Teacher Name: Shashwat
Teacher Address: Itahari
Academic Qualifications: BBA
Perfomance Index: 8
New Salary: 90000

Specialization: maths
Working Type: full time
Employment Status: employeed
Working Hours: 60
Salary: 90000
New Perfomance Index: 8

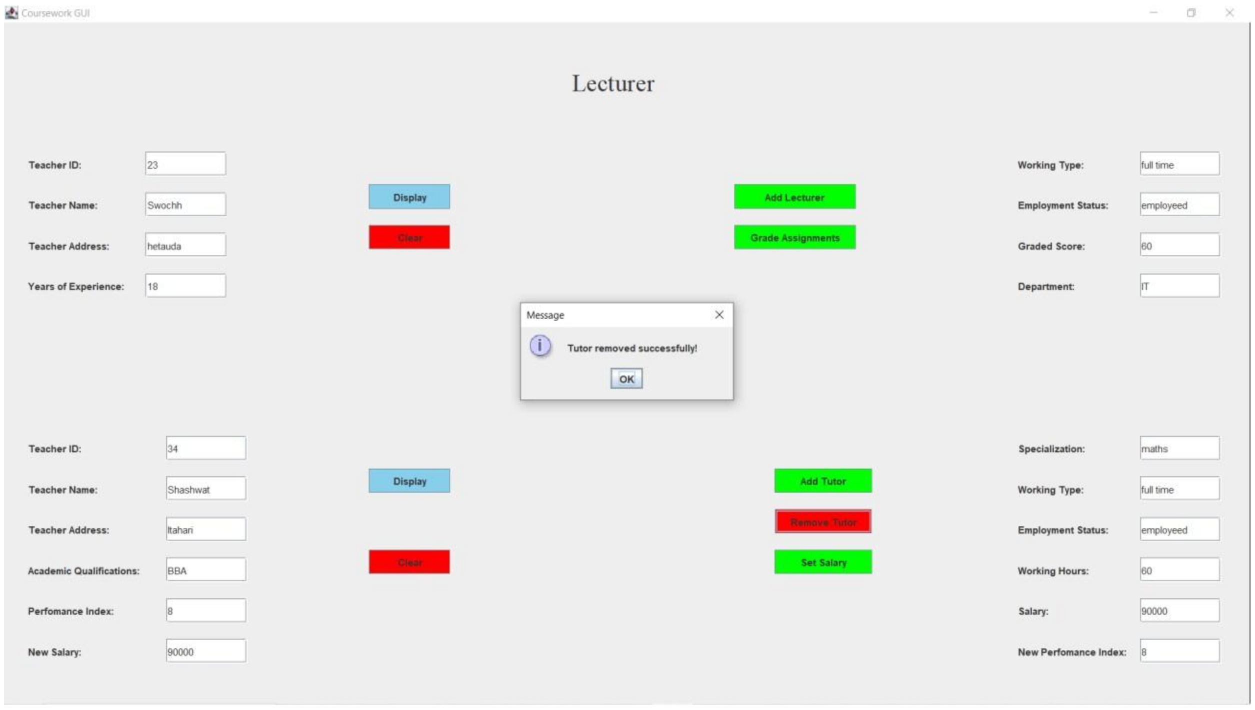
Figure 13 setting salary

2.5

To show evidence of Remove tutor:

Table 6 Removing Tutor

Objective	Showing the evidence of removing tutor.
Action	Added an existing tutor id and clicked the remove tutor button.
Expected Result	While clicking on remove tutor button the tutor should be removed.
Actual Result	While clicking on remove tutor button the tutor was removed.
Conclusion	Successful



Teacher ID:	23
Teacher Name:	Swochh
Teacher Address:	hetauda
Years of Experience:	18
Working Type:	full time
Employment Status:	employeed
Graded Score:	60
Department:	IT

Tutor removed successfully!

Teacher ID:	34
Teacher Name:	Shashwat
Teacher Address:	Itahari
Academic Qualifications:	BBA
Perfomance Index:	8
New Salary:	90000
Specialization:	maths
Working Type:	full time
Employment Status:	employeed
Working Hours:	60
Salary:	90000
New Perfomance Index:	8

Figure 14 removing tutor

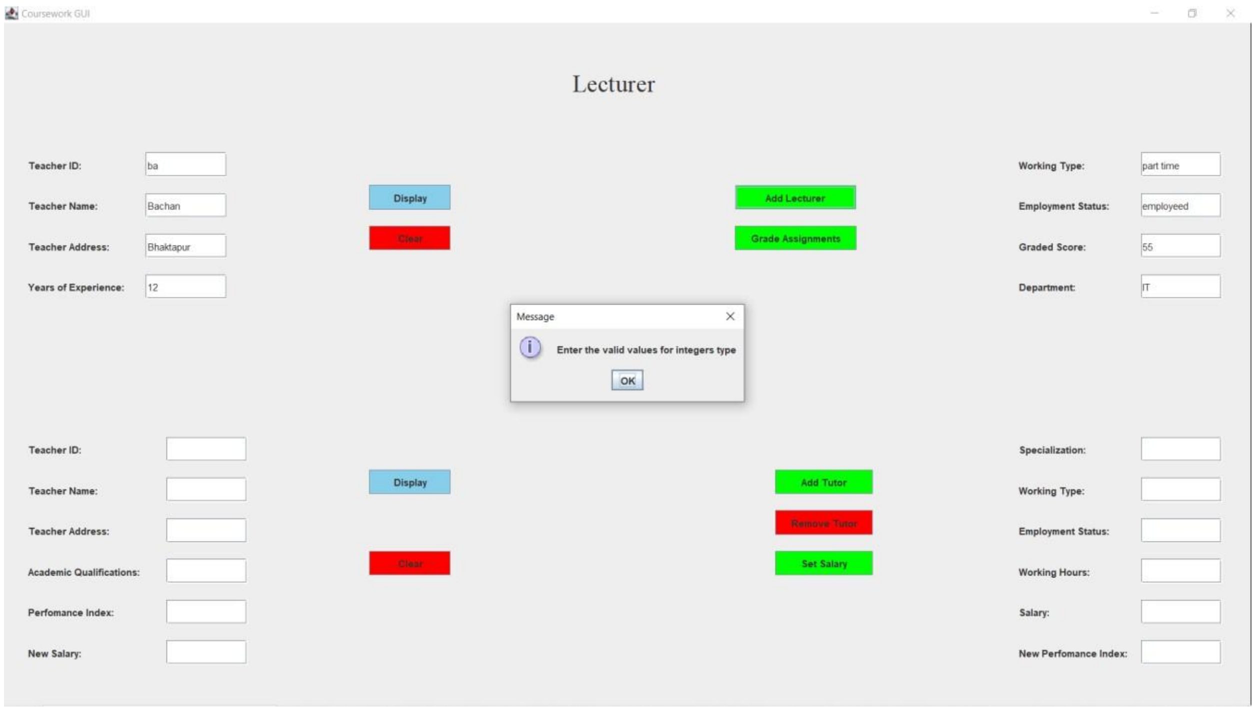
TEST 3

3.1

To test the appropriate dialog box appear when unsuitable values are entered for the teacher ID in the lecturer part.

Table 7 showing error after entering invalid value in Lecturer part

Objective	To test the appropriate dialog box appear when unsuitable values are entered in teacher ID at lecturer
Action	Passing the string value for the teacher id in lecturer part.
Expected Result	While entering string value in the teacher id it should show an appropriate dialogue box.
Actual Result	While entering string value in the teacher id it should show an appropriate dialogue box.
Conclusion	Successful



Teacher ID:	ba
Teacher Name:	Bachan
Teacher Address:	Bhaktapur
Years of Experience:	12
Working Type:	part time
Employment Status:	employeed
Graded Score:	55
Department:	IT

Message

Enter the valid values for integers type

OK

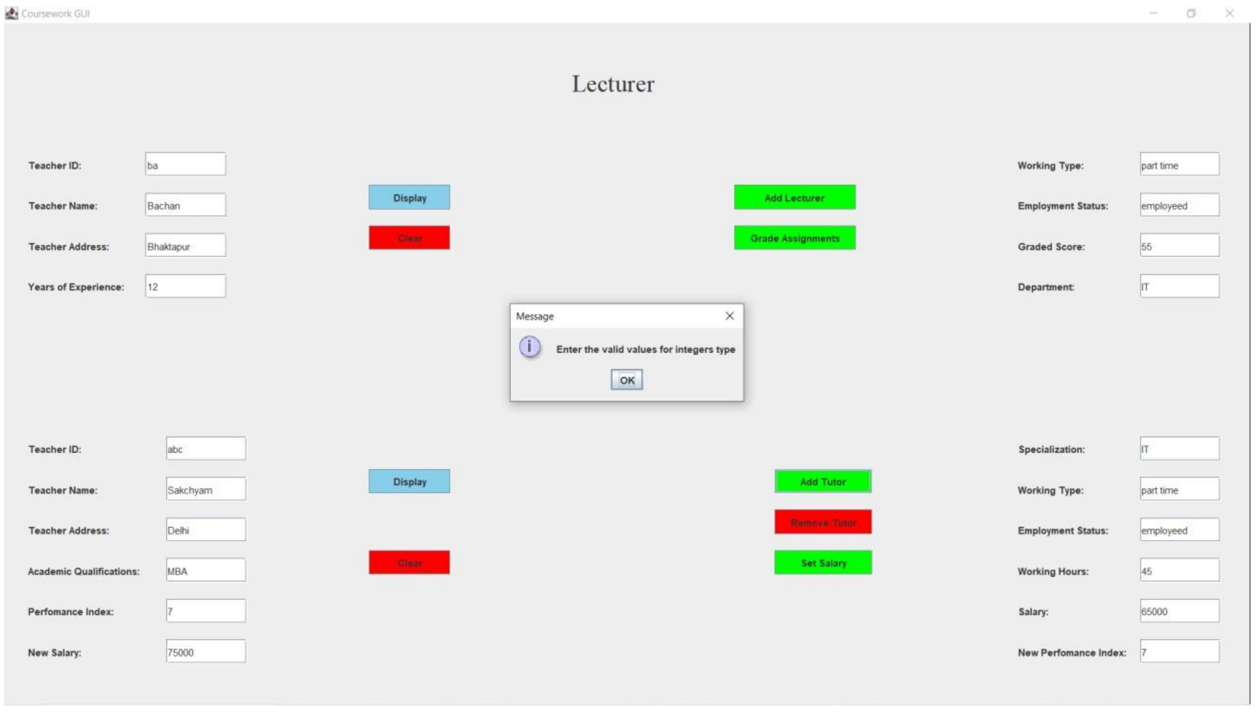
Figure 15 displaying wrong error message

3.2

To test that appropriate dialog box appear when unsuitable values are entered for the Teacher ID in the tutor part.

Table 8 showing error after entering invalid value in Tutor part

Objective	To test the appropriate dialog box appears when unsuitable values are entered in teacher ID at tutor
Action	Passing the string value for the teacher id in lecturer part.
Expected Result	While entering string value in the teacher id it should show an appropriate dialogue box.
Actual Result	While entering string value in the teacher id it did show an appropriate dialogue box.
Conclusion	Successful



Teacher ID:	ba
Teacher Name:	Bachan
Teacher Address:	Bhaktapur
Years of Experience:	12
Working Type:	part time
Employment Status:	employeed
Graded Score:	55
Department:	IT

Message

Enter the valid values for integers type

OK

Teacher ID:	abc
Teacher Name:	Sakchyam
Teacher Address:	Delhi
Academic Qualifications:	MBA
Perfomance Index:	7
New Salary:	75000
Specialization:	IT
Working Type:	part time
Employment Status:	employeed
Working Hours:	45
Salary:	65000
New Perfomance Index:	7

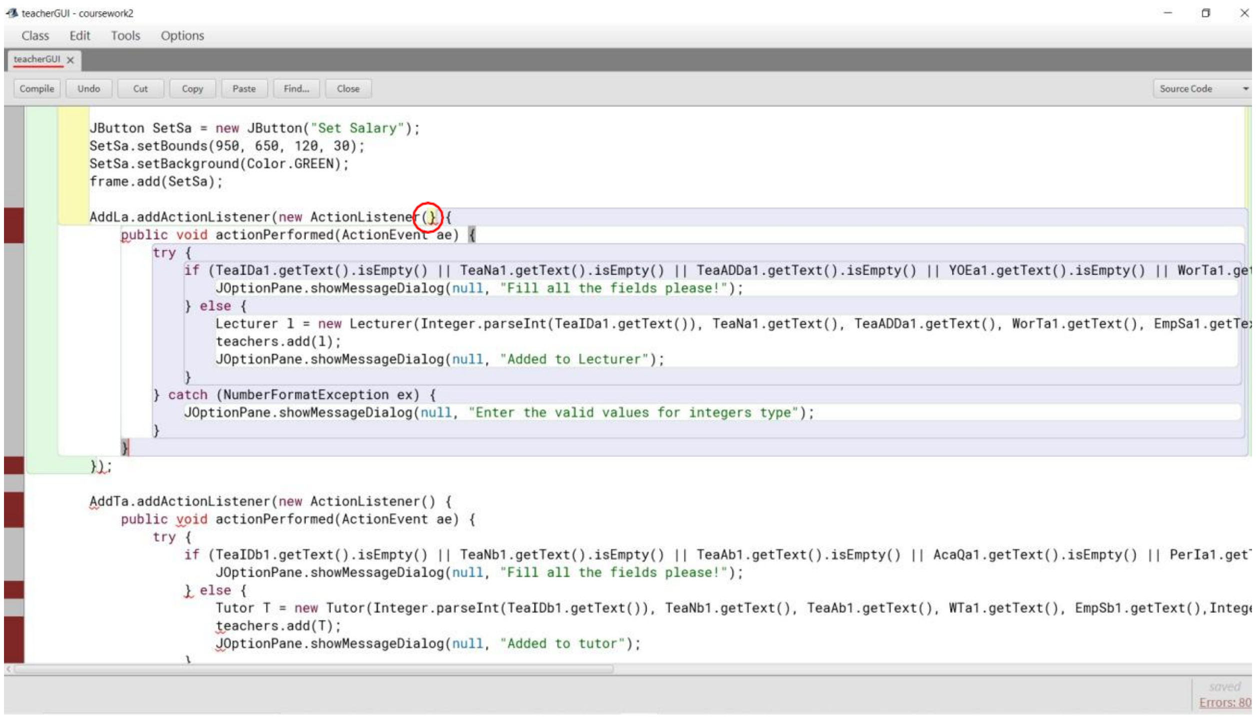
Figure 16 error message at tutor

6. ERROR DETECTION

The system does not guarantee that the data received by a device is the same as the data transmitted by another device when data is transferred between devices. When a message is transmitted and not received exactly at the recipient end, this is called an error.

Syntax Error:

A syntax error is a mistake in code that refuses the rules of the programming language and keeps the code from working properly. These typical programming errors happen when programmers write code incorrectly by violating the syntax rules of the language, like misspelling commands or using incorrect punctuation.



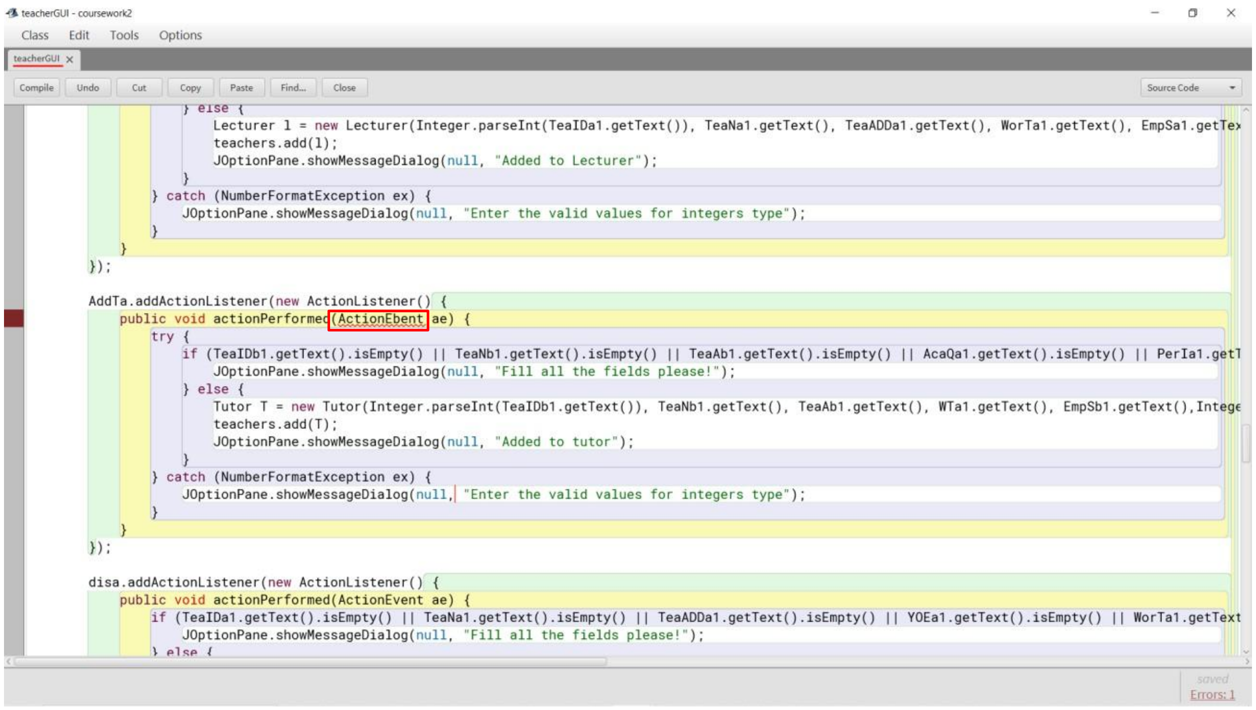
```
JButton SetSa = new JButton("Set Salary");
SetSa.setBounds(950, 650, 120, 30);
SetSa.setBackground(Color.GREEN);
frame.add(SetSa);

AddLa.addActionListener(new ActionListener(}){
    public void actionPerformed(ActionEvent ae) {
        try {
            if (TeaIDa1.getText().isEmpty() || TeaNa1.getText().isEmpty() || TeaADDa1.getText().isEmpty() || YOEa1.getText().isEmpty() || WorTa1.get
                JOptionPane.showMessageDialog(null, "Fill all the fields please!");
            } else {
                Lecturer l = new Lecturer(Integer.parseInt(TeaIDa1.getText()), TeaNa1.getText(), TeaADDa1.getText(), WorTa1.getText(), EmpSa1.getTe
                teachers.add(l);
                JOptionPane.showMessageDialog(null, "Added to Lecturer");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(null, "Enter the valid values for integers type");
        }
    }
});

AddTa.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        try {
            if (TeaIDb1.getText().isEmpty() || TeaNb1.getText().isEmpty() || TeaAb1.getText().isEmpty() || AcaQa1.getText().isEmpty() || PerIa1.get
                JOptionPane.showMessageDialog(null, "Fill all the fields please!");
            } else {
                Tutor T = new Tutor(Integer.parseInt(TeaIDb1.getText()), TeaNb1.getText(), TeaAb1.getText(), WTa1.getText(), EmpSb1.getText(),Intege
                teachers.add(T);
                JOptionPane.showMessageDialog(null, "Added to tutor");
```

Figure 17 Syntax error

Here, I got a syntax error by keeping the wrong bracket. I solved this by adding the appropriate bracket.

Semantic Error:

A semantic error is a bug in your code that makes it impossible for the compiler to understand it. Even though the logic you've written might be sound, the way it's written will make the program crash.



```
                } else {
                    Lecturer l = new Lecturer(Integer.parseInt(TeaIDa1.getText()), TeaNa1.getText(), TeaADDa1.getText(), WorTa1.getText(), EmpSa1.getTex
                    teachers.add(l);
                    JOptionPane.showMessageDialog(null, "Added to Lecturer");
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Enter the valid values for integers type");
            }
        }
    });

    AddTa.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEbent ae) {
            try {
                if (TeaIDb1.getText().isEmpty() || TeaNb1.getText().isEmpty() || TeaAb1.getText().isEmpty() || AcaQa1.getText().isEmpty() || PerIa1.getT
                    JOptionPane.showMessageDialog(null, "Fill all the fields please!");
                } else {
                    Tutor T = new Tutor(Integer.parseInt(TeaIDb1.getText()), TeaNb1.getText(), TeaAb1.getText(), WTa1.getText(), EmpSb1.getText(),Intege
                    teachers.add(T);
                    JOptionPane.showMessageDialog(null, "Added to tutor");
                }
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Enter the valid values for integers type");
            }
        }
    });

    disa.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            if (TeaIDa1.getText().isEmpty() || TeaNa1.getText().isEmpty() || TeaADDa1.getText().isEmpty() || YOEa1.getText().isEmpty() || WorTa1.getText
                JOptionPane.showMessageDialog(null, "Fill all the fields please!");
            } else {
```

Figure 18 Semantic error

Here, I made a mistake by making a spelling error. I fixed the word with appropriate alphabet that should be there.

7. CONCLUSION

As we complete our project, we have created a useful tool for managing teachers and their data. Adding new lecturers and tutors, managing salaries, and grading assignments are all made simple by our program. Because of its easy-to-use design, anyone who needs to handle teacher data can access it. This coursework has taught us a lot about developing reliable and user-friendly software. We are getting better at creating graphical user interfaces that make it obvious how to use the program, and we have neatly organized our code to ensure smooth operation.

One important thing we've learned is how important it is to handle mistakes appropriately. Try and catch blocks allow us to identify errors such as typing letters instead of numbers and provide users with error messages that help them correct the mistake. All in all, this project has been an interesting educational opportunity. It has taught us useful programming and problem-solving techniques and demonstrated how to make software that is both efficient and easy to use. We are pleased with our accomplishments and eager to keep improving these abilities in the future.

8. APPENDIX

For Teacher:

```
// Teacher class

public class Teacher {

    // Attributes

    private int teacherID;

    private String teacherName;

    private String address;

    private String workingType;

    private String employmentStatus;

    private int workingHours;

    // Constructor with parameters

    public Teacher(int teacherID, String teacherName, String address, String
workingType, String employmentStatus) {

        // instance variables

        this.teacherID = teacherID;

        this.teacherName = teacherName;

        this.address = address;

        this.workingType = workingType;

        this.employmentStatus = employmentStatus;

    }

    // Getter Methods (Accessor Methods) for attributes

    public int getTeacherID() {
```

```
    return teacherID;

}

public String getTeacherName() {

    return teacherName;

}

public String getAddress() {

    return address;

}

public String getWorkingType() {

    return workingType;

}

public String getEmploymentStatus() {

    return employmentStatus;

}

public int getWorkingHours() {

    return workingHours;

}

// Setter Method

public void setWorkingHours(int workingHours) {

    this.workingHours = workingHours;

}

// Method to display the details of the teacher

public void display() {
        System.out.println("Teacher ID: " + teacherID);

        System.out.println("Teacher Name: " + teacherName);

        System.out.println("Address: " + address);

        System.out.println("Working Type: " + workingType);

        System.out.println("Employment Status: " + employmentStatus);

        if (workingHours > 0) {

            System.out.println("Working Hours: " + workingHours);

        } else {

            System.out.println("Working Hours: Not assigned");

        }

    }

}
```

```
}
```

For Lecturer:

```
// Lecturer class

public class Lecturer extends Teacher {

   private String department;

   private int yearsOfExperience;

   private int gradedScore;

   private boolean hasGraded;

   public Lecturer(int teacherId, String teacherName, String address, String
workingType, String employmentStatus, String department, int yearsOfExperience, int
gradedScore) {

      super(teacherId, teacherName, address, workingType, employmentStatus);

      this.department = department;

      this.yearsOfExperience = yearsOfExperience;

      this.gradedScore = gradedScore;

      this.hasGraded = false;

   }

   // Getter Methods for attributes

   public String getDepartment() {

      return department;

   }

   public int getYearsOfExperience() {

      return yearsOfExperience;

   }
```

```
    public int getGradedScore() {
        return gradedScore;
    }
    public boolean getHasGraded() {
        return hasGraded;
    }
    // Method to grade assignments
    public String gradeAssignment(int score, String department, int yearsOfExperience) {
        if (!getHasGraded()) {
            if (this.department.equals(department) && this.yearsOfExperience >=
yearsOfExperience) {
                gradedScore = score;
                if (score >= 90) {
                    hasGraded = true;
                    return "Result: A";
                } else if (score >= 80) {
                    hasGraded = true;
                    return "Result: B";
                } else if (score >= 70) {
                    hasGraded = true;
                    return "Result: C";
                } else if (score >= 60) {
                    hasGraded = true;
        }

                    return "Result: D";

                } else {

                    return "Result: E";

                }

            } else {

                return "Unable to grade assignments at this time.";

            }

        } else {

            return "Assignment already graded.";

        }

    }

    // Method to display the details of the lecturer

    @Override

    public void display() {

        super.display();

        System.out.println("Department: " + department);

        System.out.println("Years of Experience: " + yearsOfExperience);

        if (getHasGraded()) {

            System.out.println("Graded Score: " + gradedScore);

        } else {

            System.out.println("Graded Score: Not available");

        }

    }
```

```
}
```

For Tutor:

```
//Class Tutor is a child of Teacher class

//creating child class of teacher class

public class Tutor extends Teacher{

   private int workingHours;

   private double salary;

   private String specialization;

   private String academicQualifications;

   private int performanceIndex;

   private boolean isCertified;

    //using constructor for Tutor

    public Tutor(int teacherId, String teacherName, String address, String workingType,
String employmentStatus,

    int workingHours, double salary, String specialization, String academicQualifications,
int performanceIndex) {

        super(teacherId, teacherName, address, workingType, employmentStatus);

        this.workingHours = workingHours;

        this.salary = salary;

        this.specialization = specialization;

        this.academicQualifications = academicQualifications;

        this.performanceIndex = performanceIndex;

        this.isCertified = false;
    super.setWorkingHours(workingHours);// set working hours for Tutor object

}

//Using Getter Method (accessor method)

public int getWorkingHours() {

    return workingHours;

}

public double getSalary(){

    return salary;

}

public String getSpecialization(){

    return specialization;

}

public String getAcademicQualifications(){

    return academicQualifications;

}

public int getPerformanceIndex(){

    return performanceIndex;

}

public boolean getIsCertified(){

    return isCertified;

}

//Using Setter Method (mutator method)

     public String setSalary(double newSalary, int newPerformanceIndex) {
        if (!isCertified && newPerformanceIndex > 5 && workingHours > 20) {
            double appraisalPercentage;
            if (newPerformanceIndex >= 5 && newPerformanceIndex <= 7) {
                appraisalPercentage = 5;
            } else if (newPerformanceIndex >= 8 && newPerformanceIndex <= 9) {
                appraisalPercentage = 10;
            } else { // newPerformanceIndex is 10
                appraisalPercentage = 20;
            }
            double appraisal = (appraisalPercentage / 100) * salary;
            salary += appraisal + newSalary - salary;
            performanceIndex = newPerformanceIndex;
            isCertified = true;
            return "Salary approved for " + getTeacherName() + ". New salary: " + salary;
        } else {
            return "Salary cannot be approved at this time for " + getTeacherName();
        }
    }
      public void removeTutor() {
        if (!isCertified) {
            salary = 0;
            specialization = "";
            academicQualifications = "";
            performanceIndex = 0;

            isCertified = false;

            System.out.println("Tutor removed successfully.");

        } else {

            System.out.println("Cannot remove certified tutor.");

        }

        }

        //Display method

       @Override

       public void display() {

        super.display();

        System.out.println("Specialization: " + specialization);

        System.out.println("Academic Qualifications: " + academicQualifications);

        System.out.println("Performance Index: " + performanceIndex);

        System.out.println("Salary: " + salary);

}

}
```

For teacherGUI:

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.Iterator;


public class teacherGUI {

    private ArrayList<Teacher> teachers = new ArrayList<>();


    public void GUI() {


        JFrame frame = new JFrame("Coursework GUI");

        frame.setVisible(true);

        frame.setSize(700, 700);

        frame.setLayout(null);


        // FOR LECTURER

        JLabel lec = new JLabel("Lecturer");

        lec.setBounds(700, 10, 200, 130);

        lec.setFont(new Font("Times New Roman", Font.PLAIN, 30));

        frame.add(lec);
```

```
JLabel TeaIDa = new JLabel("Teacher ID:");

TeaIDa.setBounds(30, 150, 100, 50);

frame.add(TeaIDa);


JTextField TeaIDa1 = new JTextField();

TeaIDa1.setBounds(175, 160, 100, 30);

frame.add(TeaIDa1);


JLabel TeaNa = new JLabel("Teacher Name:");

TeaNa.setBounds(30, 200, 100, 50);

frame.add(TeaNa);


JTextField TeaNa1 = new JTextField();

TeaNa1.setBounds(175, 210, 100, 30);

frame.add(TeaNa1);


JLabel TeaADDa = new JLabel("Teacher Address:");

TeaADDa.setBounds(30, 250, 100, 50);

frame.add(TeaADDa);


JTextField TeaADDa1 = new JTextField();

TeaADDa1.setBounds(175, 260, 100, 30);
```

```
frame.add(TeaADDa1);


JLabel YOEa = new JLabel("Years of Experience:");

YOEa.setBounds(30, 300, 150, 50);

frame.add(YOEa);


JTextField YOEa1 = new JTextField();

YOEa1.setBounds(175, 310, 100, 30);

frame.add(YOEa1);


JLabel WorTa = new JLabel("Working Type:");

WorTa.setBounds(1250, 150, 100, 50);

frame.add(WorTa);


JTextField WorTa1 = new JTextField();

WorTa1.setBounds(1400, 160, 100, 30);

frame.add(WorTa1);


JLabel EmpSa = new JLabel("Employment Status:");

EmpSa.setBounds(1250, 200, 150, 50);

frame.add(EmpSa);


JTextField EmpSa1 = new JTextField();
```

```
EmpSa1.setBounds(1400, 210, 100, 30);

frame.add(EmpSa1);


JLabel GraSa = new JLabel("Graded Score:");

GraSa.setBounds(1250, 250, 100, 50);

frame.add(GraSa);


JTextField GraSa1 = new JTextField();

GraSa1.setBounds(1400, 260, 100, 30);

frame.add(GraSa1);


JLabel Depa = new JLabel("Department:");

Depa.setBounds(1250, 300, 100, 50);

frame.add(Depa);


JTextField Depa1 = new JTextField();

Depa1.setBounds(1400, 310, 100, 30);

frame.add(Depa1);


JButton disa = new JButton("Display");

disa.setBounds(450, 200, 100, 30);

disa.setBackground(new Color(135, 206, 235));

frame.add(disa);
```

```
JButton clea = new JButton("Clear");

clea.setBounds(450, 250, 100, 30);

clea.setBackground(Color.RED);

frame.add(clea);


JButton AddLa = new JButton("Add Lecturer");

AddLa.setBounds(900, 200, 150, 30);

AddLa.setBackground(Color.GREEN);

frame.add(AddLa);


JButton GraAa = new JButton("Grade Assignments");

GraAa.setBounds(900, 250, 150, 30);

GraAa.setBackground(Color.GREEN);

frame.add(GraAa);


// FOR TUTOR

JLabel Tut = new JLabel("Tutor");

Tut.setBounds(700, 350, 200, 130);

Tut.setFont(new Font("Times New Roman", Font.PLAIN, 30));

frame.add(Tut);

JLabel TeaIDb = new JLabel("Teacher ID:");

TeaIDb.setBounds(30, 500, 100, 50);
```

```
frame.add(TeaIDb);

JTextField TeaIDb1 = new JTextField();

TeaIDb1.setBounds(200, 510, 100, 30);

frame.add(TeaIDb1);

JLabel TeaNb = new JLabel("Teacher Name:");

TeaNb.setBounds(30, 550, 100, 50);

frame.add(TeaNb);

JTextField TeaNb1 = new JTextField();

TeaNb1.setBounds(200, 560, 100, 30);

frame.add(TeaNb1);

JLabel TeaAb = new JLabel("Teacher Address:");

TeaAb.setBounds(30, 600, 100, 50);

frame.add(TeaAb);

JTextField TeaAb1 = new JTextField();

TeaAb1.setBounds(200, 610, 100, 30);

frame.add(TeaAb1);


JLabel AcaQa = new JLabel("Academic Qualifications:");

AcaQa.setBounds(30, 650, 150, 50);

frame.add(AcaQa);

JTextField AcaQa1 = new JTextField();

AcaQa1.setBounds(200, 660, 100, 30);

frame.add(AcaQa1);
```

```
JLabel PerIa = new JLabel("Perfomance Index:");

PerIa.setBounds(30, 700, 150, 50);

frame.add(PerIa);

JTextField PerIa1 = new JTextField();

PerIa1.setBounds(200, 710, 100, 30);

frame.add(PerIa1);

JLabel NeSala = new JLabel("New Salary:");

NeSala.setBounds(30, 750, 150, 50);

frame.add(NeSala);

JTextField NeSala1 = new JTextField();

NeSala1.setBounds(200, 760, 100, 30);

frame.add(NeSala1);


JLabel Spea = new JLabel("Specialization:");

Spea.setBounds(1250, 500, 150, 50);

frame.add(Spea);

JTextField Spea1 = new JTextField();

Spea1.setBounds(1400, 510, 100, 30);

frame.add(Spea1);

JLabel WTa = new JLabel("Working Type:");

WTa.setBounds(1250, 550, 100, 50);

frame.add(WTa);

JTextField WTa1 = new JTextField();
```

```
WTa1.setBounds(1400, 560, 100, 30);

frame.add(WTa1);

JLabel EmpSb = new JLabel("Employment Status:");

EmpSb.setBounds(1250, 600, 150, 50);

frame.add(EmpSb);

JTextField EmpSb1 = new JTextField();

EmpSb1.setBounds(1400, 610, 100, 30);

frame.add(EmpSb1);

JLabel WorHa = new JLabel("Working Hours:");

WorHa.setBounds(1250, 650, 100, 50);

frame.add(WorHa);

JTextField WorHa1 = new JTextField();

WorHa1.setBounds(1400, 660, 100, 30);

frame.add(WorHa1);

JLabel Salb = new JLabel("Salary:");

Salb.setBounds(1250, 700, 100, 50);

frame.add(Salb);

JTextField Salb1 = new JTextField();

Salb1.setBounds(1400, 710, 100, 30);

frame.add(Salb1);

JLabel NPIa = new JLabel("New Perfomance Index:");

NPIa.setBounds(1250, 750, 150, 50);

frame.add(NPIa);
```

```
JTextField NPI1 = new JTextField();

NPI1.setBounds(1400, 760, 100, 30);

frame.add(NPI1);

JButton Disb = new JButton("Display");

Disb.setBounds(450, 550, 100, 30);

Disb.setBackground(new Color(135, 206, 235));

frame.add(Disb);

JButton Cleb = new JButton("Clear");

Cleb.setBounds(450, 650, 100, 30);

Cleb.setBackground(Color.RED);

frame.add(Cleb);

JButton AddTa = new JButton("Add Tutor");

AddTa.setBounds(950, 550, 120, 30);

AddTa.setBackground(Color.GREEN);

frame.add(AddTa);

JButton RemTa = new JButton("Remove Tutor");

RemTa.setBounds(950, 600, 120, 30);

RemTa.setBackground(Color.RED);

frame.add(RemTa);


JButton SetSa = new JButton("Set Salary");

SetSa.setBounds(950, 650, 120, 30);

SetSa.setBackground(Color.GREEN);
```

```
        frame.add(SetSa);
        AddLa.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                try {
                    if (TeaIDa1.getText().isEmpty() || TeaNa1.getText().isEmpty() ||
TeaADDa1.getText().isEmpty() || YOEa1.getText().isEmpty() ||
WorTa1.getText().isEmpty() || EmpSa1.getText().isEmpty() ||
GraSa1.getText().isEmpty() || Depa1.getText().isEmpty()) {
                        JOptionPane.showMessageDialog(null, "Fill all the fields please!");
                    } else {
                        Lecturer l = new Lecturer(Integer.parseInt(TeaIDa1.getText()),
TeaNa1.getText(), TeaADDa1.getText(), WorTa1.getText(), EmpSa1.getText(),
Depa1.getText(), Integer.parseInt(YOEa1.getText()),
Integer.parseInt(GraSa1.getText()));
                        teachers.add(l);
                        JOptionPane.showMessageDialog(null, "Added to Lecturer");
                    }
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(null, "Enter the valid values for integers
type");
                }
            }
        });
        AddTa.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                try {

                    if (TeaIDb1.getText().isEmpty() || TeaNb1.getText().isEmpty() ||
TeaAb1.getText().isEmpty() || AcaQa1.getText().isEmpty() || PerIa1.getText().isEmpty()
|| Salb1.getText().isEmpty() || Spea1.getText().isEmpty() || WTa1.getText().isEmpty() ||
EmpSb1.getText().isEmpty() || WorHa1.getText().isEmpty()) {

                        JOptionPane.showMessageDialog(null, "Fill all the fields please!");

                    } else {

                        Tutor T = new Tutor(Integer.parseInt(TeaIDb1.getText()),
TeaNb1.getText(), TeaAb1.getText(), WTa1.getText(),
EmpSb1.getText(),Integer.parseInt( WorHa1.getText())
,Double.parseDouble(Salb1.getText()), Spea1.getText(), AcaQa1.getText(),
Integer.parseInt(PerIa1.getText()));

                        teachers.add(T);

                        JOptionPane.showMessageDialog(null, "Added to tutor");

                    }

                } catch (NumberFormatException ex) {

                    JOptionPane.showMessageDialog(null, "Enter the valid values for integers
type");

                }

            }

        });

        disa.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent ae) {

                if (TeaIDa1.getText().isEmpty() || TeaNa1.getText().isEmpty() ||
TeaADDa1.getText().isEmpty() || YOEa1.getText().isEmpty() ||
WorTa1.getText().isEmpty() || EmpSa1.getText().isEmpty() ||
GraSa1.getText().isEmpty() || Depa1.getText().isEmpty()) {
                    JOptionPane.showMessageDialog(null, "Fill all the fields please!");
                } else {
                    for (Teacher l : teachers) {
                        if (l instanceof Lecturer) {
                            l.display();
                        }
                    }
                }
            }
        });
        Disb.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                if (TeaIDb1.getText().isEmpty() || TeaNb1.getText().isEmpty() ||
TeaAb1.getText().isEmpty() || AcaQa1.getText().isEmpty() || PerIa1.getText().isEmpty()
|| Salb1.getText().isEmpty() || Spea1.getText().isEmpty() || WTa1.getText().isEmpty() ||
EmpSb1.getText().isEmpty() || WorHa1.getText().isEmpty()) {
                    JOptionPane.showMessageDialog(null, "Fill all the fields please!");
                } else {
                    for (Teacher T : teachers) {
                        if (T instanceof Tutor) {
                            T.display();
                        }
                    }
                }
            }
        });
        clea.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                TeaIDa1.setText("");
                TeaNa1.setText("");
                TeaADDa1.setText("");
                YOEa1.setText("");
                WorTa1.setText("");
                EmpSa1.setText("");
                GraSa1.setText("");
                Depa1.setText("");
            }
        });
        Cleb.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                TeaIDb1.setText("");
                TeaNb1.setText("");
                TeaAb1.setText("");
                AcaQa1.setText("");
                PerIa1.setText("");
```

```
            Salb1.setText("");
            Spea1.setText("");
            WTa1.setText("");
            EmpSb1.setText("");
            WorHa1.setText("");
        }
    });
    GraAa.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            try {
                int teacherId = Integer.parseInt(TeaIDa1.getText());
                int gradedScore = Integer.parseInt(GraSa1.getText());
                String department = Depa1.getText();
                int yearsOfExperience = Integer.parseInt(YOEa1.getText());
                for (Teacher teacher : teachers) {
                    if (teacher instanceof Lecturer && teacher.getTeacherID() == teacherId) {
                        Lecturer lecturer = (Lecturer) teacher; // downcasting
                        String grade = lecturer.gradeAssignment(gradedScore, department,
yearsOfExperience);
                        JOptionPane.showMessageDialog(frame, "Graded assignment result:
" + grade);
                        return;
                    }
                }

                JOptionPane.showMessageDialog(frame, "Lecturer with ID " + teacherId +
" not found!");

            } catch (NumberFormatException ex) {

                JOptionPane.showMessageDialog(frame, "Please enter valid numeric
values for Teacher ID, Graded Score, Years of Experience.");

            }

        }

    });

    RemTa.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            try {

                // Get input value from text field

                int TeacherID = Integer.parseInt(TeaIDb1.getText());

                // Find and remove the matching tutor

                boolean removed = false;

                for (Iterator<Teacher> iterator = teachers.iterator(); iterator.hasNext(); ) {

                    Teacher teacher = iterator.next();

                    if (teacher instanceof Tutor && teacher.getTeacherID() == TeacherID) {

                        iterator.remove();

                        removed = true;

                        // Display success message
                    JOptionPane.showMessageDialog(frame, "Tutor removed
successfully!");
                    break;
                  }
                }
                // Display error message if tutor not found
                if (!removed) {
                  JOptionPane.showMessageDialog(frame, "Tutor with ID " + TeacherID +
" not found!");
                }
              } catch (NumberFormatException ex) {
                // Display error message for invalid input
                JOptionPane.showMessageDialog(frame, "Only integers values is
accepted for  Teacher ID.");
              }
            }
          });
        // ActionListener for Set Salary button
SetSa.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        try {
            if (TeaIDb1.getText().isEmpty() || PerIa1.getText().isEmpty() ||
NeSala1.getText().isEmpty()) {
```

```
                JOptionPane.showMessageDialog(null, "Please enter a valid ID, Performance
Index, and New Salary.");

            } else {

                int tutorId = Integer.parseInt(TeaIDb1.getText());

                int newPerformanceIndex = Integer.parseInt(NPI1.getText());

                double newSalary = Double.parseDouble(NeSala1.getText());

                // Updating the tutor's salary and performance index

               for (Teacher teacher : teachers) {

                          if (teacher instanceof Tutor && teacher.getTeacherID() == tutorId) {

                              Tutor tutor = (Tutor) teacher;

                              String message = tutor.setSalary(newSalary,
newPerformanceIndex);

                              JOptionPane.showMessageDialog(null, message);

                              break;

                          }

                      }

                  }

                } catch (NumberFormatException ex) {

                    JOptionPane.showMessageDialog(frame, "Only integers values are
accepted for ID, Performance Index, and New Salary.");

                }

            }

        });
```

```
    }

    public static void main(String[] args) {

        teacherGUI teacherGUI = new teacherGUI();

        teacherGUI.GUI();

    }

}
```