Module Code & Module Title

CS4001NI Programming

COURSEWORK-2

Assessment Weightage & Type

30% Individual Coursework

Year and Semester

Autumn Semester 2023-2024

Student Name: Bhumika Karki

Group: C2

London Met ID: 23047584

College ID: NP01CP4A230031

Assignment Due Date: 10, May 2024

Assignment Submission Date: 9, May 2024

Contents

Table of Figures

Table of Tables

1. Introduction

The following coursework comprises a Java program, which is based on our first coursework where the students were intended to develop a real-life time scenario with the main class of Teacher and its sub classes: Lecturer class and Tutor class with the principle of object-orientation. Now considering them as a reference, we had to create a graphical user interface to apply them and add the data to the Teacher class's array list. Here, the guidelines were carefully followed in order to ensure the smooth launch and operation of the program. The codes from the first coursework were the same, the usage of the methods was much helpful in the second course as well. After that, the GUI was created in another class using several swing concepts, like JLabels, JTextFields, JButtons, and many more. Further on the functionality of the buttons were then modified by event handling and exception handling ideas. Each button was assigned a certain function based on the inquiry, such as adding, grading assignments, setting salaries, and displaying and clearing data based on the text entered into the designated text fields.

1.1. Goals and Objectives

The coursework requires the use of Java components and its package in order to properly complete and answer the questions. It focuses on dealing with an actual situation involving the addition and removal of teachers and coding it to produce the intended result. The following are the goals and objectives of the coursework:

- Utilize the object-oriented concept of Java to a real-world scenario. This includes designing a class to represent a teacher, as well as two subclasses to represent Lecturer and Tutor, respectively.
- Include the teacher's input into the project, where the initial part of the course work focused on creating a graphical user interface (GUI) using those ideas.

- Keep the Teacher class's Array List updated with the necessary requirements, characteristics, and elements required for Lecturer and Tutor.
- Utilization of the majority of Java's Swing components to develop a window-based application, including JLabels, JTextFields, and JButtons, as well as the addition of label styles via the Abstract Window Toolkit.
- To make sure the program runs correctly, implement event handling and exception handling. This includes displaying the relevant message dialogue boxes when a button is pressed or throwing an exception when an error is detected.
- Create an Array List for the Teacher parent class in which all the properties and details of Lecturer and Tutor child class objects are stored.

1.2. Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers write once, and run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java was first released in 1995 and is widely used for developing applications for desktop, web, and mobile devices. Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications. Java is a versatile and widely used object-oriented programming language known for its platform independence and robustness. Developed by Sun Microsystems (now part of Oracle Corporation) in the mid-1990s, Java's "Write Once, Run Anywhere" principle allows it to run on any system with a Java Virtual Machine (JVM). Its object-oriented nature promotes code reusability and maintainability, while its extensive standard library simplifies development tasks. Java's emphasis on security and regular updates makes it a top choice for various applications (GeeksforGeeks, 2024).

1.2.1. Swing

Java Swing is a fundamental framework of Java used to create window-based applications. It’s an extension of the Abstract Windowing Toolkit (AWT) and provides capabilities to design GUI (Graphical User Interface) based desktop applications in Java. The Swing library provides a rich set of widgets and packages to deliver complex components like tables, lists, scroll panes, colour chooser, and tabbed panels, effectively assisting the developers to create comprehensive graphic user interfaces. Java Swing proves to be very impressive with its flexibility of configuration and scalability for complex visual representations. It empowers developers to create highly interactive and customizable interfaces that smoothly integrate with the underlying data models. It’s extensively used to facilitate an intuitive interaction between the user and the application, delivering a better user experience. Swing’s capabilities support more than just creating and managing windows, it extends to handling events, laying out components, customizing the look and feel, and managing threads that ensure a responsive user interface (DevX, 2023).

1.2.2. Event Handling

Event handling in Java refers to the procedure that manages events and triggers appropriate actions in response to those events. It involves implementing event handlers, which are sets of instructions that execute specific actions when events occur. Event handling in Java consists of two main components: the event source, where the event originates, and the event listener, responsible for responding to events. Listeners must be registered with the source object to receive event notifications. Events in Java represent changes in the state of objects, typically occurring when users interact with the graphical user interface (GUI). Types of events in Java include foreground events, requiring direct user interaction with GUI components, and background events, such as system interrupts or hardware failures. Overall, event handling in Java is the process of controlling an event and taking appropriate action if one occurs (Scaler, 2023).

1.2.3. Exception Handling

Exception handling in Java is a crucial technique used to manage unexpected events, known as exceptions, during program execution. These exceptions can disrupt the normal flow of the program and potentially harm its performance. Java's exception handling mechanism, which employs the "try" and "catch" keywords, is designed to address these issues effectively. The "try" block encloses code that may throw an exception, while the "catch" block handles these exceptions to ensure that the program continues to run smoothly. By implementing proper exception handling, Java applications can maintain their reliability and performance even in the face of runtime faults (Simplilearn, 2024).

1.3. Tools Used

A range of tools was employed throughout the coursework, from inception to completion, to gain a distinct advantage. The primary responsibilities included coding, involving writing and compiling code within an IDE, and presenting the coursework with sufficient depth. The following terms describe the applications of these tools:

1.3.1. Blue J

BlueJ is an IDE designed for Java programming, especially for educational use. It's known for its user-friendly interface and essential features, making it ideal for beginners. While it lacks some advanced features found in other IDEs like Eclipse, it stands out for its simplicity and reliability. BlueJ simplifies project creation and includes debugging tools, although it may have minor stability issues. Overall, it's a solid choice for learners and new developers, but advanced users may find it lacking for complex projects (softonic, 2023).

1.3.2. MS Word

Microsoft Word is a graphical word processing software developed by Microsoft Corporation. It's designed to create, edit, and format text-based documents easily. As part of the Microsoft Office suite, it offers a user-friendly interface and a wide range of tools for tasks like text formatting, spell check, grammar check, template usage, graphics insertion, collaboration, clip art, and document management. It features is its ability to handle various document types, from letters to reports and resumes. It's widely popular as the most-used word processor globally, thanks to its availability on almost every Windows computer, making it easily accessible. Microsoft Word allows users to save documents in its format for convenient sharing and portability via USB flash drives. It also supports creating different formats like business correspondence, business cards, brochures, and newsletters, making it a preferred choice for individuals and businesses alike due to its ease of use and comprehensive feature set (LinkedIn, n.d.).

1.3.3. draw.io

Draw.io is free online diagram software that can be used to make a flowchart, network diagram, UML (Unified Modelling Language), ER diagram, to design database schema, to build BPMN, to build circuit diagram and more.

draw.io is proprietary software for making diagrams and charts. The software lets you choose from an automatic layout function or create a custom layout. They have a large selection of shapes and hundreds of visual elements to make your diagram or chart one-of-a-kind. The drag-and-drop feature makes it simple to create a great looking diagram or chart (Computer, 2020).

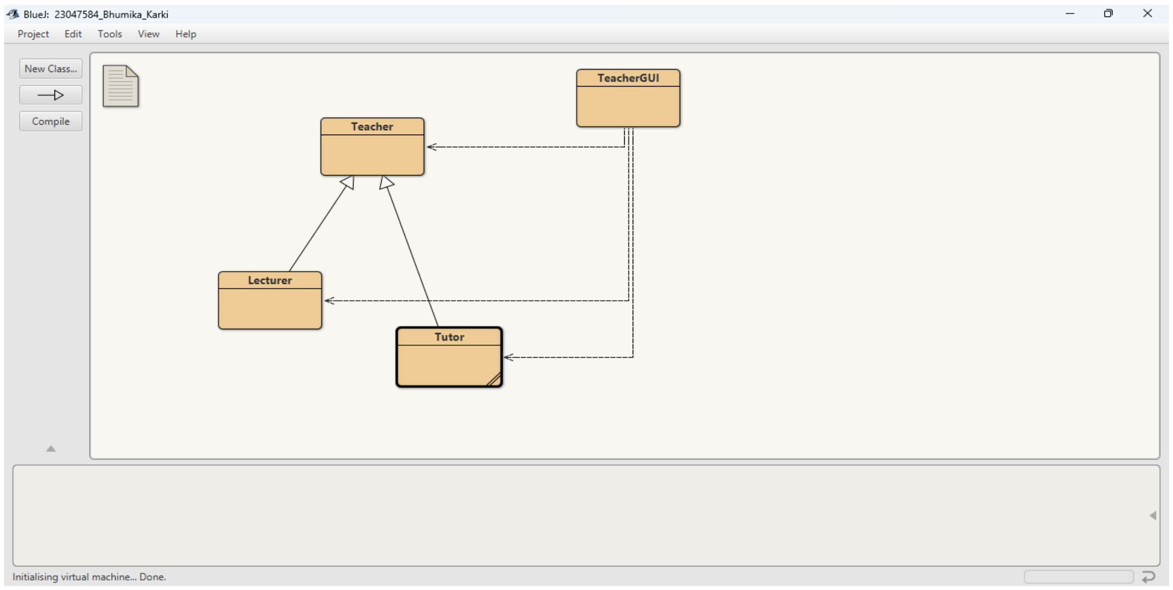
2. Class Diagram

A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram shows the building blocks of any object-oriented system. Class diagrams depict a static view of the model, or part of the model, describing what attributes and behaviour it has rather than detailing the methods for achieving operations. Class diagrams are most useful in illustrating relationships between classes and interfaces. Generalizations, aggregations, and associations are all valuable in reflecting inheritance, composition, or usage, and connections respectively (Systems, n.d.).

The class diagram's purpose can be stated as follows:

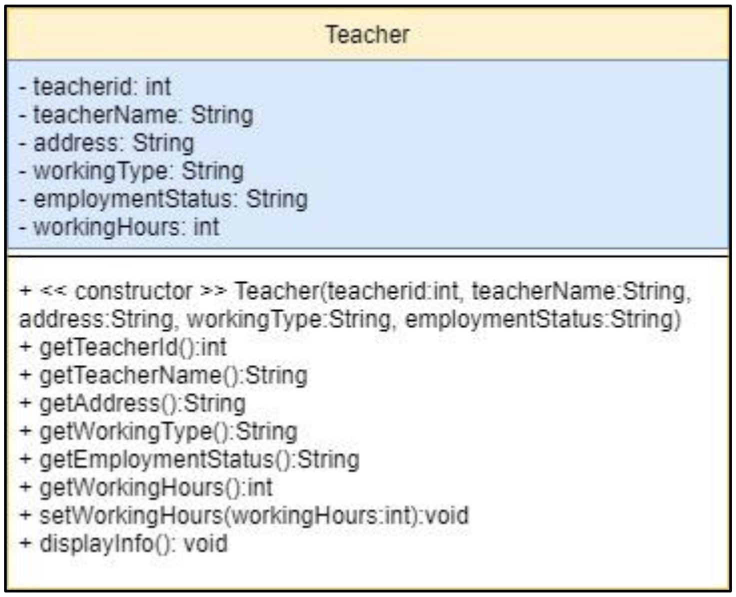
- Shows static structure of classifiers in a system.
- Diagram provides a basic notation for other structure diagrams prescribed by UML.
- Helpful for developers and other team members too
- Business Analysts can use class diagrams to model systems from a business perspective.



Initialising virtual machine... Done.

Figure 1:Class Diagram

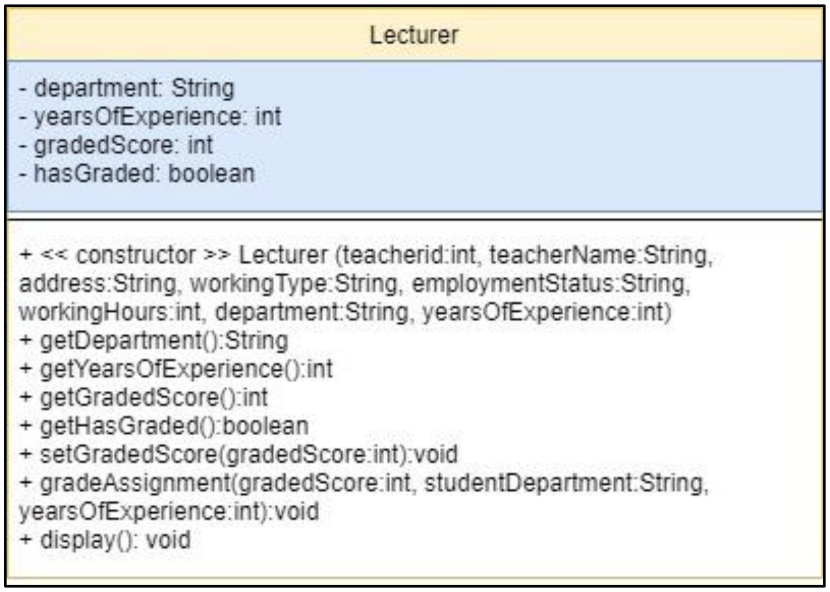
2.1. Teacher Class



```
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)
+ getTeacherId():int
+ getTeacherName():String
+ getAddress():String
+ getWorkingType():String
+ getEmploymentStatus():String
+ getWorkingHours():int
+ setWorkingHours(workingHours:int):void
+ displayInfo(): void
```

Figure 2: Class Diagram of Teacher Class

2.2. Lecturer Class



```
Lecturer
- department: String
- yearsOfExperience: int
- gradedScore: int
- hasGraded: boolean
+ << constructor >> Lecturer (teacherid:int, teacherName:String,
address:String, workingType:String, employmentStatus:String,
workingHours:int, department:String, yearsOfExperience:int)
+ getDepartment():String
+ getYearsOfExperience():int
+ getGradedScore():int
+ getHasGraded():boolean
+ setGradedScore(gradedScore:int):void
+ gradeAssignment(gradedScore:int, studentDepartment:String,
yearsOfExperience:int):void
+ display(): void
```

Figure 3: Class Diagram for Lecturer Class

2.3. Tutor Class

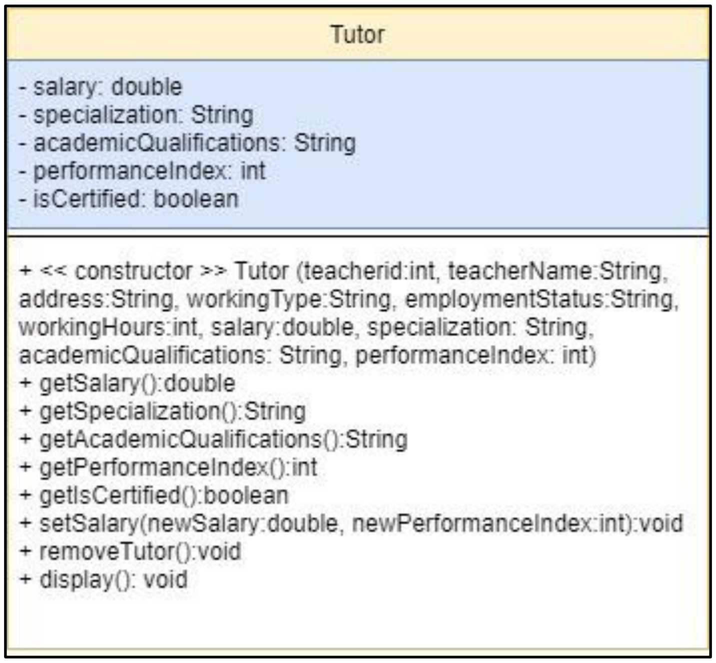
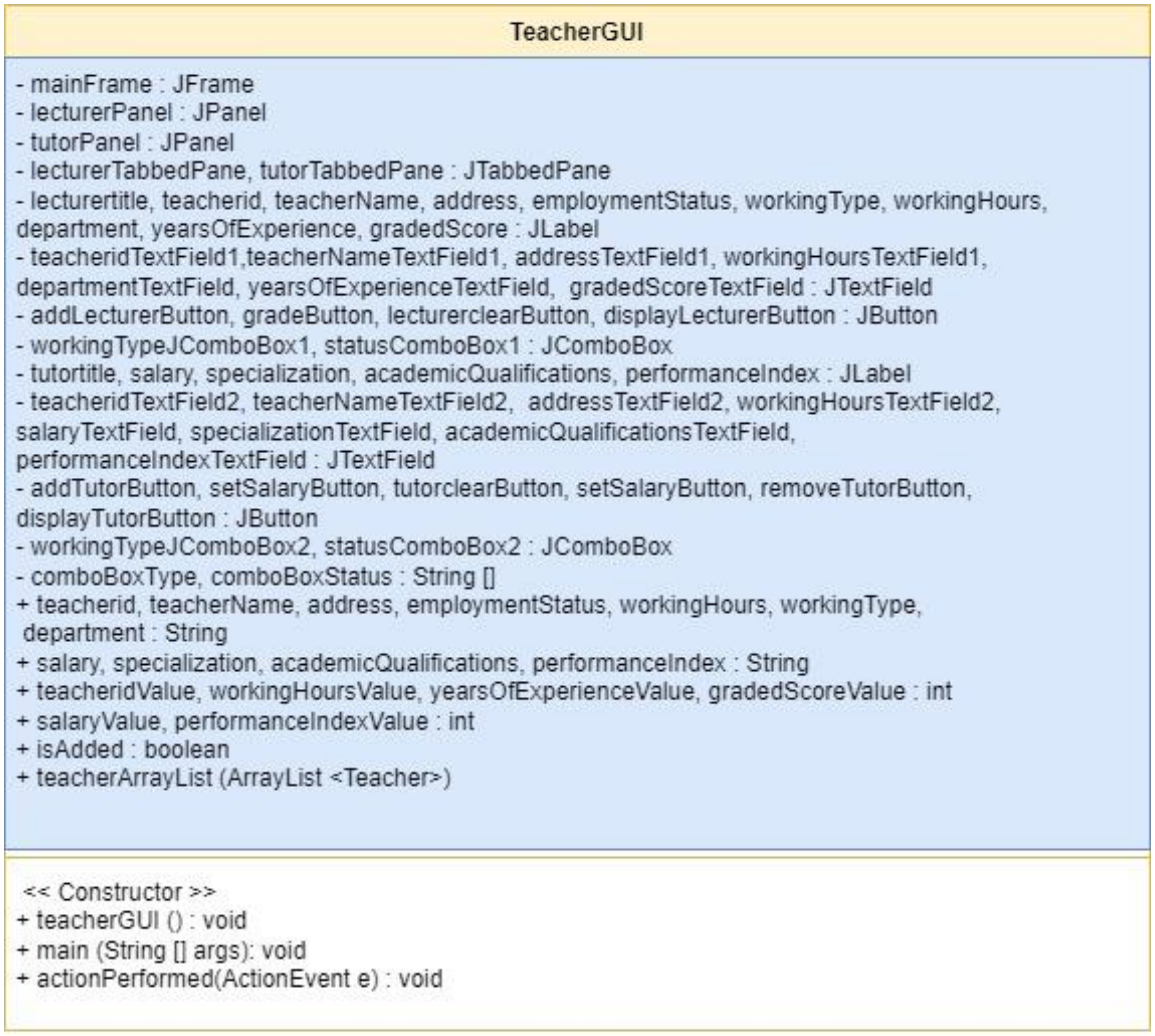

Tutor
- salary: double
- specialization: String
- academicQualifications: String
- performanceIndex: int
- isCertified: boolean
+ << constructor >> Tutor (teacherid:int, teacherName:String, address:String, workingType:String, employmentStatus:String, workingHours:int, salary:double, specialization: String, academicQualifications: String, performanceIndex: int)
+ getSalary():double
+ getSpecialization():String
+ getAcademicQualifications():String
+ getPerformanceIndex():int
+ getIsCertified():boolean
+ setSalary(newSalary:double, newPerformanceIndex:int):void
+ removeTutor():void
+ display(): void

Figure 4: Class Diagram of Tutor Class

2.4. TeacherGUI Class



TeacherGUI

- mainFrame : JFrame
- lecturerPanel : JPanel
- tutorPanel : JPanel
- lecturerTabbedPane, tutorTabbedPane : JTabbedPane
- lecturertitle, teacherid, teacherName, address, employmentStatus, workingType, workingHours, department, yearsOfExperience, gradedScore : JLabel
- teacheridTextField1,teacherNameTextField1, addressTextField1, workingHoursTextField1, departmentTextField, yearsOfExperienceTextField, gradedScoreTextField : JTextField
- addLecturerButton, gradeButton, lecturerclearButton, displayLecturerButton : JButton
- workingTypeJComboBox1, statusComboBox1 : JComboBox
- tutortitle, salary, specialization, academicQualifications, performanceIndex : JLabel
- teacheridTextField2, teacherNameTextField2, addressTextField2, workingHoursTextField2, salaryTextField, specializationTextField, academicQualificationsTextField, performanceIndexTextField : JTextField
- addTutorButton, setSalaryButton, tutorclearButton, setSalaryButton, removeTutorButton, displayTutorButton : JButton
- workingTypeJComboBox2, statusComboBox2 : JComboBox
- comboBoxType, comboBoxStatus : String []

+ teacherid, teacherName, address, employmentStatus, workingHours, workingType, department : String
+ salary, specialization, academicQualifications, performanceIndex : String
+ teacheridValue, workingHoursValue, yearsOfExperienceValue, gradedScoreValue : int
+ salaryValue, performanceIndexValue : int
+ isAdded : boolean
+ teacherArrayList (ArrayList <Teacher>)

<< Constructor >>
+ teacherGUI () : void
+ main (String [] args): void
+ actionPerformed(ActionEvent e) : void

Figure 5: Class Diagram for Class TeacherGUI

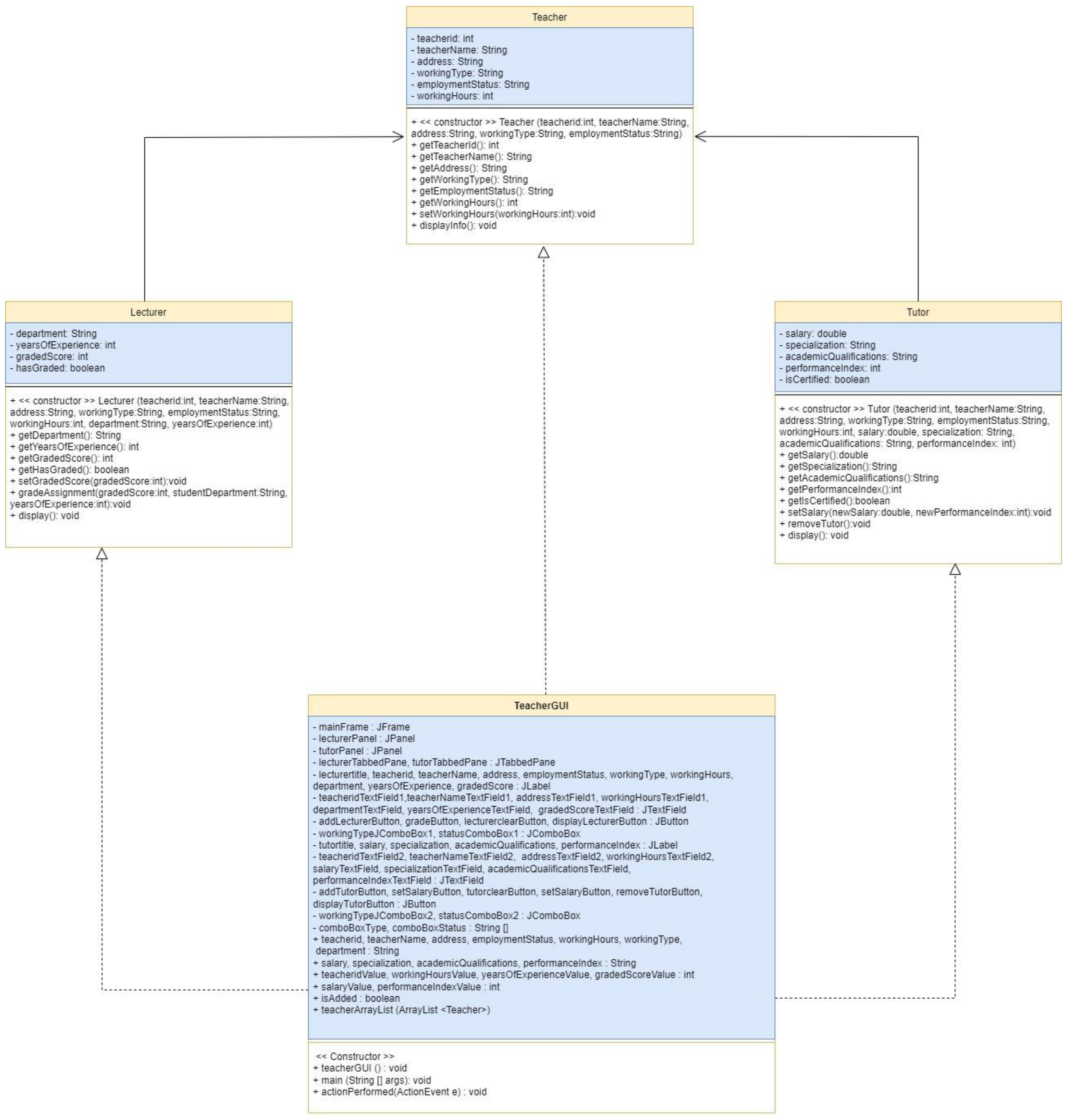

Teacher
- teacherid: int
- teacherName: String
- address: String
- workingType: String
- employmentStatus: String
- workingHours: int
+ << constructor >> Teacher (teacherid:int, teacherName:String, address:String, workingType:String, employmentStatus:String)
+ getTeacherId(): int
+ getTeacherName(): String
+ getAddress(): String
+ getWorkingType(): String
+ getEmploymentStatus(): String
+ getWorkingHours(): int
+ setWorkingHours(workingHours:int):void
+ displayInfo(): void
Lecturer
- department: String
- yearsOfExperience: int
- gradedScore: int
- hasGraded: boolean
+ << constructor >> Lecturer (teacherid:int, teacherName:String, address:String, workingType:String, employmentStatus:String, workingHours:int, department:String, yearsOfExperience:int)
+ getDepartment(): String
+ getYearsOfExperience(): int
+ getGradedScore(): int
+ getHasGraded(): boolean
+ setGradedScore(gradedScore:int):void
+ gradeAssignment(gradedScore:int, studentDepartment:String, yearsOfExperience:int):void
+ display(): void
Tutor
- salary: double
- specialization: String
- academicQualifications: String
- performanceIndex: int
- isCertified: boolean
+ << constructor >> Tutor (teacherid:int, teacherName:String, address:String, workingType:String, employmentStatus:String, workingHours:int, salary:double, specialization: String, academicQualifications: String, performanceIndex: int)
+ getSalary():double
+ getSpecialization():String
+ getAcademicQualifications():String
+ getPerformanceIndex():int
+ getIsCertified():boolean
+ setSalary(newSalary:double, newPerformanceIndex:int):void
+ removeTutor():void
+ display(): void
TeacherGUI
- mainFrame : JFrame
- lecturerPanel : JPanel
- tutorPanel : JPanel
- lecturerTabbedPane, tutorTabbedPane : JTabbedPane
- lecturertitle, teacherid, teacherName, address, employmentStatus, workingType, workingHours, department, yearsOfExperience, gradedScore : JLabel
- teacheridTextField1,teacherNameTextField1, addressTextField1, workingHoursTextField1, departmentTextField, yearsOfExperienceTextField, gradedScoreTextField : JTextField
- addLecturerButton, gradeButton, lecturerclearButton, displayLecturerButton : JButton
- workingTypeJComboBox1, statusComboBox1 : JComboBox
- tutortitle, salary, specialization, academicQualifications, performanceIndex : JLabel
- teacheridTextField2, teacherNameTextField2, addressTextField2, workingHoursTextField2, salaryTextField, specializationTextField, academicQualificationsTextField, performanceIndexTextField : JTextField
- addTutorButton, setSalaryButton, tutorclearButton, setSalaryButton, removeTutorButton, displayTutorButton : JButton
- workingTypeJComboBox2, statusComboBox2 : JComboBox
- comboBoxType, comboBoxStatus : String []
+ teacherid, teacherName, address, employmentStatus, workingHours, workingType, department : String
+ salary, specialization, academicQualifications, performanceIndex : String
+ teacheridValue, workingHoursValue, yearsOfExperienceValue, gradedScoreValue : int
+ salaryValue, performanceIndexValue : int
+ isAdded : boolean
+ teacherArrayList (ArrayList <Teacher>)
<< Constructor >>
+ teacherGUI () : void
+ main (String [] args): void
+ actionPerformed(ActionEvent e) : void

Figure 6: Inheritance Class Diagram

3. Pseudocode

Pseudocode is an informal language used by programmers to develop algorithms. It's text-based and helps in designing algorithms without getting into detailed implementation specifics. Key words like while, do, for, if, and switch are used in pseudocode, and statements are typically indented for clarity. It's widely used to create an initial overview or draft of a program, ensuring that the logic is sound before actual coding begins. Pseudocode focuses on the main flow of the program using simple English words, aiding in understanding without being executable code itself. It's a valuable technique for implementing algorithms as it enhances program comprehensibility. These are a rule that a coder should follow for pseudocode (University, 2020).

i. Write only one statement per line
ii. Capitalize initial keyword
iii. Indent to show hierarchy
iv. End multiline structures
v. Keep statement language independent

3.2. Teacher Class

```
CREATE a parent class Teacher
DO
    DECLARE instance variable teacherid as int using private modifier
    DECLARE instance variable teacherName as String using private modifier
    DECLARE instance variable address as String using private modifier
    DECLARE instance variable workingType as String using private modifier
    DECLARE instance variable employmentStatus as String using private modifier
    DECLARE instance variable workingHours as int using private modifier

CREATE constructor of Teacher class and PASS the required parameters
DO
INITIALIZE the value of teacherName as teacherName
INITIALIZE the value of teacherid as teacherid
INITIALIZE the value of address as address
INITIALIZE the value of workingType as workingType
INITIALIZE the value of employmentStatus as employmentStatus

CREATE an accessor method getTeacherId() with return type int
    DO
        RETURN teacherid
    END DO



CREATE an accessor method getTeacherName() with return type String
    DO
        RETURN teacherName
    END DO
CREATE an accessor method getAddress() with return type String
    DO
        RETURN address
    END DO

CREATE an accessor method getWorkingType () with return type String
    DO
        RETURN workingType
    END DO

CREATE an accessor method getEmploymanetStatus () with return type String
    DO
        RETURN employmentStatus
    END DO

CREATE an accessor method getWorkingHours () with return type int
    DO
        RETURN workingHours
    END DO

CREATE method setWorkingHours (PASS parameter workingHours as int)
    DO
            SET the value of workingHours to workingHours
END DO

CREATE method displayInfo () with return type void
    DO
```

```
        PRINT the value of teacherid
        PRINT the value of teacherName
        PRINT the value of address
        PRINT the value of workingType
        PRINT the value of employmentStatus
        IF workingHours is empty
                PRINT a suitable message
        END IF
        ELSE
                PRINT workingHours
        END DO
END DO
```

3.2. Lecturer Class

```
CREATE class Lecturer EXTENDED FROM super class Teacher
DO
        DECLARE instance variable department as String using private modifier
        DECLARE instance variable yearsOfExperience as intusing private modifier
        DECLARE instance variable gradedScore as int using private modifier
        DECLARE instance variable hasGraded as Boolean using private modifier

CREATE the constructor Lecturer Class and PASS the required parameters

CALL parameters from parent class Teacher using super keyword

        INITIALIZE the value of department to department
        INITIALIZE the value of yearsOfExperience to yearsOfExperience
        INITIALIZE the value of gradedScore to zero
```

```
    **INITIALIZE** the value of hasGraded to false

**CREATE** an accessor method getDepartment () with return type String
    **DO**
        RETURN department
    **END DO**

**CREATE** an accessor method get YearsOfExperience () with return type int
    **DO**
        RETURN yearsOfExperience
    **END DO**

**CREATE** an accessor method getGradedScore () with return type int
    **DO**
        RETURN gradedScore
    **END DO**

**CREATE** an accessor method get hasGraded () with return type boolean
    **DO**
        RETURN hasGraded
    **END DO**

**CREATE** method setGradedScore (PASS parameter gradedScore as int)
    **DO**
        **SET** the value of gradedScore to gradedScore
**END DO**
```

```
CREATE method gradeAssignment and PASS parameters with return type void
DO
    IF (hasGraded is true, years of experience is greater than or equal to five, and
          Department is same as studentDepartment)
        SET the value of gradedScore to gradedScore
    END IF
    ELSE
        PRINT a suitable message
    END ELSE

CREATE method display () with return type void
DO
    CALL method displayInfo () from parent class Teacher using super keyword
        PRINT the value of department
        PRINT the value of yearsOfExperience
IF (hasGraded is true)
    PRINT the value of gradedScore
    END IF

    ELSE
        PRINT a suitable message
    END DO
END DO
```

3.3. Tutor Class

```
CREATE class Tutor EXTENDED FROM super class Teacher
DO
```

```
    **DECLARE** instance variable salary as double using private modifier
    **DECLARE** instance variable specialization as String using private modifier
    **DECLARE** instance variable academicQualifications as String using private modifier
    **DECLARE** instance variable performanceIndex as int using private modifier
    **DECLARE** instance variable isCertified as Boolean using private modifier

CREATE the constructor Tutor Class and PASS the required parameters

CALL parameters from parent class Teacher using super keyword
CALL method setWorkingHours from super class and set the value to workingHours
INITIALIZE the value of salary to salary
INITIALIZE the value of specialization to specialization
INITIALIZE the value of academicQualifications to academicQualifications
INITIALIZE the value of performanceIndex to performanceIndex
INITIALIZE the value of isCertified to false

CREATE an accessor method getSalary () with return type double
    DO
        RETURN salary
    END DO

CREATE an accessor method getSpecialization () with return type String
    DO
        RETURN specialization
    END DO

CREATE an accessor method getAcademicQualifications () with return type String
    DO
```

```
        RETURN academicQualifications
    END DO

CREATE an accessor method getPerformanceIndex () with return type int
    DO
        RETURN performanceIndex
    END DO

CREATE an accessor method getIsCertified () with return type String
    DO
        RETURN isCertified
    END DO
CREATE method setSalary (newSalary as double, newPerformanceIndex as int) with return type void
DO
    IF (isCertified is true, newPerformanceIndex is greater than five, and
      getWorkingHours () is greater than twenty)
        SET salary to salary plus (appraisal multiplied to salary)
        SET performanceIndex to performanceIndex
        SET isCertified to true
    END IF
    ELSE
       PRINT “The salary cannot be approved. The tutor is not certified yet.”

CREATE method removeTutor ()
    IF (isCertified is true)
        SET the value of salary to zero
        SET the value of specialization to empty
        SET the value of academicQualifications to empty
        SET the value of performanceIndex to zero
        SET the value of isCertified to false
    END IF
    ELSE
        PRINT a suitable message

CREATE a method display () with return type void
DO
    IF (isCertified is false)
    CALL method displayInfo () from parent class Teacher using super keyword
    END IF
    ELSE
        CALL method displayInfo () from parent class Teacher using super
keyword
        PRINT the value of salary
        PRINT the value of specialization
        PRINT the value of academicQualifications
        PRINT the value of performanceIndex
    END DO
END DO
```

3.4. TeacherGUI Class

```
IMPORT required packages and classes
CREATE TeacherGUI class IMPLEMENTED TO ActionListener
DO
    DECLARE frame as JFrame
    DECLARE panel1, panel2 as JPanel
    DECLARE tb as JTabbedPane
    DECLARE adequate number of JLabels for Lecturer
    DECLARE adequate number of JTextField for Lecturer
    DECLARE adequate number of JButtons for Lecturer
    DECLARE adequate number of JComboBox for Lecturer
    DECLARE adequate number of JLabels for Tutor
    DECLARE adequate number of JTextField for Tutor
    DECLARE adequate number of JButtons for Tutor
    DECLARE adequate number of JComboBox for Tutor
    CREATE ArrayList ArrayList <Teacher> teacherArrayList
    CREATE method teachergui()
    DO
        INITIALIZE frame
        SET bounds to (800,620) and Visibility to true
        INITIALIZE panel1, panel2
        SET bounds to (800,620) and Layout to null
        ADD font2, font3, font4
        SET font2 to ("Franklin Gothic Book", Font.PLAIN,16)
        SET font3 to ("Franklin Gothic Book", Font.BOLD,23)
        SET font4 to "Arial", Font.BOLD,14)

        INITIALIZE panel1 to tb
        INITIALIZE panel2 to tb
        ADD tb on frame

        INITIALIZE all the components of JLabel of Lecturer
        SET bounds to all the components of JLabel of Lecturer
        ADD font to all the components of JLabel of Lecturer
```

```
        ADD all the components of JLabel of Lecturer to panel1

        INITIALIZE all the components of JTextField of Lecturer
        SET bounds to all the components of JTextField of Lecturer
        ADD all the components of JTextField of Lecturer to panel1

        INITIALIZE all the components of JButton of Lecturer
        SET bounds to all the components of JButton of Lecturer
        ADD all the components of JButton of Lecturer to panel1

        INITIALIZE all the components of JLabel of Tutor
        SET bounds to all the components of JLabel of Tutor
        ADD font to all the components of JLabel of Tutor
        ADD all the components of JLabel of Tutor to panel2

        INITIALIZE all the components of JTextField of Tutor
        SET bounds to all the components of JTextField of Tutor
        ADD all the components of JTextField of Tutor to panel2

        INITIALIZE all the components of JButton of Tutor
        SET bounds to all the components of JButton of Tutor
        ADD all the components of JButton of Tutor to panel2
        REGISTER addActionListener to all components of JButton
    ENDDO

    OVERRIDE abstract method actionPerformed (ActionEvent e) of ActionListener
interface
    DO
        IF (e.getSource()==button for clearing lecturer)
```

```
DO
    SET TEXT to "for all components of JLabel of Lecturer"
END DO
ENDIF

ELSE IF (e.getSource()==button for clearing Tutor)
DO
    SET TEXT to "for all components of JLabel of Lecturer"
END DO
END ELSEIF

ELSE IF (e.getSource() == button for adding lecturer)
DO
    TRY
    DO
        GET TEXT as teacherid, teacherName, address,
        workingType, employmentStatus, workingHours,
        department, yearsOfExperience
        INITIALIZE isAdded to false

        IF (teacherid.isEmpty() || teacherName.isEmpty() ||
        address.isEmpty() || workingHours.isEmpty() ||
        yearsOfExperience.isEmpty())
        DO
            THROW new Exception ()
            ENDDO
            ENDIF
```

```
ELSE
DO
PARSE teacheridValue to Integer and store on
teacherid
PARSE workingHoursValue to Integer and store on
workingHours
PARSE yearsOfExperienceValue to Integer and store
on yearsOfExperience
IF department.matches(“.*\\d.*”)
DO
    DISPLAY dialog box with suitable message
ENDDO
ENDIF
ELSE IF ( lecturer.getTeacherId()) ==(teacheridValue)
DO
    DISPLAY dialog box with suitable message
ENDDO
END ELSE IF
ELSE
DO
    FOR EACH LOOP in teacher and teacherList
    DOWNCAST object of Teacher as Lecturer
    type of Lecturer
    IF (lecturer.getTeacherId()) ==(teacheridValue)
    DO
        FINALIZE isAdded to true
    ENDDO
    END IF
```

```
                END FOR
                IF (isAdded == true)
                DO
                    DISPLAY error dialog box with suitable
                    message
                ENDDO
                END ELSE IF
            END DO
            END ELSE
        END DO
    END DO

    CATCH (NumberFormatException exc)
    DO
        DISPLAY dialog box with suitable message
    ENDDO
    CATCH (Exception ex)
    DO
        DISPLAY dialog box with suitable message
    ENDDO
    END ELSE IF

    ELSE IF (e.getSource() == button for adding Tutor)
    DO
        TRY
        DO
        GET TEXT as teacherid, teacherName, address,
        workingType, employmentStatus, workingHours,
        salary, specialization, academicQualification,
        performanceIndex
        INITIALIZE isAdded to false
        IF (teacherid.isEmpty() || teacherName.isEmpty() ||
        address.isEmpty() || workingHours.isEmpty() ||
        salary.isEmpty()  || specialization.isEmpty() ||
        academicQualification.isEmpty() ||
        performanceIndex.isEmpty())
        DO
            DISPLAY suitable error message
        ENDDO
        ENDIF

        ELSE
        DO
            PARSE teacheridValue to Integer and store on
            teacherid
            PARSE workingHoursValue to Integer and store on
            workingHours
            PARSE salaryValue to Double and store
            on salary
            PARSE performanceIndexValue to Integer and store
            on performanceIndexValue
            IF (tutor.getTeacherId()) ==(teacheridValue)
            DO
                DISPLAY dialog box with suitable message
            ENDDO
            END IF
```

```
DO
    FOR EACH LOOP in teacher and teacherList
    DOWNCAST object of Teacher as Tutor
    type of Tutor
    IF (tutor.getTeacherId()) ==(teacheridValue)
    DO
        FINALIZE isAdded to true
    ENDDO
    END IF
    END FOR
    IF (isAdded == true)
    DO
        DISPLAY error dialog box with suitable
        message
    ENDDO
END IF
ELSE IF (isAdded == false)
DO
    CREATE (teacheridValue, teacherName,
    address, workingType, employmentStatus,
    workingHoursValue, salaryValue,
    academicQualification, specialization,
    performanceIndexValue)
    ADD tutor to teacherArrayList
    DISPLAY successfully added dialog box
    With suitable message
    END DO
    END ELSE IF
```

```
                END DO
                END ELSE
            ENDDO
        ENDDO
        CATCH (NumberFormatException exc)
        DO
            DISPLAY dialog box with suitable message
        ENDDO
    ENDDO
    END ELSE IF

    ELSE IF (e.getSource() == button to grade assignment)
    DO
        TRY
        DO
            GET TEXT as teacherid, gradedScore, department,
            yearsOfExperience
            INITIALIZE isAdded = false
            IF (teacherid.isEmpty() || gradedScore.isEmpty() ||
            department.isEmpty() || yearsOfExperience.isEmpty())
            DO
                DISPLAY suitable error message
            ENDDO
            ENDIF

            ELSE
            DO
                FOR LOOP (int i = 0; i<teacherArrayList.size(); i++)
```

```
DO
    IF ((teacherArrayList.get(i).getTeacherId()) ==
    teacheridValue)
    END IF
ENDDO
END FOR
IF (isAdded == true)
DO
    FOR EACH LOOP in Lecturer and Teacher
    ArrayList
    DO
    IF (teacherList instanceof Lecturer)
    DO
    DOWNCAST object of Lecturer as lecturer type of
    teacherList
    IF (lecturer.getTeacherId()) ==(teacheridValue)
    DO
        IF (lecturer.getHasGraded() == true)
        DO
        DISPLAY dialog box with suitable Message
        ENDDO
        END IF
        ELSE IF (lecturer.getHasGraded()==false)
        DISPLAY success dialog box with suitable
        Message
        ENDDO
        END ELSE IF
    ENDDO
```

```
                    END IF
                    ENDDO
                    END IF
                    END FOR
                ENDDO
                END IF

                ELSE IF (isAdded == false)
                DO
                    DISPLAY error box
                ENDDO
                END ELSE IF
            ENDDO
            END ELSE
        END ELSE

    ENDDO
    CATCH (Exception exc)
    DO
        DISPLAY dialog box with suitable message
    ENDDO
    END ELSE IF

    ELSE IF (e.getSource() == setSalaryButton)
    DO
        TRY
        DO
            GET TEXT as teacherid, salary, performanceIndex
```

```
INITIALIZE isAdded = false
IF (teacherid.isEmpty() || salary.isEmpty() ||
performanceIndex.isEmpty())
DO
    DISPLAY an error message
ENDDO
ENDIF

DO
    PARSE teacheridValue to Integer and store on
    teacherid
    PARSE salaryValue to Double and store
    on salary
    PARSE performanceIndexValue to Integer and store
    on performanceIndexValue

ELSE
DO
FOR LOOP (int i = 0; i < teacherArrayList.size(); i++)
DO
    IF (teacherArrayList.get(i).getTeacherId() == teacheridValue)
    FINALIZE isAdded = true
    END IF
ENDDO
END FOR
IF (isAdded == true)
DO
    FOR EACH LOOP (Teacher teacherList : teacherArrayList)
```

```
        DO
        IF (teacherList instanceof Tutor)
        DO
        DOWNCAST object of an Tutor type of tutor of
        teacheridValue
        DO
            IF (tutor.getIsCertified()==true)
            DO
            DISPLAY dialog box with suitable message
            ENDDO
            END IF
            ELSE IF (tutor.getIsCertified() == false)
            DO
            DISPLAY successfully updated dialog box with
            Suitable message
            ENDDO
            END ELSE IF
        ENDDO
        END IF
        ENDDO
        END IF
        ENDDO
        END FOR
    ENDDO
    END IF
    ELSE IF (isAdded == false)
    DO
        DISPLAY dialog box
```

```
                ENDDO
                END ELSE IF
            ENDDO
            END ELSE
        END ELSE

ENDDO
CATCH (NumberFormatException exc)
DO
        DISPLAY dialog box with suitable message
ENDDO
END ELSE IF

ELSE IF (e.getSource() == removeTutorButton)
        DO
            GET TEXT as teacherid
            TRY
            DO
                IF (teacherid.isEmpty())
                DO
                    DISPLAY dialog box with suitable message
                ENDDO
                ELSE
                DO
                    PARSE teacheridValue to Integer and store on teacherid
                    INITIALIZE isAdded = false

                    FOR LOOP (int i = 0; i < teacherArrayList.size(); i++)
        DO
            IF (teacherArrayList.get(i).getTeacherId() ==
            teacheridValue)

                IF (teacherArrayList.get(i) is instance of Tutor)
                DO
                    DOWNCAST object of Tutor type of
                    tutor of teacherArrayList.get(i)
                    IF (tutor.getIsCertified() == true)
                    DO
                        DISPLAY dialog box with suitable
                        message
                    END IF
                    ELSE IF (tutor.getIsCertified() == false)
                    DO
                        REMOVE tutor from
                        teacherArrayList
                        DISPLAY success dialog box
                        with suitable message
                    END ELSE IF
                END DO
            END IF
        END FOR

        IF (isAdded == false)
        DO
            DISPLAY dialog box with suitable message
        END IF
    END ELSE
```

```
        ENDDO
        CATCH (NumberFormatException exc)
        DO
            DISPLAY dialog box with suitable message
        ENDDO
    END ELSE IF

    CREATE main method (String [] args)
        DO
            CALL teacherGUI()
        ENDDO
    ENDDO
```

4. Method Description

In Java, a method is a collection of lines of code intended to carry out certain tasks. Java developers can save time by using code that can be repeated and modified in other programs because of their reusable and extensible design. Java method declarations describe the properties of the method, including its name, arguments, return type, and visibility. Also, it has six internal components that together make up the "Method header (Roberts, 2023)."

4.1. Teacher Class

Methods	Description
getTeacherId()	The accessor method returns the value of teacher ID with int data type
getTeacherName()	The accessor method returns the value of teacher Name with String data type
getAddress()	The accessor method returns the value of address with String data type
getWorkingType()	The accessor method returns the value of working Type with String data type
getEmploymentStatus()	The accessor method returns the value of employment Status with String data type
getWorkingHours()	The accessor method returns the value of working Hours with int data type
setWorkingHours(int workingHours)	The mutator method sets the value for working hours by accepting variables of int data type as working Hours
displayInfo()	The method displays all the attributes

Table 1: Methos description of Teacher Class

4.2. Lecturer Class

Methods	Description
getDepartment()	The accessor method returns the value of department with String data type
getYearsOfExperience()	The accessor method returns the value of years Of Experience with int data type
getGradedScore()	The accessor method returns the value of graded Score with int data type
getHasGraded()	The accessor method returns the value of has Graded with Boolean data type
setGradedScore(int gradedScore)	The mutator method sets the value for Graded Score by accepting variables of int data type as gradedScore
gradeAssignment(int gradedScore, String studentDepartment, int yearsOfExperience)	The method which accepts graded Score as int, student Department as String, years Of experience as int to grade assignment by examining has Graded status
display()	The method displays all the suitable output for its details

Table 2: Methods description of Lecturer Class

4.3. Tutor Class

Methods	Description
getSalary()	The accessor method returns the value of salary with double data type
getSpecialization()	The accessor method returns the value of specialization with String data type
getAcademicQualifications()	The accessor method returns the value of academic Qualifications with String data type
getPerformanceIndex()	The accessor method returns the value of performance Index with int data type
getIsCertified()	The accessor method returns the value of is Certified with boolean data type
setSalary(double newSalary, int newPerformance)	The mutator method sets the value for salary by accepting variables of double data type as new salary and int data type as new Performance
removeTutor()	The method which to remove tutor by examining is Certified status
display()	The method displays all the suitable output for its details

Table 3: Methods description of Tutor Class

4.4. TeacherGUI Class

Methods	Description
TeacherGUI	A public method with no return type, this is the method that arranges and adds all of the GUI's components. Two JPanels were added to the single frame's JTabbedPane, along with additional swing components such as JButton, JTextField, and JLabel with their borders specified properly.
main (String [] args)	This is the main method of the class which calls the static methods where we have created the graphics user interface, without creating an object of the class.
actionPerformed (ActionEvent e)	The term "Action Performed" refers to all of our GUI's event and exception handling scenarios. The class is initially implemented using the ActionListener interface, followed by the essential components for which event handling is required. The ActionListener interface is then registered using the keyword "this", and finally the actionPerformed function is overrided. It adds courses to the array list, registers courses found through the array list, and removes courses that meet its criteria. Similarly, we may utilize the display and clear buttons to display and clear the relevant data and text fields, respectively.
getSource()	This method is defined in the EventObject class (java.awt). We get a pointer to the source object from AWTEvent. The function returns the object for either of the events upon pressing the specified button in the coding area. It is a component of the actionPerformed method, which determines the action taken and the button that was clicked.

getText ()	This method is used to get a particular text from the message catalogue—or, in the case of a graphical user interface, the text field. This method retrieves the text from its field; here, a text is identifiable by its JTextField variable name. Using this technique, you can compare text that is retrieved from a text field with text or items that are stored in the array list.
setText ()	This method is used to modify the display text in the text field. The text of this StringCharacterIterator is set to the desired text when this method is called, and the given screen element's parameter is set to that text as well. By using this method, all text fields are cleared and set to an empty string when the clear button is pressed.
isEmpty ()	This method tests if the input string is empty or not, returning true if the string contains zero characters, else false. To catch an exception, isEmpty was used to check if any text fields were left empty.
Matches ()	This method is used to determine whether a string satisfies a given expression or not, which means that throughout the coding session, it was used to ensure that only string values were used in text fields such as IDs and names.
parseInt ()	This method returns a new integer value that has been initialized to the value specified in the string attribute definition. After parsing a single string parameter, it returns an integer value that matches the string argument. First, local variables were retrieved and values in strings were taken from the text field.

Size ()	This method is part of the List interface and is used to determine the number of elements in a list container. It accepts no parameters and returns the total number of elements contained in the Array List, which was used to iterate over the while array list to check for added elements.
add ()	This method is used to insert a specific element into a set collection of the Array List, so that when an object of the Lecturer and Tutor is formed, it must be put in the array list that accepts the object as an argument.
getHasGraded ()	The accessor method was called from Lecturer Class to determine whether the assignment was graded or not. The instance was first first checked for a valid teacher list via downcasting, and if the status hasGraded was true, grading assignment was not possible; however, if the status of hasGraded was false, the grading method from the so-called parent class was called and the parameter was passed.
getIsCertified ()	The accessor method was called from the Tutor class to determine whether the teacher is certified or not so that we can do the salary appraisal and remove the tutor accordingly. The instance was first checked for a valid teacher list via downcasting, and if the status of isCertified was true, salary of the teacher was appraised but removing the tutor wasn’t possible; however, if the status of isCertified was false, the setting salary method from the so-called parent class was called and the parameter was passed, and removing the tutor was possible in this condition.

display ()	Once invoked, the display method returns a list of all the details associated with the TeacherGUI teacher’s list. The Lecturer and Tutor objects were constructed and thus used to invoke the display method via down casting the parent class as a type of the sub class using the for each loop iterating over each element in that class.
clear ()	After we insert all the values at last these clear methods is used to clear all the text so that we can insert text again.

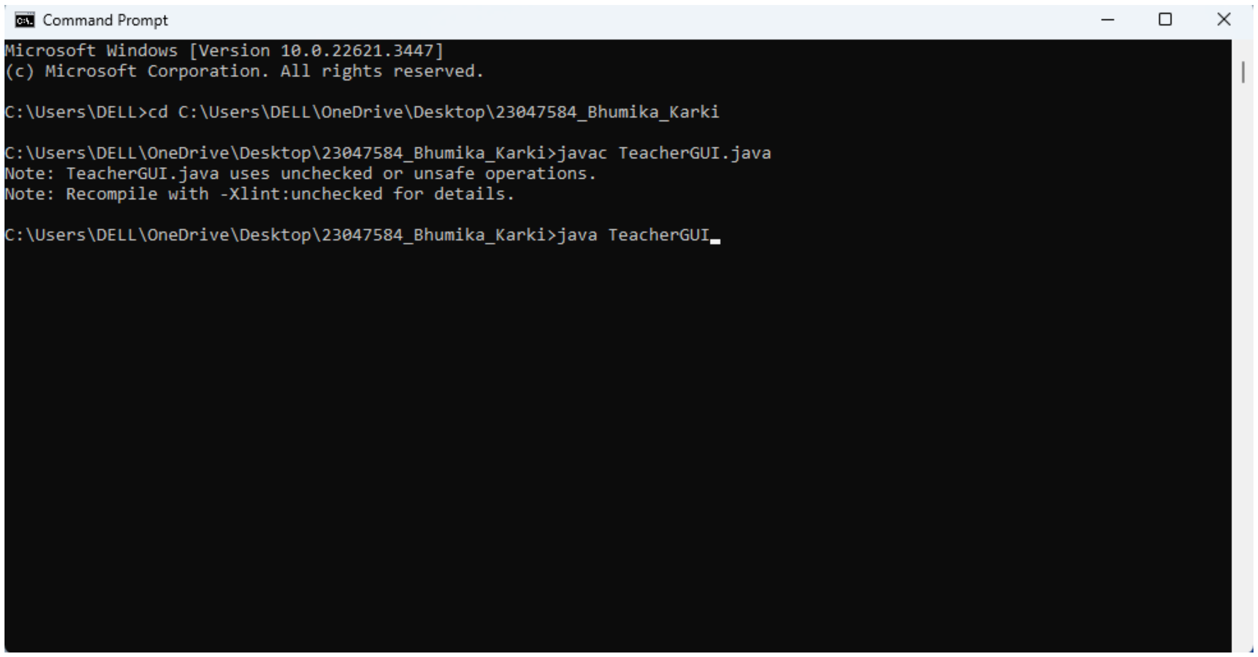
Table 4: Methods description of TeacherGUI Class

5. Inspection

5.1. Inspection Test: 1

Test Number	**1**
Objective	To compile and run the program using command prompt.
Action	• The command prompt was opened and the directory for the specified java file was allocated. • The java file was first compiled and then executed.
Expected Result	The .java program files would compile, and the GUI would be displayed.
Actual Result	The .java program files was compile, and the GUI was be displayed.
Conclusion	The test was successful.

Table 5: Inspection for compiling and running the program through cmd



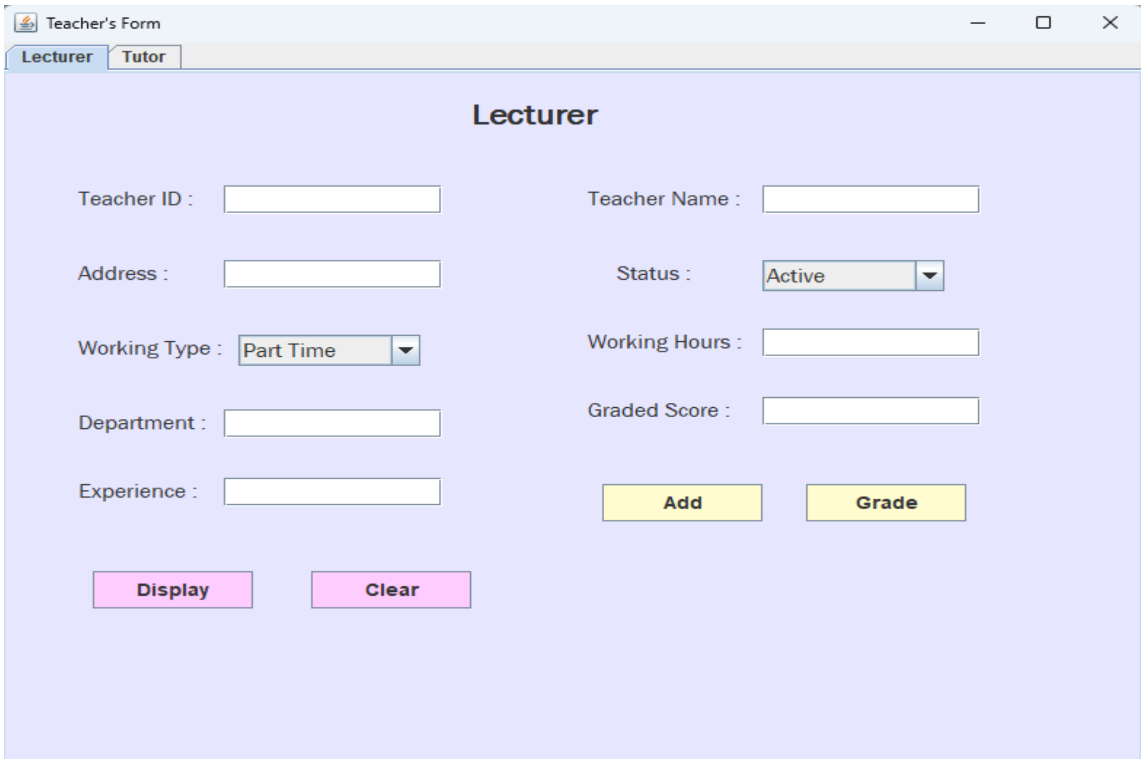
```
Microsoft Windows [Version 10.0.22621.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>cd C:\Users\DELL\OneDrive\Desktop\23047584_Bhumika_Karki

C:\Users\DELL\OneDrive\Desktop\23047584_Bhumika_Karki>javac TeacherGUI.java
Note: TeacherGUI.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\DELL\OneDrive\Desktop\23047584_Bhumika_Karki>java TeacherGUI
```

Figure 7: Compiling and running the program through cmd



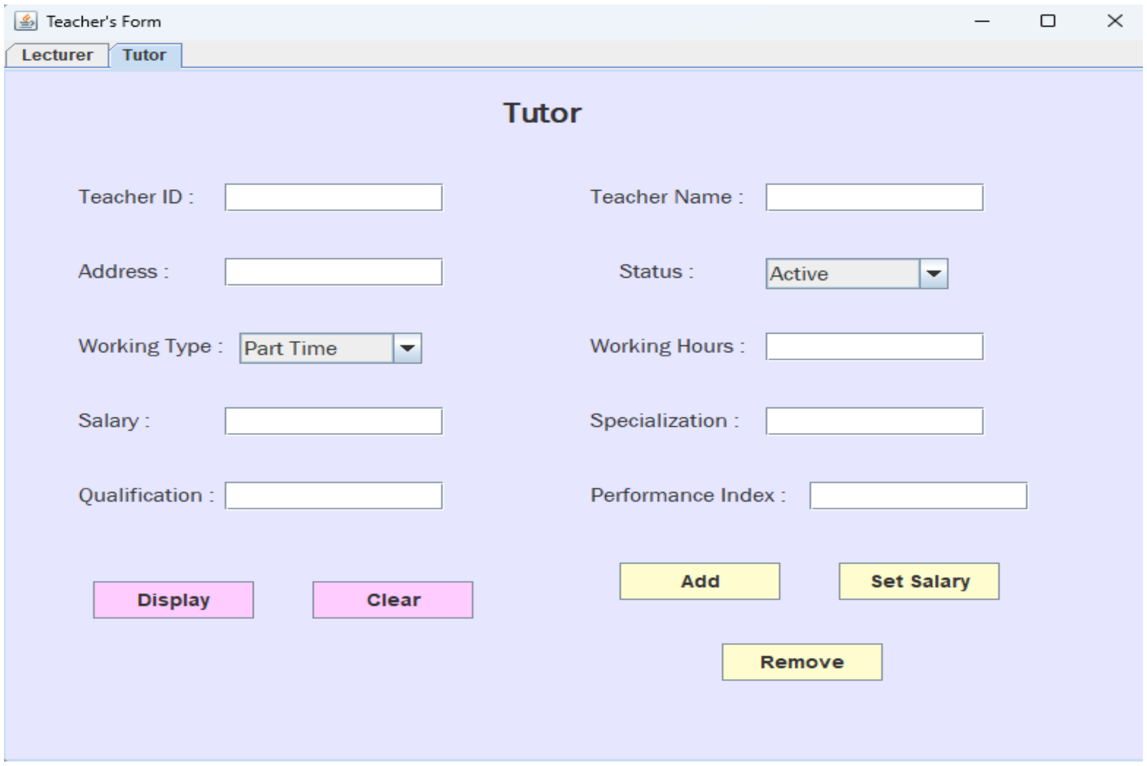
Teacher's Form

Lecturer | Tutor

Lecturer

Teacher ID :		Teacher Name :	
Address :		Status :	Active
Working Type :	Part Time	Working Hours :	
Department :		Graded Score :	
Experience :			

Figure 9: Program after running (Lecturer)



Teacher's Form

Lecturer | Tutor

Tutor

Teacher ID :		Teacher Name :	
Address :		Status :	Active
Working Type :	Part Time	Working Hours :	
Salary :		Specialization :	
Qualification :		Performance Index :	

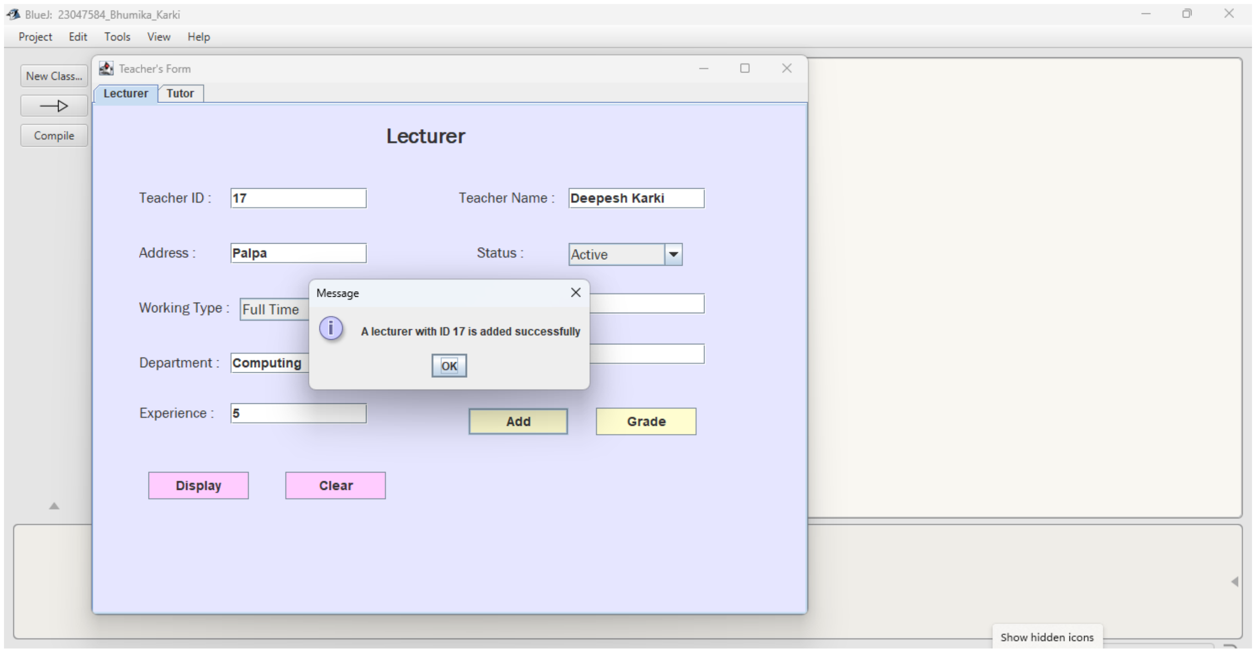
Figure 8: Program after running (Tutor)

5.2. Inspection Test: 2

i. Addition of the Lecturer

Test Number	**2**
Objective	To add Lecturer
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Deepesh Karki Address : Palpa Status : Active Working Hours : 8 Working Type : Full Time Department : Computing Experience : 5 • The “Add” button was clicked to add a Lecturer. • The “Display” button was clicked to inspect addition of Lecturer to the array list of the Teacher Class.
Expected Result	A JOptionPane or a dialog box stating that the Lecturer was successfully added would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the Lecturer was successfully added was displayed.
Conclusion	The test was successful.

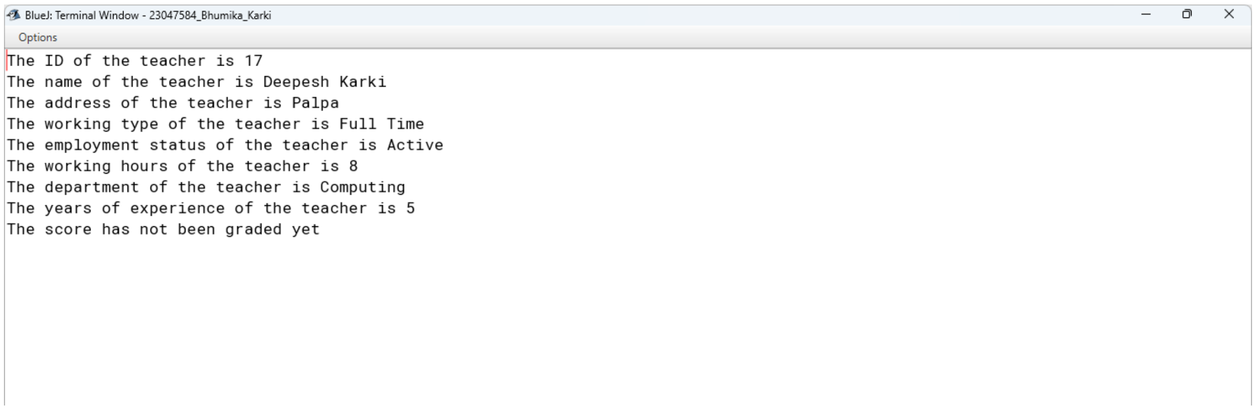
Table 6: Inspection for the Addition of the Lecturer



Teacher ID : 17
Teacher Name : Deepesh Karki
Address : Palpa
Status : Active
Working Type : Full Time
Department : Computing
Experience : 5

A lecturer with ID 17 is added successfully

Figure 10: Addition of Lecturer



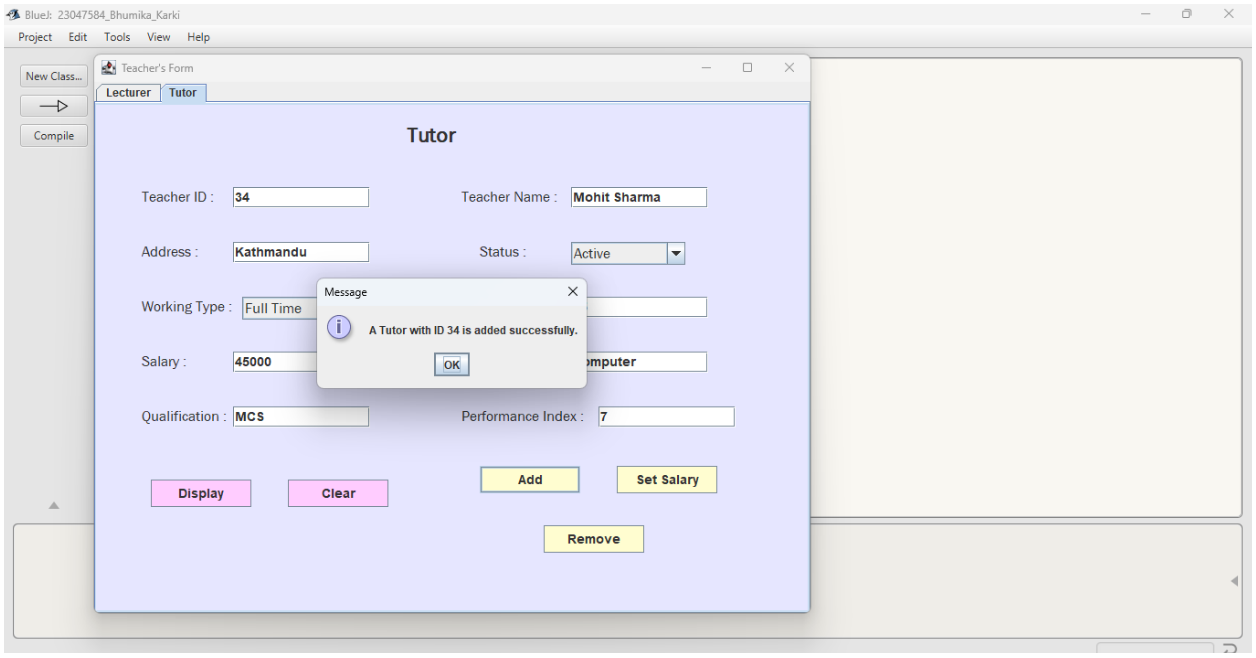
```
The ID of the teacher is 17
The name of the teacher is Deepesh Karki
The address of the teacher is Palpa
The working type of the teacher is Full Time
The employment status of the teacher is Active
The working hours of the teacher is 8
The department of the teacher is Computing
The years of experience of the teacher is 5
The score has not been graded yet
```

Figure 11: Display for Addition of Lecturer

ii. Addition of the Tutor

Test Number	**3**
Objective	To add Tutor
Action	• The parameters were filled as follows: • Teacher ID : 34 Teacher Name: Mohit Sharma Address : Kathmandu Status : Active Working Hours : 25 Working Type : Full Time Salary : 45000 Specialization : Computer Qualification : MCS Performance Index : 7 • The “Add” button was clicked to add a Tutor. • The “Display” button was clicked to inspect addition of Tutor to the array list of the Teacher Class.
Expected Result	A JOptionPane or a dialog box stating that the Tutor was successfully added would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the Tutor was successfully added was displayed.
Conclusion	The test was successful.

Table 7: Inspection for Addition of the Tutor



Tutor

Teacher ID : 34

Teacher Name : Mohit Sharma

Address : Kathmandu

Status : Active

Working Type : Full Time

Salary : 45000

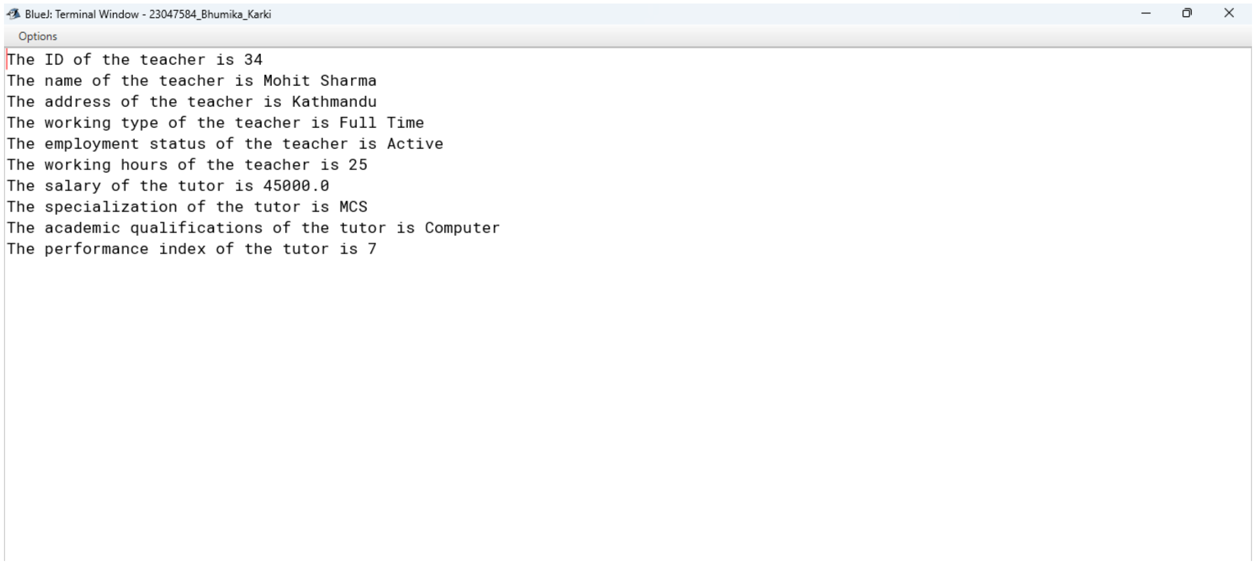
mputer

Qualification : MCS

Performance Index : 7

A Tutor with ID 34 is added successfully.

Figure 12: Addition of Tutor



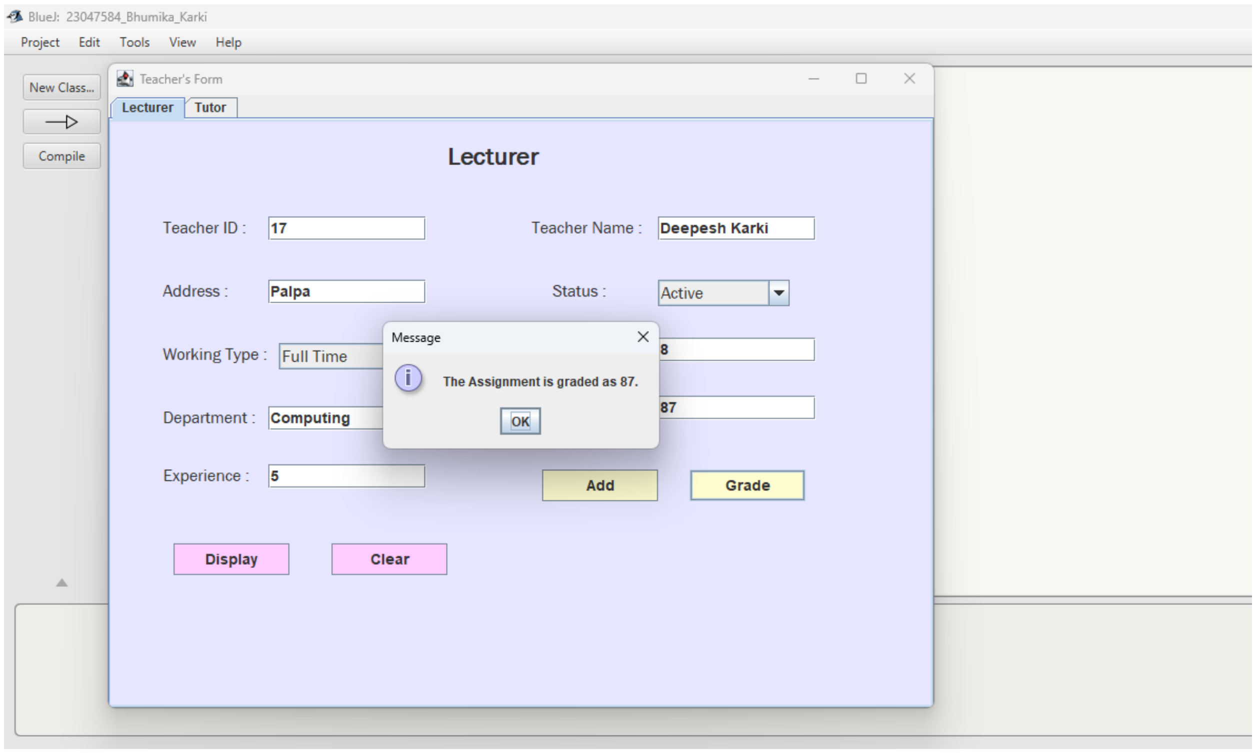
```
The ID of the teacher is 34
The name of the teacher is Mohit Sharma
The address of the teacher is Kathmandu
The working type of the teacher is Full Time
The employment status of the teacher is Active
The working hours of the teacher is 25
The salary of the tutor is 45000.0
The specialization of the tutor is MCS
The academic qualifications of the tutor is Computer
The performance index of the tutor is 7
```

Figure 13: Display for Addition of Tutor

iii. To Grade Assignments from Lecturer

Test Number	**4**
Objective	To grade assignment from Lecturer.
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Deepesh Karki Address : Palpa Status : Active Working Hours : 8 Working Type : Full Time Department : Computing Experience : 5 Graded Score : 87 • The "Add" button was clicked to add a Lecturer. • The "Grade" button was clicked to grade assignments. • The "Display" button was clicked to inspect addition of Lecturer to the array list of the Teacher Class.
Expected Result	A JOptionPane or a dialog box stating that the assignment was graded successfully would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the assignment was graded successfully was displayed.
Conclusion	The test was successful.

Table 8: Inspection to Grade Assignment



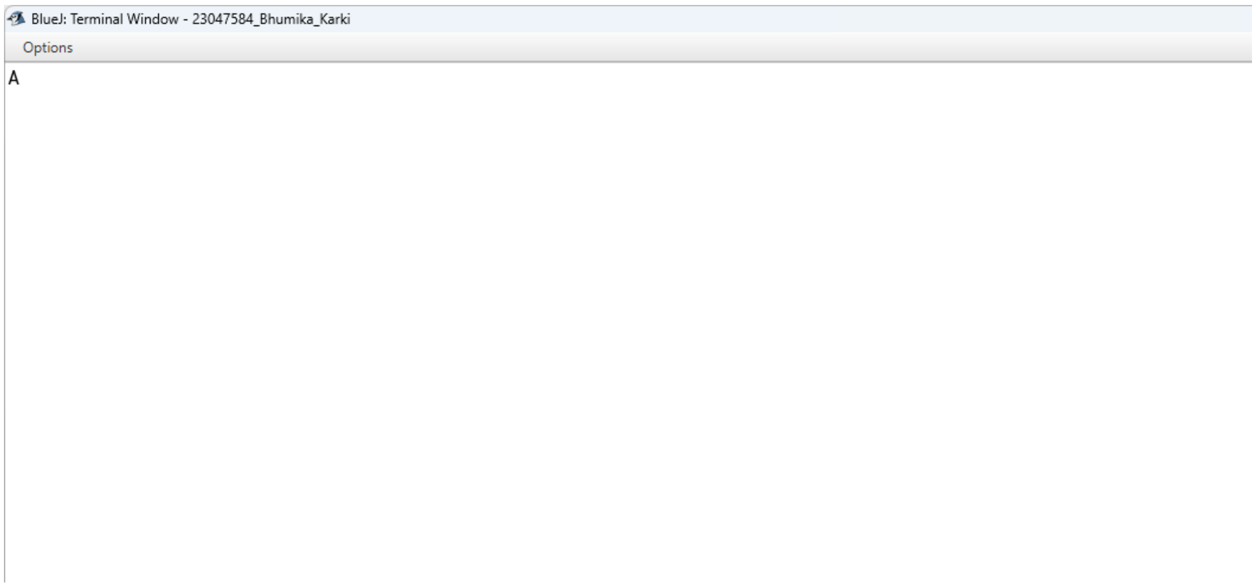
Teacher ID : 17
Teacher Name : Deepesh Karki
Address : Palpa
Status : Active
Working Type : Full Time
Department : Computing
Experience : 5

8

87

The Assignment is graded as 87.

Figure 14: Grading Assignment



```
A
```

Figure 15: Display for Grading Assignment

iv. To set the salary of the Tutor

Test Number	**5**
Objective	To add Tutor
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Mohit Sharma Address : Kathmandu Status : Active Working Hours : 25 Working Type : Full Time Salary : 45000 Specialization : Computer Qualification : Msc Computer Science Performance Index : 7 • The “Add” button was clicked to add a Tutor. • The “Set Salary” button was clicked to set new salary of the tutor. • The “Display” button was clicked to inspect addition of Tutor to the array list of the Teacher Class.
Expected Result	A JOptionPane or a dialog box stating that the salary was updated successfully would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the salary was updated successfully was displayed.
Conclusion	The test was successful.

Table 9: Inspection to set new salary

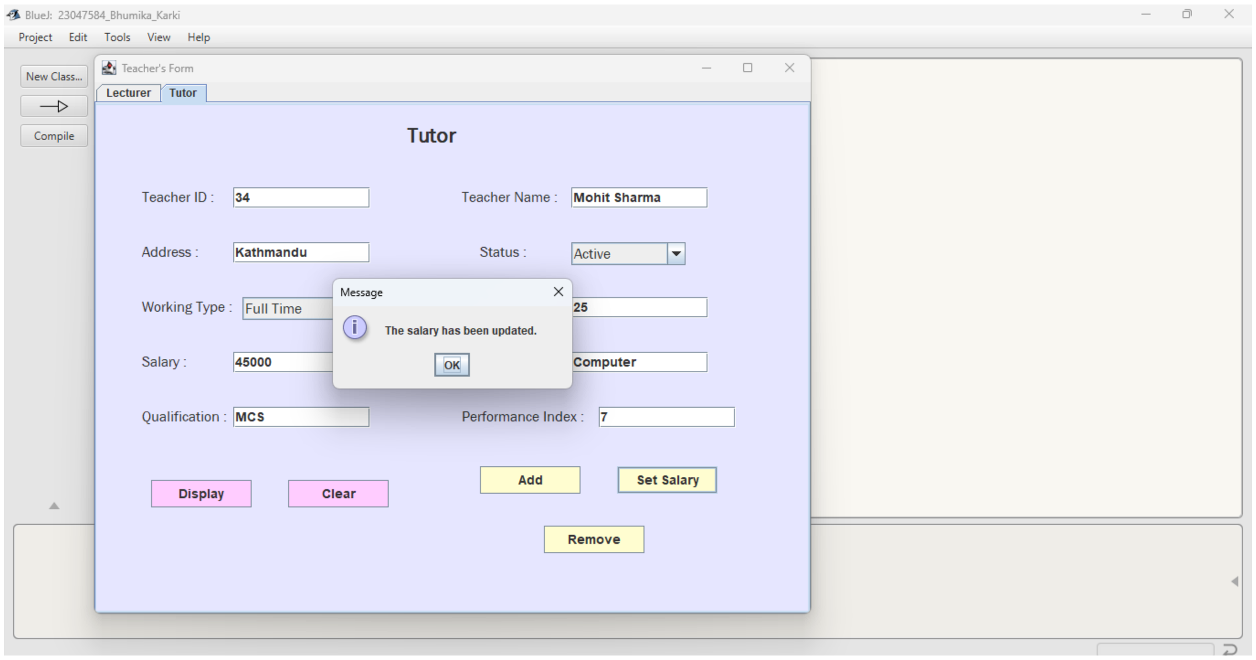
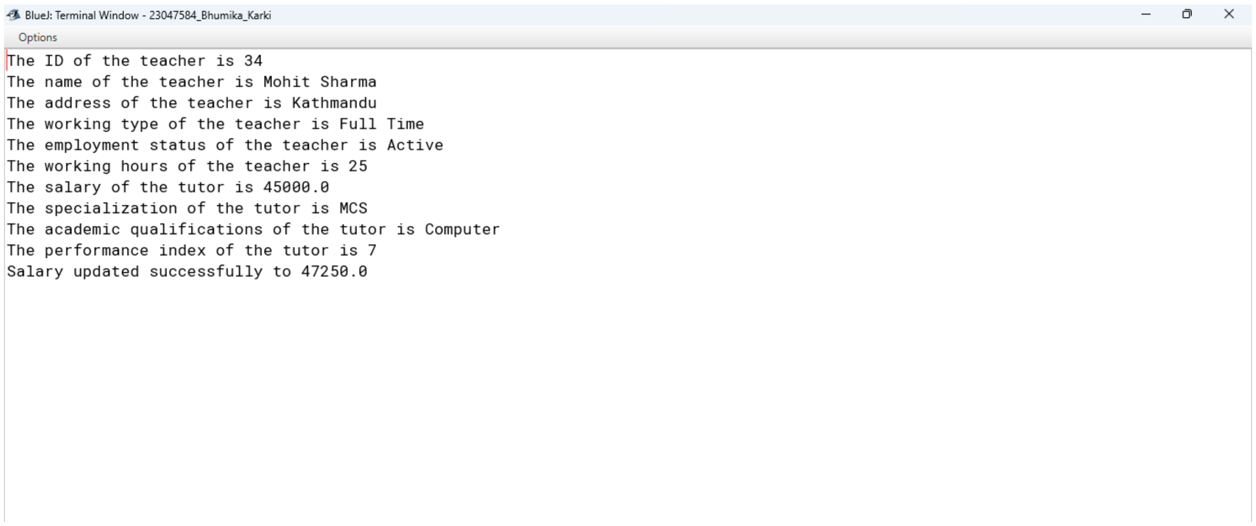


Figure 16: Setting new salary of Tutor



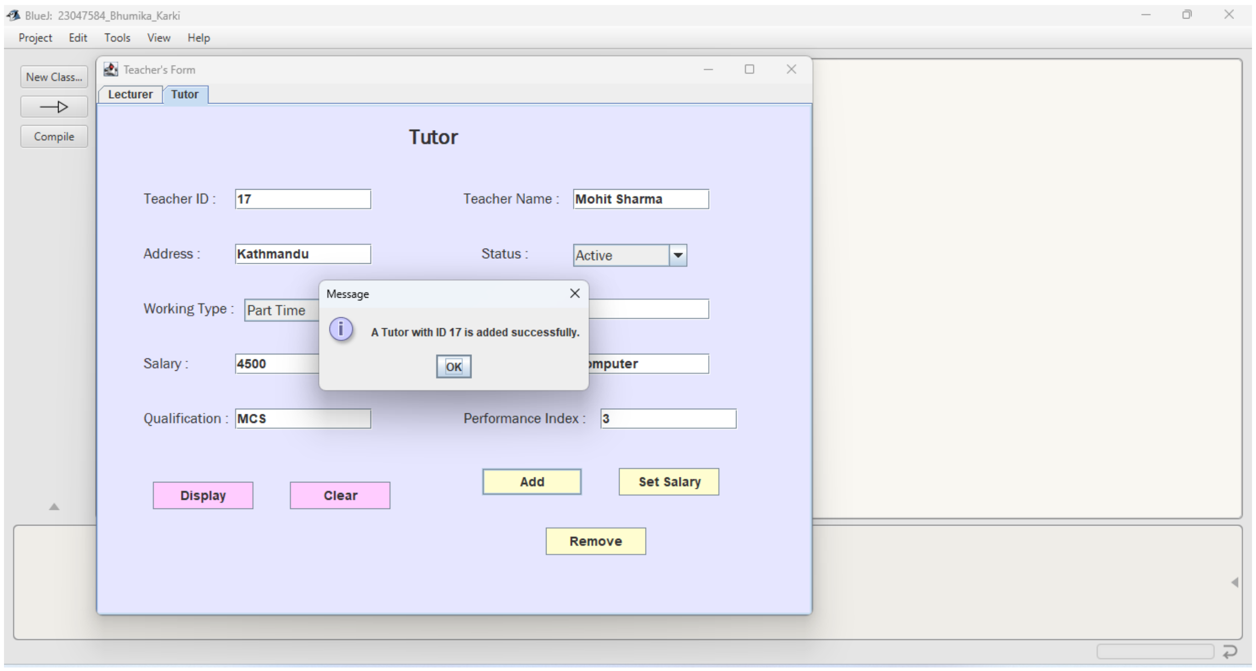
```
The ID of the teacher is 34
The name of the teacher is Mohit Sharma
The address of the teacher is Kathmandu
The working type of the teacher is Full Time
The employment status of the teacher is Active
The working hours of the teacher is 25
The salary of the tutor is 45000.0
The specialization of the tutor is MCS
The academic qualifications of the tutor is Computer
The performance index of the tutor is 7
Salary updated successfully to 47250.0
```

Figure 17: Display for Setting new salary of Tutor

v. To remove the Tutor

Test Number	**6**
Objective	To remove the Tutor.
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Mohit Sharma Address : Kathmandu Status : Active Working Hours : 8 Working Type : Part Time Salary : 4500 Specialization : Computer Qualification : MCS Performance Index : 3 • The “Add” button was clicked to add a Tutor. • The “Remove” button was clicked to remove the tutor. • The “Display” button was clicked to see if the tutor still exist in the Array List.
Expected Result	A JOptionPane or a dialog box stating that the tutor is removed successfully would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the tutor is removed successfully was displayed.
Conclusion	The test was successful.

Table 10: Inspection to remove tutor



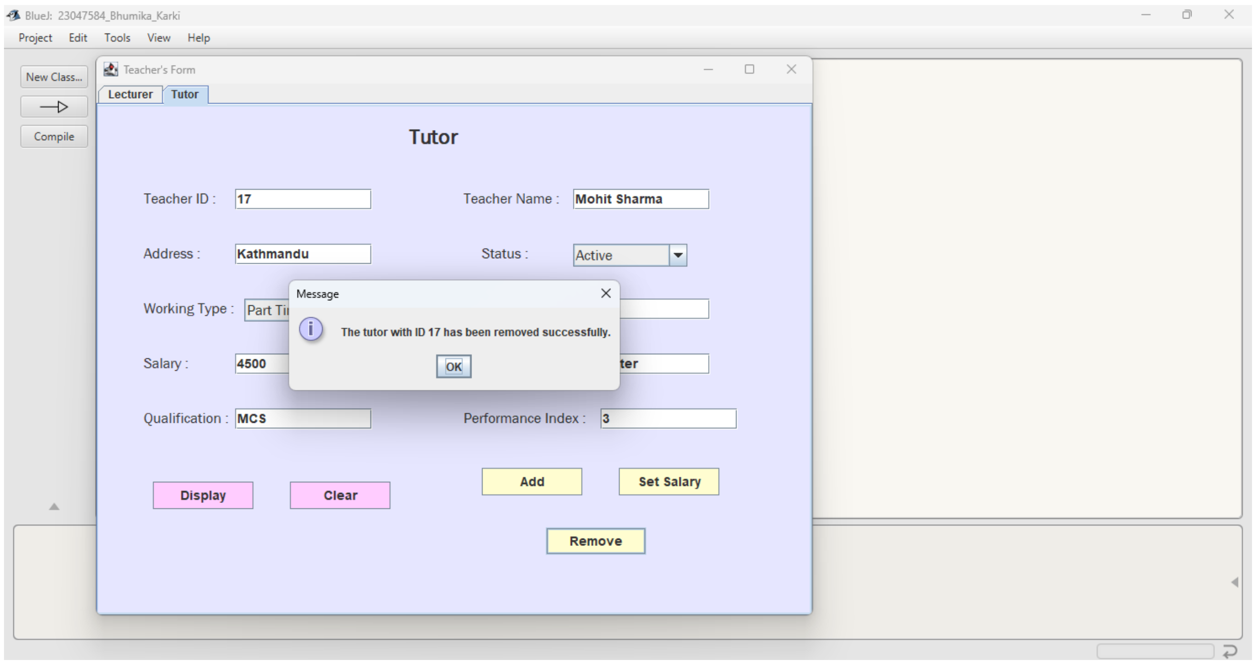
Tutor

Teacher ID : 17
Teacher Name : Mohit Sharma
Address : Kathmandu
Status : Active
Working Type : Part Time
Salary : 4500
Qualification : MCS
Performance Index : 3

Message

A Tutor with ID 17 is added successfully.

Figure 18: Adding the Tutor



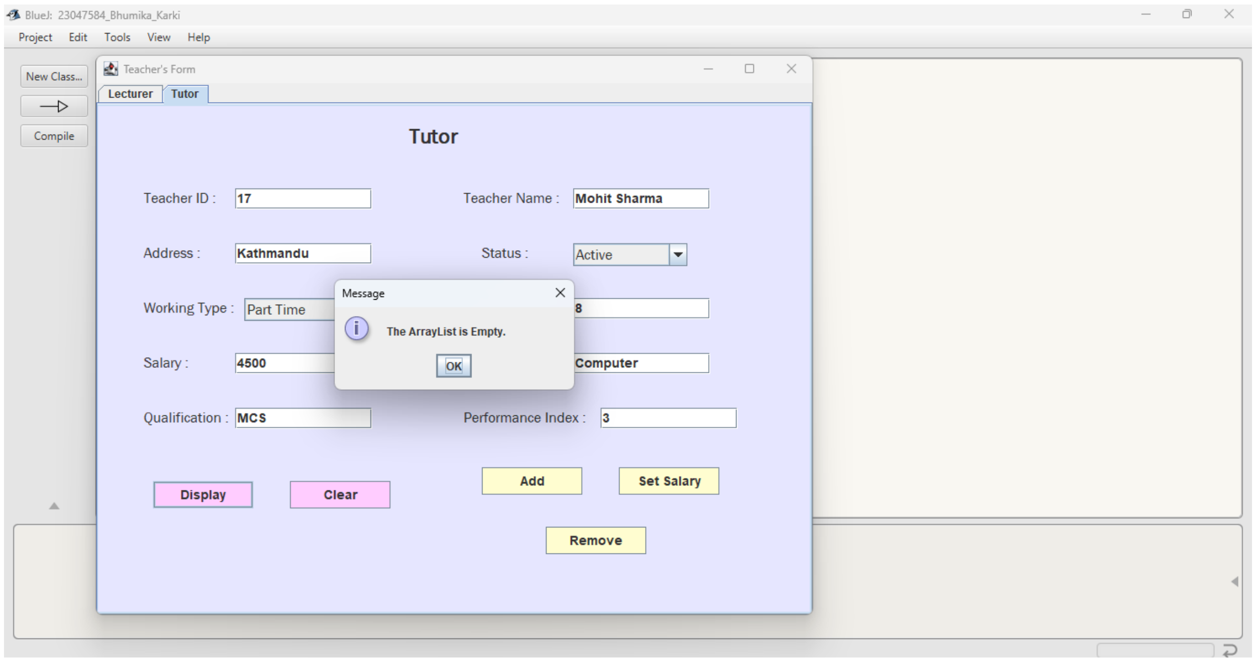
Tutor

Teacher ID : 17
Teacher Name : Mohit Sharma
Address : Kathmandu
Status : Active
Salary : 4500
Qualification : MCS
Performance Index : 3

Message

The tutor with ID 17 has been removed successfully.

Figure 19: Removing the Tutor



Tutor

Teacher ID : 17
Teacher Name : Mohit Sharma
Address : Kathmandu
Status : Active
Working Type : Part Time
8
Salary : 4500
Computer
Qualification : MCS
Performance Index : 3

Message

The ArrayList is Empty.

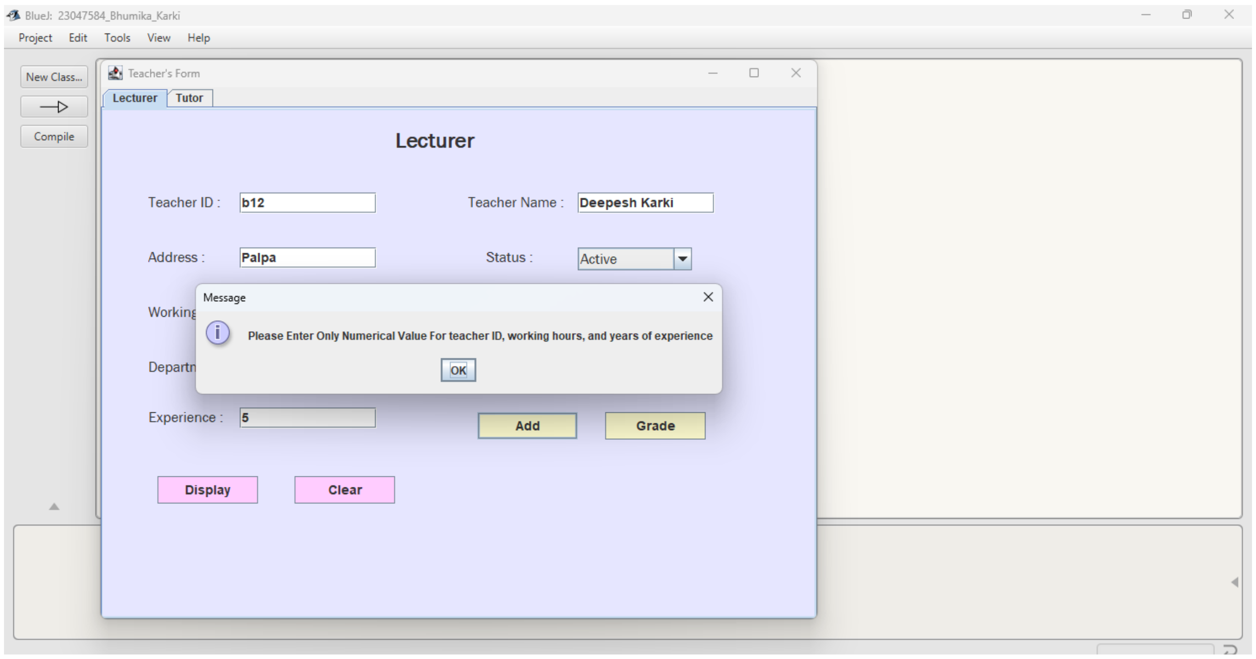
Figure 20: Checking the Array List

5.3. Inspection Test: 3

i. Addition of unsuitable values for Teacher ID

Test Number	**7**
Objective	To add unsuitable value for Teacher ID
Action	• The parameters were filled as follows: • Teacher ID : b12 Teacher Name: Deepesh Karki Address : Palpa Status : Active Working Hours : 8 Working Type : Full Time Department : Computing Experience : 5 • The “Add” button was clicked to add a Lecturer.
Expected Result	A JOptionPane or a dialog box stating to add only numerical value for teacherID would be displayed.
Actual Result	A JOptionPane or a dialog box stating to add only numerical value for teacherID was displayed.
Conclusion	The test was successful.

Table 11: Inspection for addition of unsuitable values for teacherID



Lecturer

Teacher ID : b12

Teacher Name : Deepesh Karki

Address : Palpa

Status : Active

Experience : 5

Message

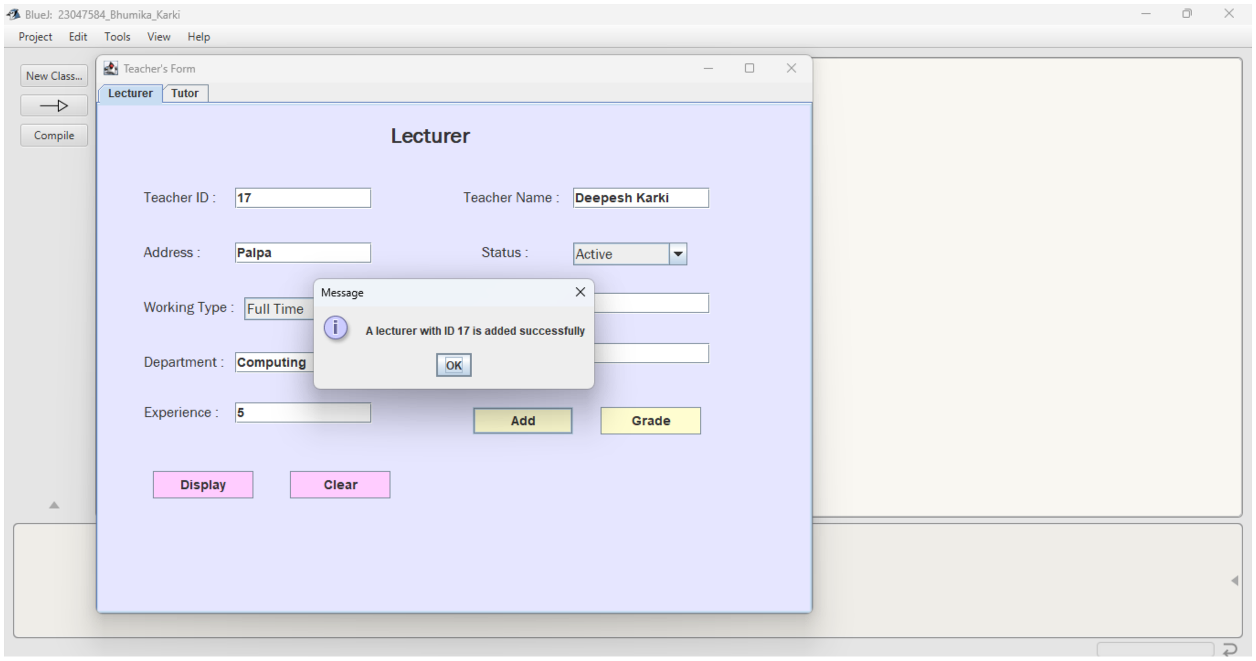
Please Enter Only Numerical Value For teacher ID, working hours, and years of experience

Figure 21: Addition of unsuitable value for Teacher ID

ii. To add an already added Lecturer

Test Number	**8**
Objective	To add a duplicate Teacher ID
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Deepesh Karki Address : Palpa Status : Active Working Hours : 8 Working Type : Full Time Department : Computing Experience : 5 • The "Add" button was clicked to add a Lecturer. • The "Add" button was re-clicked to add a duplicate teacher ID with the same as before.
Expected Result	A JOptionPane or a dialog box stating that the Lecturer was already added would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the Lecturer was already added was displayed.
Conclusion	The test was successful.

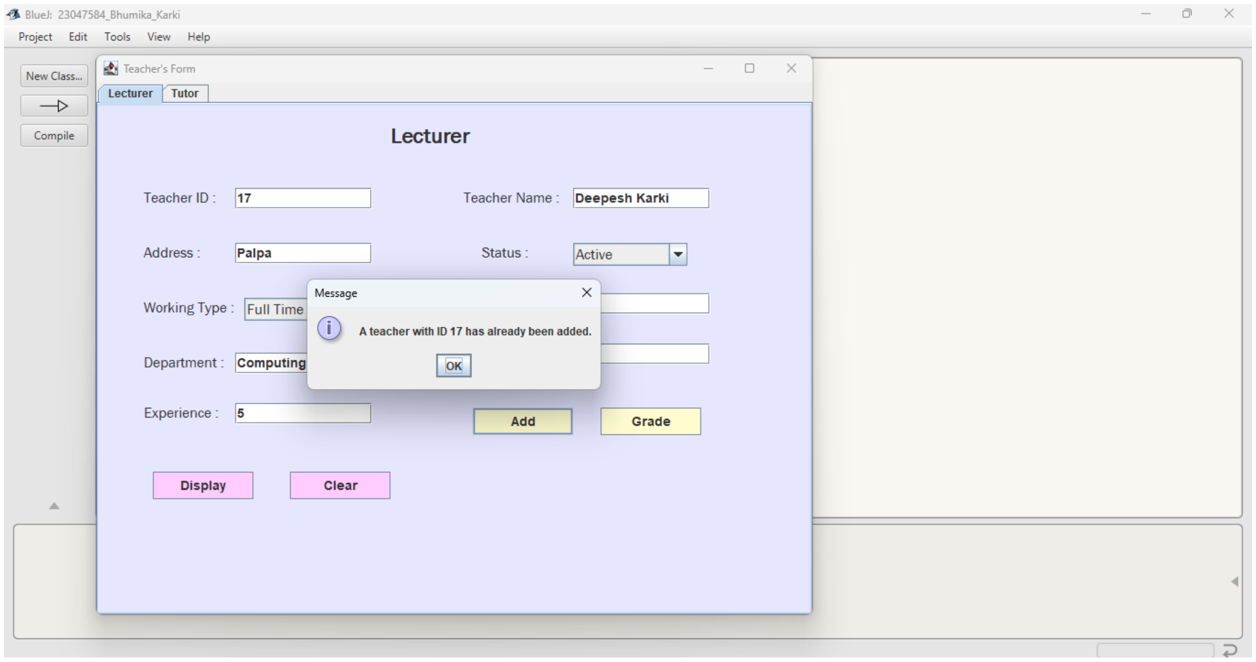
Table 12: Inspection for Addition of duplicate Lecturer



Teacher ID : 17
Teacher Name : Deepesh Karki
Address : Palpa
Status : Active
Working Type : Full Time
Department : Computing
Experience : 5

A lecturer with ID 17 is added successfully

Figure 22: Addition of Lecturer



Teacher ID : 17
Teacher Name : Deepesh Karki
Address : Palpa
Status : Active
Working Type : Full Time
Department : Computing
Experience : 5

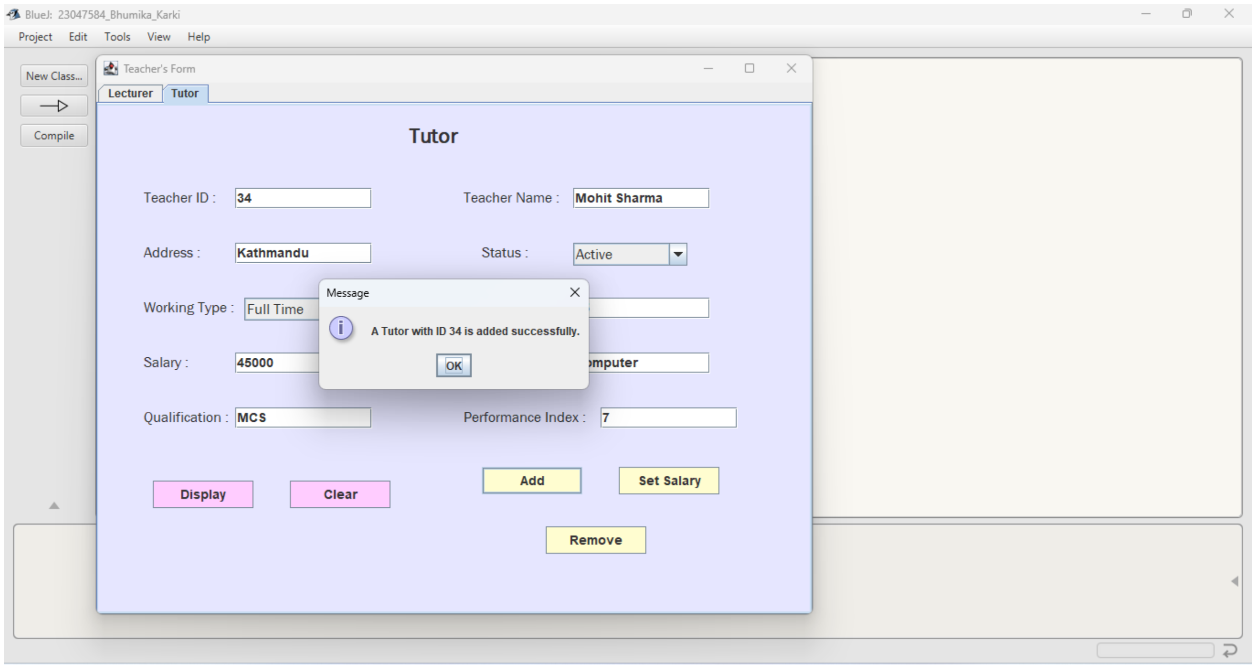
A teacher with ID 17 has already been added.

Figure 23: Addition of duplicate Lecturer

iii. To add an already added Tutor

Test Number	**9**
Objective	To add a duplicate Teacher ID
Action	• The parameters were filled as follows: • Teacher ID : 34 Teacher Name: Mohit Sharma Address : Kathmandu Status : Active Working Hours : 25 Working Type : Full Time Salary : 45000 Specialization : Computer Qualification : MCS Performance Index : 7 • The “Add” button was clicked to add a Tutor. • The “Add” button was re-clicked to add a duplicate teacher ID with the same as before.
Expected Result	A JOptionPane or a dialog box stating that the Tutor was already added would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the Tutor was added added was displayed.
Conclusion	The test was successful.

Table 13: Inspection for Addition of duplicate Tutor


BlueJ: 23047584_Bhumika_Karki

Teacher's Form

Tutor

Teacher ID : 34

Teacher Name : Mohit Sharma

Address : Kathmandu

Status : Active

Working Type : Full Time

Salary : 45000

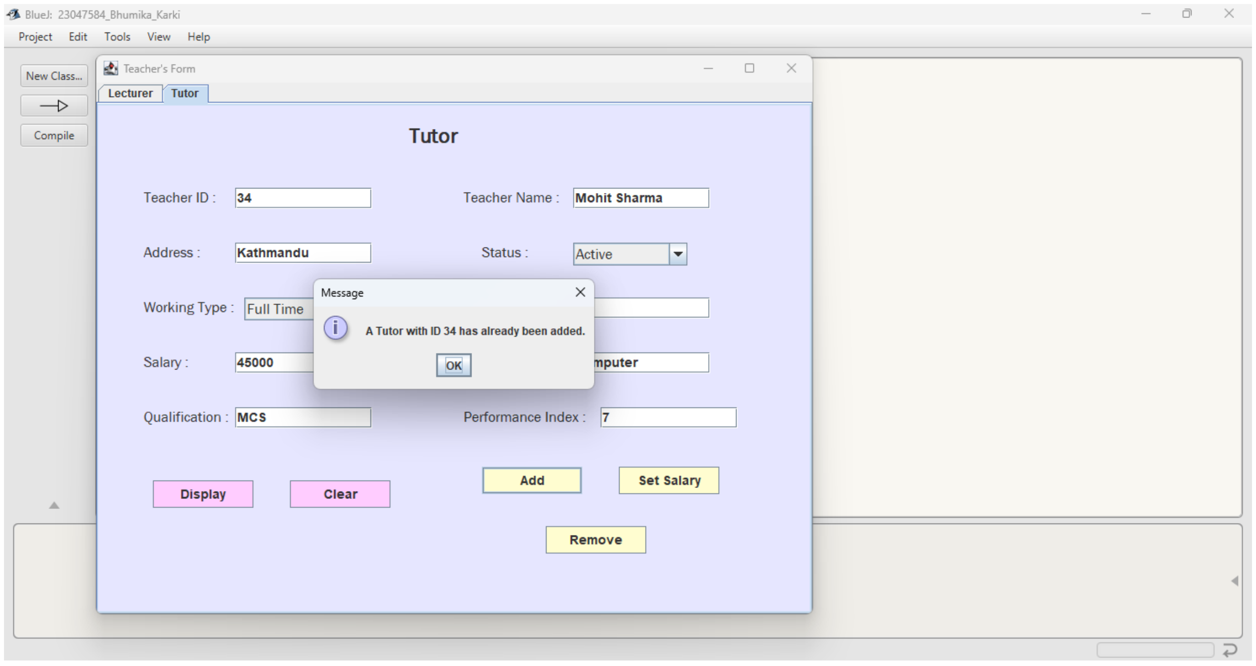
Qualification : MCS

Performance Index : 7

Message

A Tutor with ID 34 is added successfully.

Figure 24: Addition of Tutor


BlueJ: 23047584_Bhumika_Karki

Teacher's Form

Tutor

Teacher ID : 34

Teacher Name : Mohit Sharma

Address : Kathmandu

Status : Active

Working Type : Full Time

Salary : 45000

Qualification : MCS

Performance Index : 7

Message

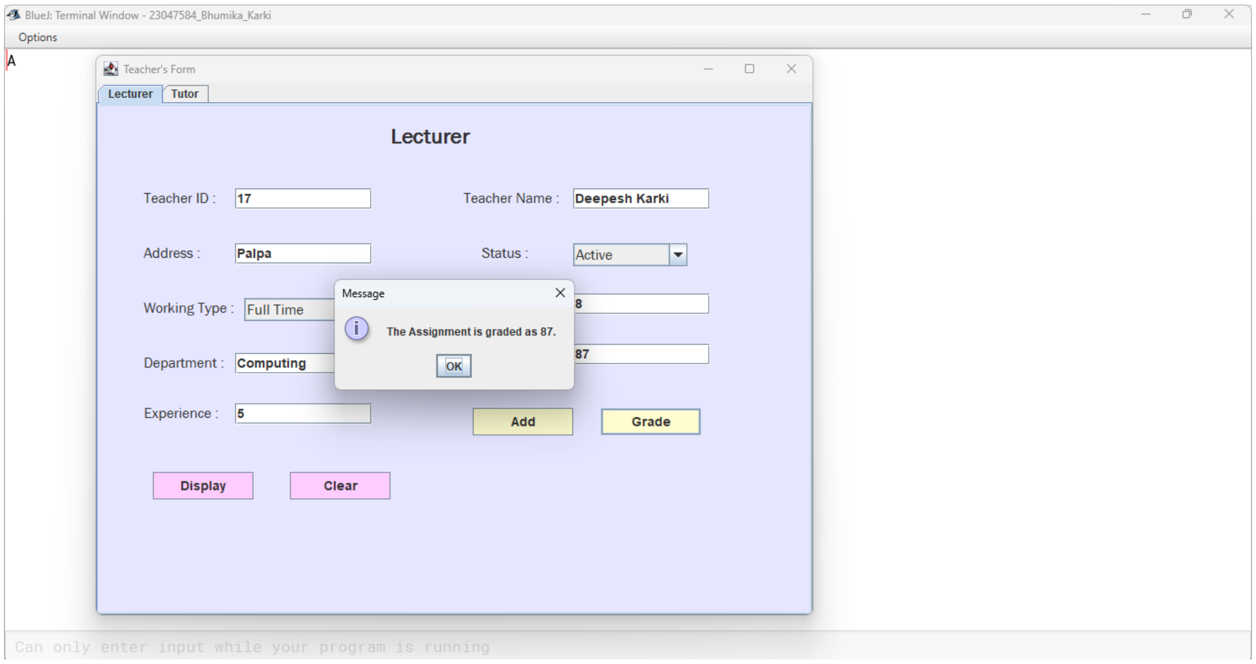
A Tutor with ID 34 has already been added.

Figure 25: Addition of duplicate Tutor

iv. To grade an already graded assignment

Test Number	**10**
Objective	To grade assignment from Lecturer.
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Deepesh Karki Address : Palpa Status : Active Working Hours : 8 Working Type : Full Time Department : Computing Experience : 5 Graded Score : 87 • The "Add" button was clicked to add a Lecturer. • The "Grade" button was clicked to grade assignments. • The "Grade" button was re-clicked to grade the assignment again.
Expected Result	A JOptionPane or a dialog box stating that the assignment was already graded would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the assignment was already graded was displayed.
Conclusion	The test was successful.

Table 14: Inspection for grading an already graded assignment


BlueJ: Terminal Window - 23047584_Bhumika_Karki
Options
Teacher's Form
Lecturer
Tutor
Display
Clear
Add
Grade
OK
Can only enter input while your program is running

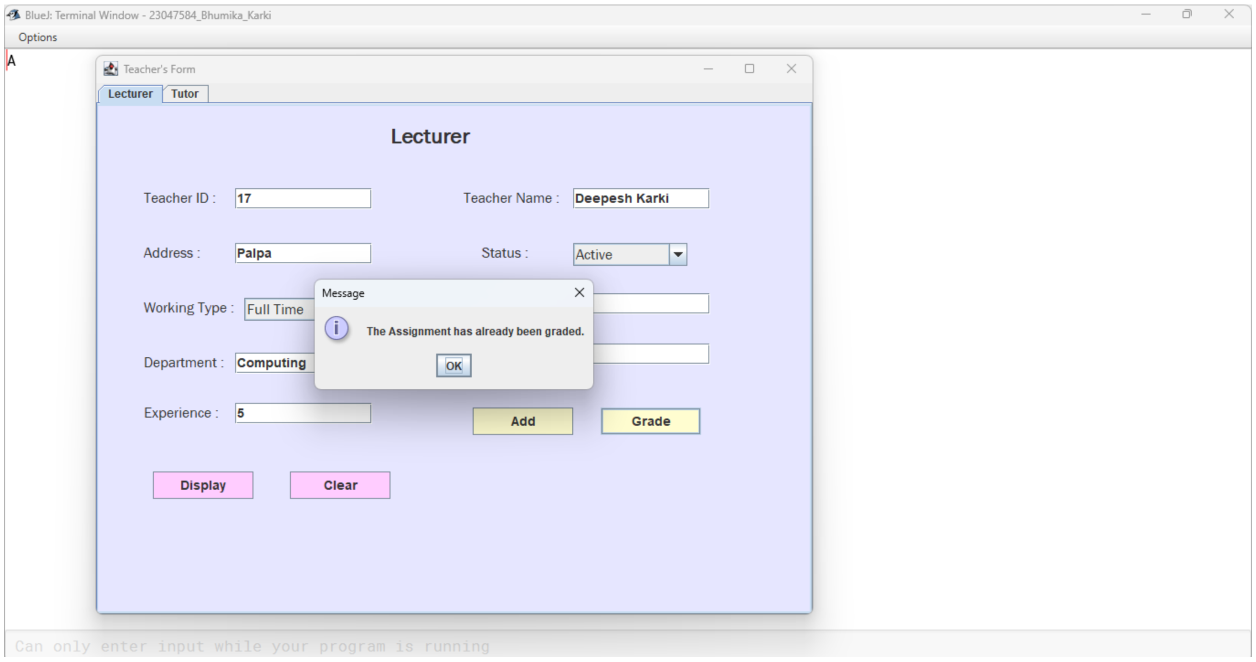
Lecturer

Teacher ID : 17
Teacher Name : Deepesh Karki
Address : Palpa
Status : Active
Working Type : Full Time
Department : Computing
Experience : 5

8
87

Message
The Assignment is graded as 87.

Figure 26: Grading an Assignment


BlueJ: Terminal Window - 23047584_Bhumika_Karki
Options
Teacher's Form
Lecturer
Tutor
Display
Clear
Add
Grade
OK
Can only enter input while your program is running

Lecturer

Teacher ID : 17
Teacher Name : Deepesh Karki
Address : Palpa
Status : Active
Working Type : Full Time
Department : Computing
Experience : 5

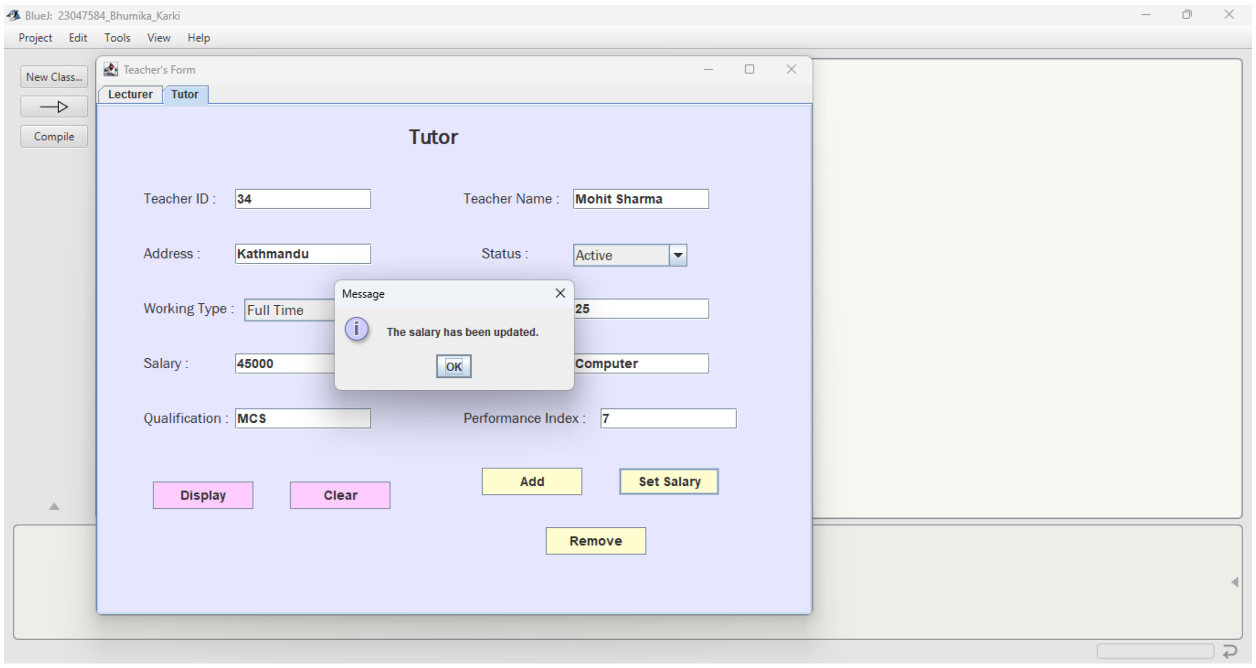
Message
The Assignment has already been graded.

Figure 27: Grading an already graded Assignment

v. To set the already appraised salary of the Tutor

Test Number	**11**
Objective	To add Tutor
Action	• The parameters were filled as follows: • Teacher ID : 17 Teacher Name: Mohit Sharma Address : Kathmandu Status : Active Working Hours : 25 Working Type : Full Time Salary : 45000 Specialization : Computer Qualification : Msc Computer Science Performance Index : 7 • The “Add” button was clicked to add a Tutor. • The “Set Salary” button was clicked to set new salary of the tutor. • The “Set Salary” button was re-clicked to set new salary of the tutor again.
Expected Result	A JOptionPane or a dialog box stating that the salary was already updated would be displayed.
Actual Result	A JOptionPane or a dialog box stating that the salary was already updated was displayed.
Conclusion	The test was successful.

Table 15: Inspection for setting an already appraised salary



BlueJ: 23047584_Bhumika_Karki

Teacher's Form

Tutor

Teacher ID : 34

Teacher Name : Mohit Sharma

Address : Kathmandu

Status : Active

Working Type : Full Time

25

Salary : 45000

Computer

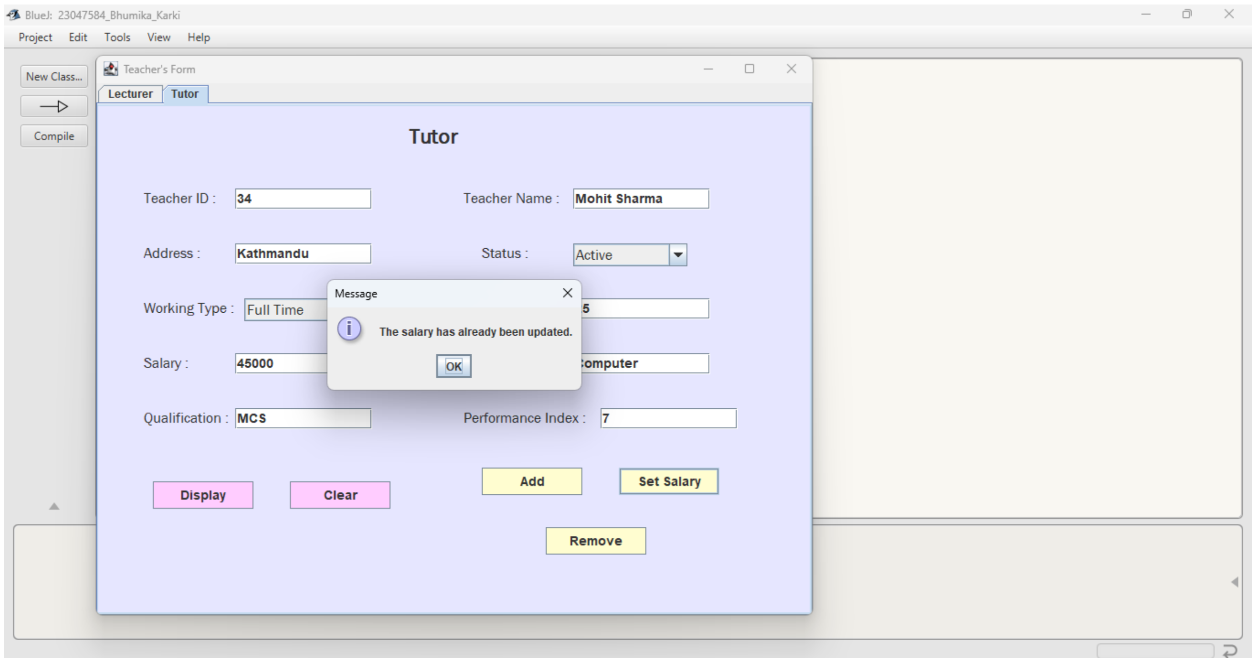
Qualification : MCS

Performance Index : 7

Message

The salary has been updated.

Figure 28: Setting new salary for the Tutor



BlueJ: 23047584_Bhumika_Karki

Teacher's Form

Tutor

Teacher ID : 34

Teacher Name : Mohit Sharma

Address : Kathmandu

Status : Active

Working Type : Full Time

5

Salary : 45000

omputer

Qualification : MCS

Performance Index : 7

Message

The salary has already been updated.

Figure 29: Setting an already appraised salary for the Tutor

6. Error Detection

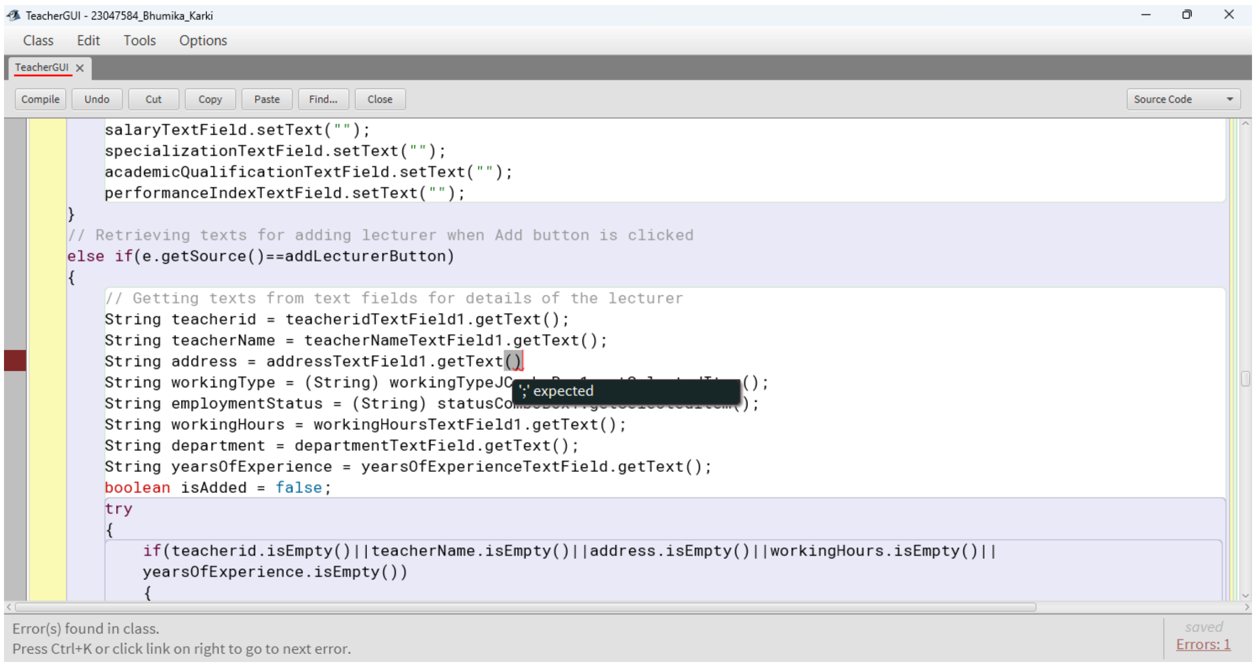
An error is an illegal statement or action made by the programmer that results in abnormal behavior from the application or program and prevents it from carrying out its tasks effectively. Until the program is constructed or run, errors typically go unnoticed. Some may prevent the program from compiling or from being executed, or they may cause the program to terminate.
To write programs, the Java programming language has a set of rules. If the programmer disregards these guidelines, an error may arise. Errors can also arise when a programmer executes an unintentional operation due to a wrong idea or concept (Mishra, 2022).

6.1. Syntax Error

Syntax errors, also known as compile errors, are errors that the compiler detects. Mistakes in code composition, such spelling a term incorrectly, leaving out important punctuation, or using an opening brace without a matching closing brace, can lead to syntax mistakes. Because the compiler indicates where these problems are and what caused them, they are typically simple to find (javaguides, n.d.).

6.1.1. Error

A syntactical error was detected in the code block displayed below. This is a result of the programmer using incorrect codes or misinterpreting the language when coding. Four lines of code in the programme remained unfinished without the semicolon ";," and the compiler prohibited the programme from executing until it was corrected.


TeacherGUI - 23047584_Bhumika_Karki

```
        salaryTextField.setText("");
        specializationTextField.setText("");
        academicQualificationTextField.setText("");
        performanceIndexTextField.setText("");
    }
    // Retrieving texts for adding lecturer when Add button is clicked
    else if(e.getSource()==addLecturerButton)
    {
        // Getting texts from text fields for details of the lecturer
        String teacherid = teacheridTextField1.getText();
        String teacherName = teacherNameTextField1.getText();
        String address = addressTextField1.getText()
        String workingType = (String) workingTypeJC
        String employmentStatus = (String) statusCo
        String workingHours = workingHoursTextField1.getText();
        String department = departmentTextField.getText();
        String yearsOfExperience = yearsOfExperienceTextField.getText();
        boolean isAdded = false;
        try
        {
            if(teacherid.isEmpty()||teacherName.isEmpty()||address.isEmpty()||workingHours.isEmpty()||
            yearsOfExperience.isEmpty())
            {
```

';' expected

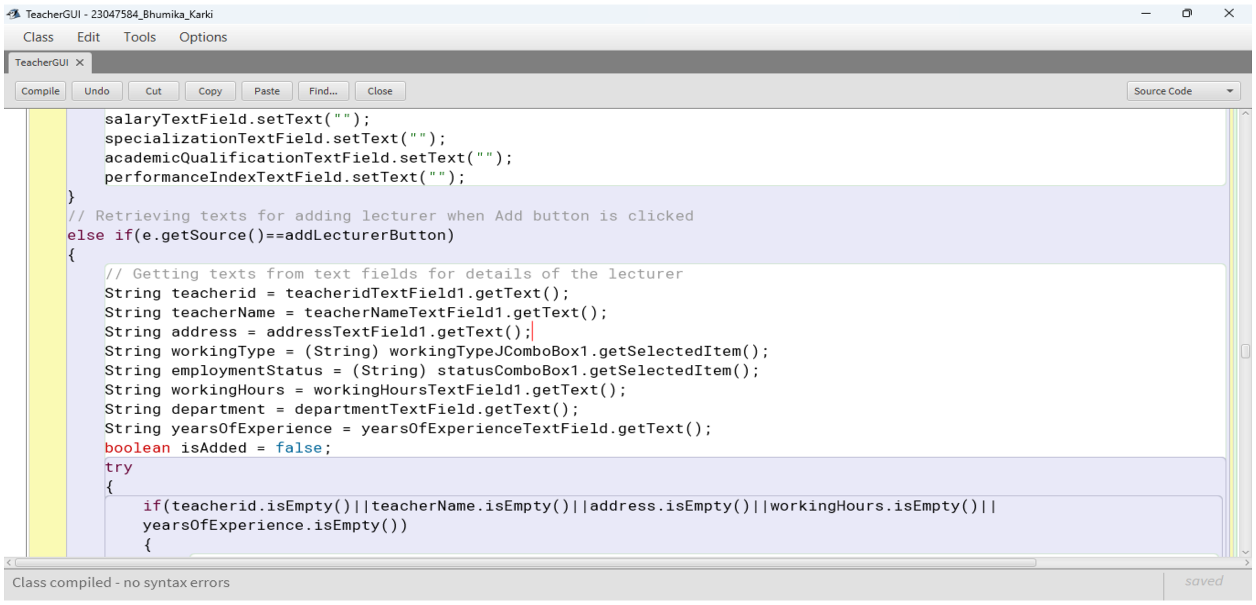
Error(s) found in class.
Press Ctrl+K or click link on right to go to next error.

saved
Errors: 1

Figure 30: Syntax Error

6.1.2. Correction

The program’s solution was very simple; by including the correct semicolon at the end of each block of text, the class was compiled successfully.


TeacherGUI - 23047584_Bhumika_Karki

```
        salaryTextField.setText("");
        specializationTextField.setText("");
        academicQualificationTextField.setText("");
        performanceIndexTextField.setText("");
    }
    // Retrieving texts for adding lecturer when Add button is clicked
    else if(e.getSource()==addLecturerButton)
    {
        // Getting texts from text fields for details of the lecturer
        String teacherid = teacheridTextField1.getText();
        String teacherName = teacherNameTextField1.getText();
        String address = addressTextField1.getText();
        String workingType = (String) workingTypeJComboBox1.getSelectedItem();
        String employmentStatus = (String) statusComboBox1.getSelectedItem();
        String workingHours = workingHoursTextField1.getText();
        String department = departmentTextField.getText();
        String yearsOfExperience = yearsOfExperienceTextField.getText();
        boolean isAdded = false;
        try
        {
            if(teacherid.isEmpty()||teacherName.isEmpty()||address.isEmpty()||workingHours.isEmpty()||
            yearsOfExperience.isEmpty())
            {
```

Class compiled - no syntax errors

saved

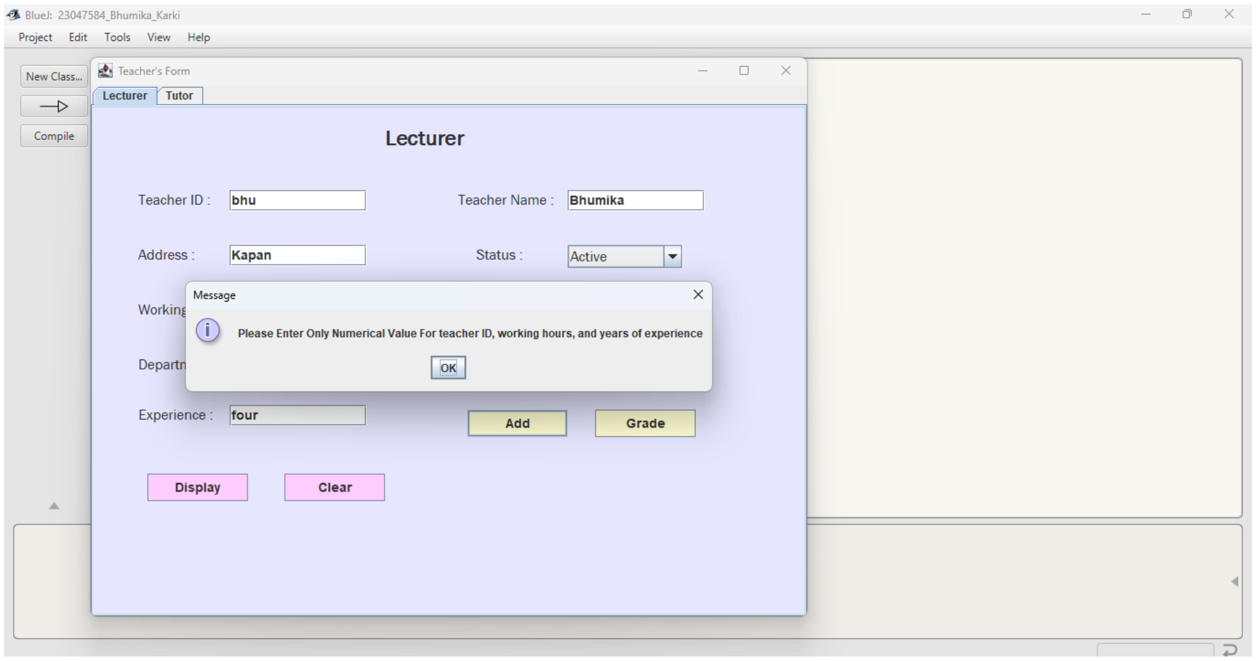
Figure 31: Solution of Syntax Error

6.2. Semantic Error

Semantic errors can result from using the incorrect operator, variable, or sequence of operations. This kind of mistake appears itself during the semantic analysis phase. Errors of this type are found during compilation (javatpoint, n.d.).

6.2.1. Error

When Java statements were utilized improperly, the code produced an error, which resulted in a semantic error. This happened because the Text Field was expecting an integer value, but the Java commands delivered string values instead. When an error occurs and the attempt block throws an exception, try and catch blocks are extremely useful. This results in the try block following a try-catch-do construct like this: try {do something;} catch (Exception e) {}.



Lecturer

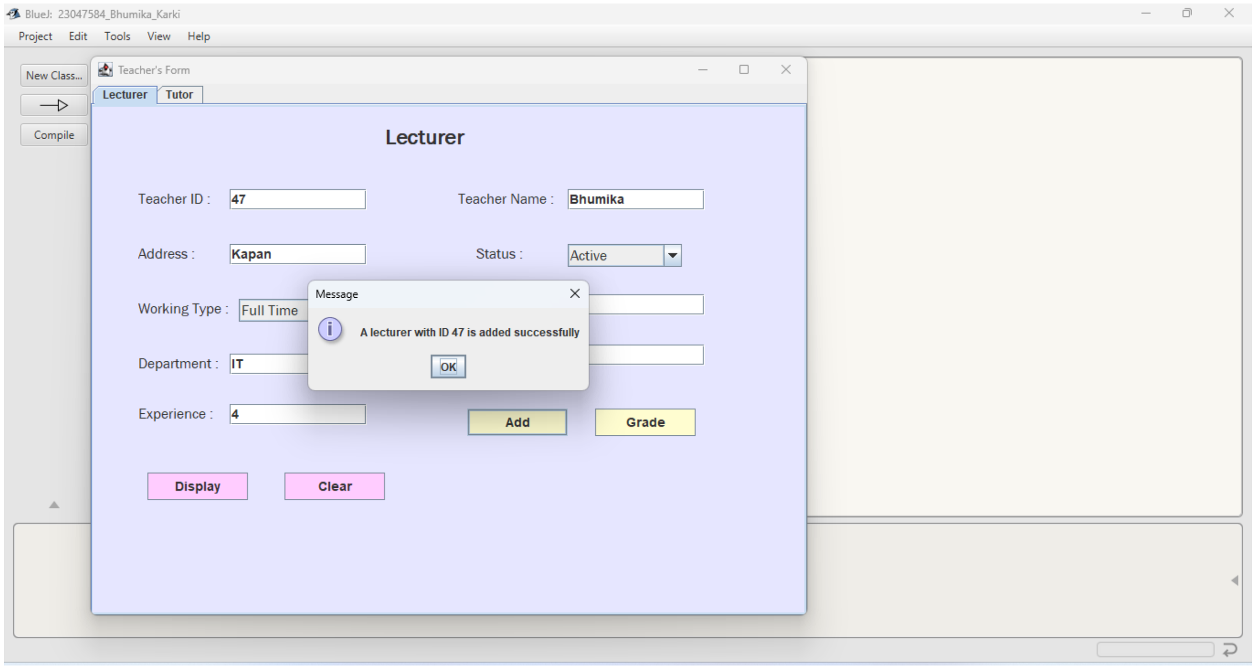
Teacher ID :	bhu	Teacher Name :	Bhumika
Address :	Kapan	Status :	Active
Experience :	four		

Please Enter Only Numerical Value For teacher ID, working hours, and years of experience

Figure 32: Semantic Error

6.2.2. Correction

Since the issue was so simple, it could be resolved quickly as well, allowing the Text Fields and Integer fields to be filled with accurate values. With the Text Fields, both String on String and Integer on Integer variables are used to make sure the compiler doesn't catch any exceptions.



Lecturer

Teacher ID : 47

Teacher Name : Bhumika

Address : Kapan

Status : Active

Working Type : Full Time

Department : IT

Experience : 4

Message

A lecturer with ID 47 is added successfully

OK

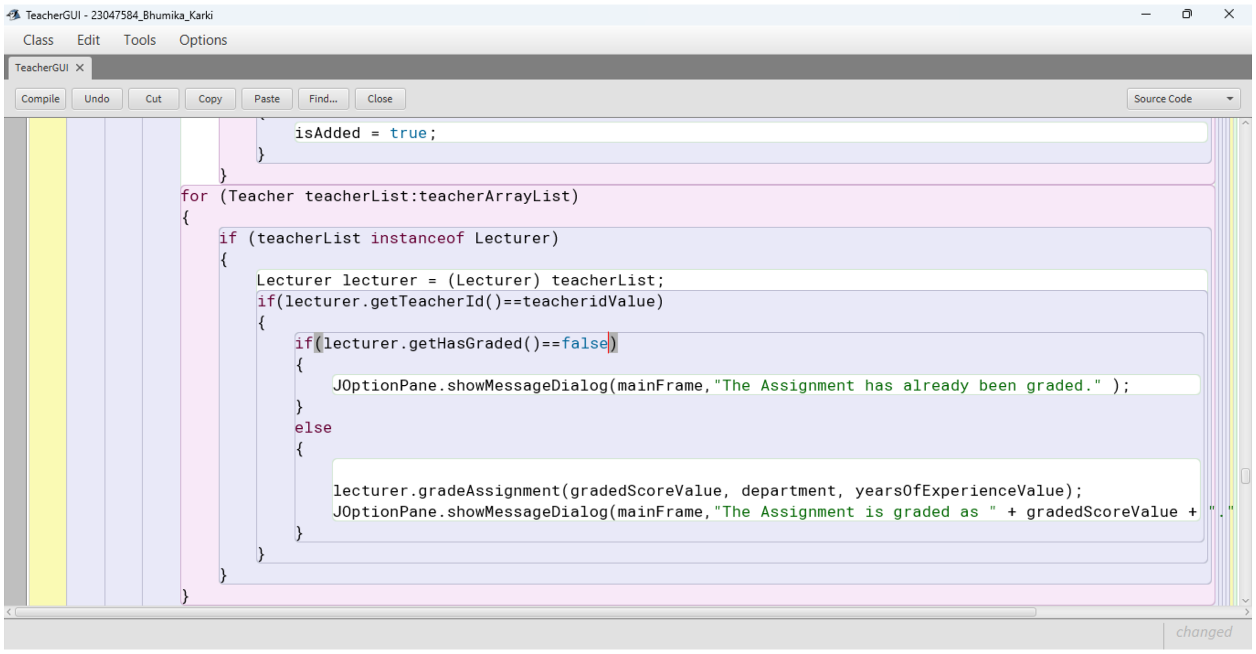
Figure 33: Solution of Semantic Error

6.3. Logical Error

The most difficult errors to locate and correct are logical ones. Because neither the Java compiler nor the JVM can identify them, so they are the most difficult ones to find. They are solely the programmer's responsibility. A logical flaw in the program will cause it to build and run correctly, but it won't produce the expected outcome. Application testers can identify logical problems by comparing the intended outcome of the program with the actual result (Mishra, 2022).

6.3.1. Error

Although the code compiles and functions properly, it gives an unexpected result. HasGraded changed from true to false on the if loop. The data were input, and when the Grade button was clicked, no change was observed.


TeacherGUI - 23047584_Bhumika_Karki
Class Edit Tools Options
TeacherGUI
Compile Undo Cut Copy Paste Find... Close
Source Code
changed

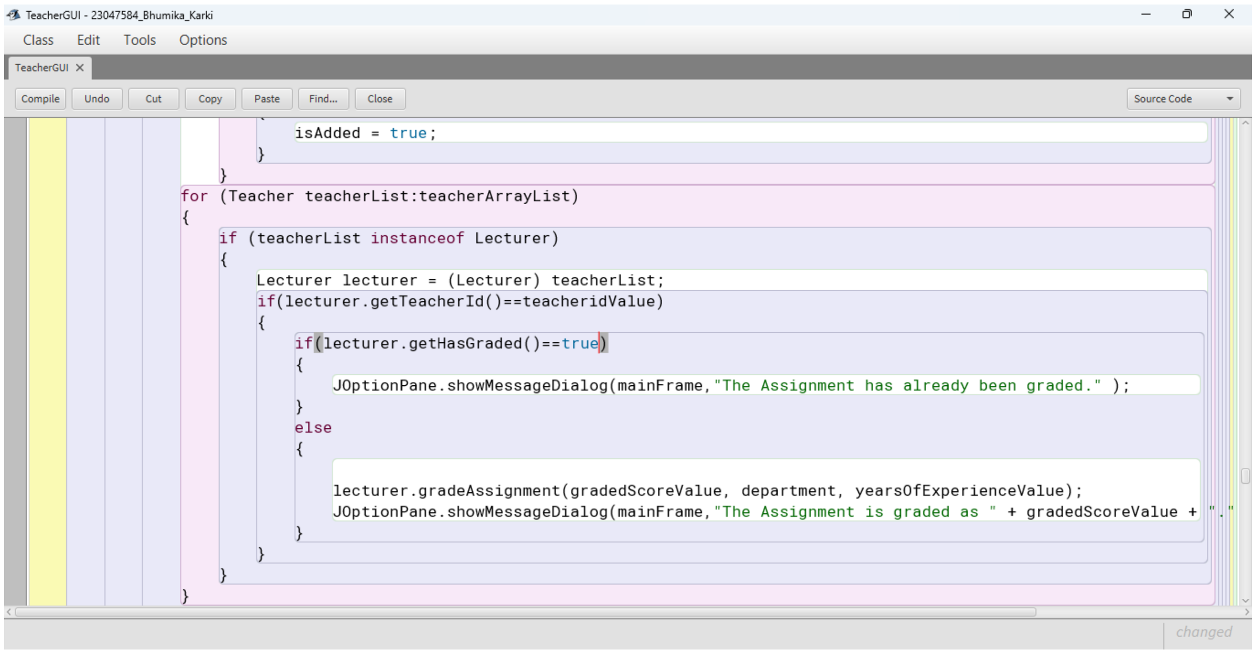
```
                isAdded = true;
            }
        }
        for (Teacher teacherList:teacherArrayList)
        {
            if (teacherList instanceof Lecturer)
            {
                Lecturer lecturer = (Lecturer) teacherList;
                if(lecturer.getTeacherId()==teacheridValue)
                {
                    if(lecturer.getHasGraded()==false)
                    {
                        JOptionPane.showMessageDialog(mainFrame,"The Assignment has already been graded." );
                    }
                    else
                    {

                        lecturer.gradeAssignment(gradedScoreValue, department, yearsOfExperienceValue);
                        JOptionPane.showMessageDialog(mainFrame,"The Assignment is graded as " + gradedScoreValue + "."
                    }
                }
            }
        }
```

Figure 34: Logical Error

6.3.2. Correction

The solution for this error would be simple, change the HasGraded value to true on the if loop for a correct output as suggested below.


TeacherGUI - 23047584_Bhumika_Karki
Class Edit Tools Options
TeacherGUI
Compile Undo Cut Copy Paste Find... Close
Source Code
changed

```
                isAdded = true;
            }
        }
        for (Teacher teacherList:teacherArrayList)
        {
            if (teacherList instanceof Lecturer)
            {
                Lecturer lecturer = (Lecturer) teacherList;
                if(lecturer.getTeacherId()==teacheridValue)
                {
                    if(lecturer.getHasGraded()==true)
                    {
                        JOptionPane.showMessageDialog(mainFrame,"The Assignment has already been graded." );
                    }
                    else
                    {

                        lecturer.gradeAssignment(gradedScoreValue, department, yearsOfExperienceValue);
                        JOptionPane.showMessageDialog(mainFrame,"The Assignment is graded as " + gradedScoreValue + "."
                    }
                }
            }
        }
```

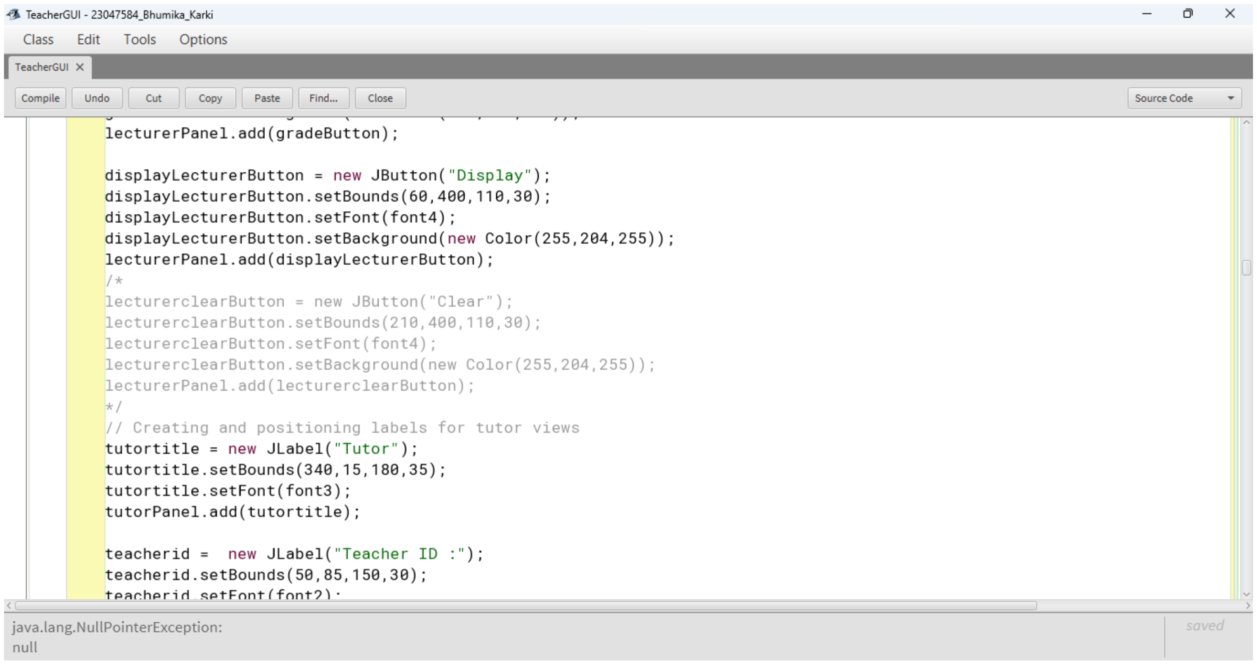
Figure 35: Solution of Logical Error

6.4. Runtime Error

Runtime errors occur after the program has successfully compiled, producing a ".class" file and giving no errors. But the program doesn't run correctly. These mistakes are found when the program is running, or during runtime. Because it lacks a way to identify these mistakes due to not having access to all runtime data, the Java compiler is unable to detect runtime faults. Java Virtual Machine (JVM) detects runtime errors while the program is executing. Known as exceptions, these runtime problems cause the program to end unexpectedly and display an error message. To deal with these runtime issues, we can use Java's exception handling features (Mishra, 2022).

6.3.1. Error

Both the block of code that handled events and the block of code that handled exceptions are affected by the error. Action listener was used to do this. Although compilation occurs, a Null Point Exception error message shows on the display panel upon calling the main method.


TeacherGUI - 23047584_Bhumika_Karki
Class Edit Tools Options
TeacherGUI
Compile Undo Cut Copy Paste Find... Close
Source Code
saved

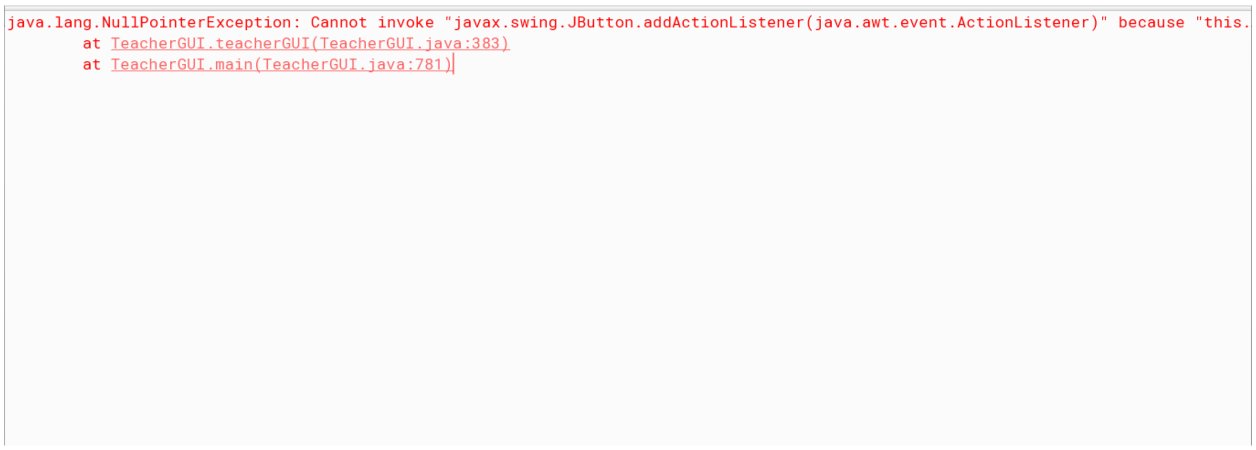
```
lecturerPanel.add(gradeButton);

displayLecturerButton = new JButton("Display");
displayLecturerButton.setBounds(60,400,110,30);
displayLecturerButton.setFont(font4);
displayLecturerButton.setBackground(new Color(255,204,255));
lecturerPanel.add(displayLecturerButton);
/*
lecturerclearButton = new JButton("Clear");
lecturerclearButton.setBounds(210,400,110,30);
lecturerclearButton.setFont(font4);
lecturerclearButton.setBackground(new Color(255,204,255));
lecturerPanel.add(lecturerclearButton);
*/
// Creating and positioning labels for tutor views
tutortitle = new JLabel("Tutor");
tutortitle.setBounds(340,15,180,35);
tutortitle.setFont(font3);
tutorPanel.add(tutortitle);

teacherid =  new JLabel("Teacher ID :");
teacherid.setBounds(50,85,150,30);
teacherid.setFont(font2);
```

```
java.lang.NullPointerException:
null
```

Figure 36: Runtime Error (1)

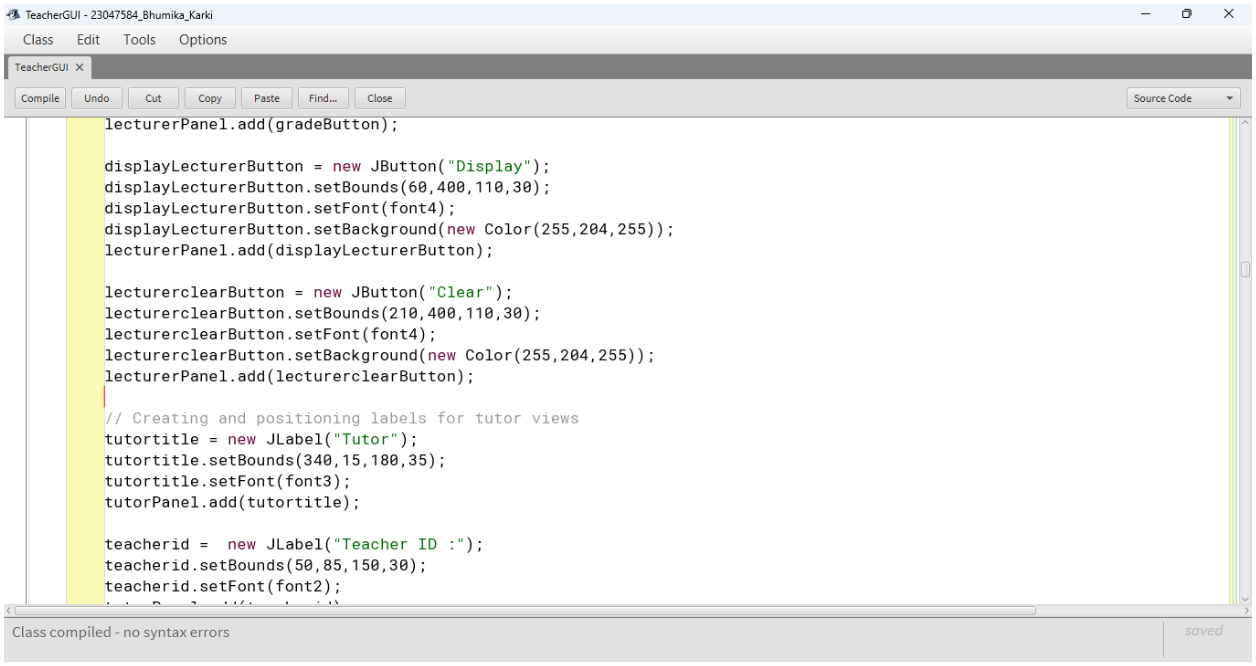


```
java.lang.NullPointerException: Cannot invoke "javax.swing.JButton.addActionListener(java.awt.event.ActionListener)" because "this.
	at TeacherGUI.teacherGUI(TeacherGUI.java:383)
	at TeacherGUI.main(TeacherGUI.java:781)
```

Figure 37: Runtime Error (2)

6.4.2. Correction

The correction to this error would be simple, follow the proper coding pattern, remove the commenting from that block and compile it which won't give any errors neither provide any exceptions when the main method is called.



```
lecturerPanel.add(gradeButton);

displayLecturerButton = new JButton("Display");
displayLecturerButton.setBounds(60,400,110,30);
displayLecturerButton.setFont(font4);
displayLecturerButton.setBackground(new Color(255,204,255));
lecturerPanel.add(displayLecturerButton);

lecturerclearButton = new JButton("Clear");
lecturerclearButton.setBounds(210,400,110,30);
lecturerclearButton.setFont(font4);
lecturerclearButton.setBackground(new Color(255,204,255));
lecturerPanel.add(lecturerclearButton);

// Creating and positioning labels for tutor views
tutortitle = new JLabel("Tutor");
tutortitle.setBounds(340,15,180,35);
tutortitle.setFont(font3);
tutorPanel.add(tutortitle);

teacherid =  new JLabel("Teacher ID :");
teacherid.setBounds(50,85,150,30);
teacherid.setFont(font2);
```

Class compiled - no syntax errors

saved

Figure 38: Solution of Runtime Error

7. Conclusion

Now that the coursework is finished, I can finally take account of what I've learned over the past few weeks. JSwing was easy to implement and install on the window applet's panel because the coursework was based on the material covered in class. finally, a simple online tutorial turned out to be quite beneficial to ensure that these coding sessions went off without a hitch and without mistakes. Throughout this entire research work, I learned about the Java Swing and AWT components, as well as Array Lists and exception handling. This has equipped me with the knowledge necessary to develop and design an intuitive and simple-to-use user interface. I discovered additional applications for the Array List as a result of my simultaneous work on it. I was able to recognise and catch the several kinds of exceptions that may occur while a program was running, as well as various abnormal situations that could emerge from unexpected user input, through the use of exception handling. However, the principles of event management proved to be somewhat difficult, and web study and teacher's manual were beneficial. Although I ran into a few problems when coding the sections on setting salary. Because of the grading assignments and setting salary procedure, expectations like creating a neat presentation of all the codes took a lot longer than expected.

I ran into a number of obstacles while completing this coursework. First off, the idea of object casting was not quite clear, which made it challenging to use in some sections of the code. Validating the form created difficulties, such as determining whether the user had left any fields empty. Because there were so many mistakes made when completing this coursework, there was an issue with the pop-up message that appeared when a button was selected. The actionPerformed method encountered some issues while accessing certain variables because the NullPointException was thrown several times. Writing the code to implement button functionality was challenging because there were many issues to resolve and conditions to meet, including adding, grading, setting salary, displaying, and removing tutors and lecturers. It was challenging for me to manage exceptions and validate the different unique inputs that users could provide to the JTextFields when I was creating this application. But by going through the course materials again, watching the lectures online, and making changes to my code, I was able to get beyond all of these obstacles. Consequently, it appeared that the coursework was mostly beneficial because I was able to build my very first application. Despite the fact that the coding process went smoothly, other tasks such as creating class diagrams and pseudocode, as well as documenting and reporting, took longer. Overall, at the end of the day I managed to do everything on schedule and deliver them in the proper manner.

8. Appendix

8.1. Teacher Class

```
public class Teacher
{
    private int teacherid;
    private String teacherName;
    private String address;
    private String workingType;
    private String employmentStatus;
    private int workingHours;

    public Teacher (int teacherid, String teacherName, String address, String
workingType, String employmentStatus)
    {
        this.teacherName=teacherName;
        this.teacherid=teacherid;
        this.address=address;
        this.workingType=workingType;
        this.employmentStatus=employmentStatus;
    }

    public int getTeacherId(){
        return this.teacherid;
    }
    public String getTeacherName(){
        return this.teacherName;
    }
    public String getAddress(){
        return this.address;
    }
    public String getWorkingType(){
        return this.workingType;
    }
    public String getEmploymentStatus(){
        return this.employmentStatus;
    }
    public int getWorkingHours(){
        return this.workingHours;
    }

    public void setWorkingHours(int workingHours){
        this.workingHours=workingHours;
    }

    public void displayInfo()
    {
        System.out.println("The ID of the teacher is " + this.teacherid);
        System.out.println("The name of the teacher is " + this.teacherName);
        System.out.println("The address of the teacher is " + this.address);
        System.out.println("The working type of the teacher is " + this.workingType);
        System.out.println("The employment status of the teacher is " +
this.employmentStatus);

        if (workingHours > 0)
        {
            System.out.println("The working hours of the teacher is " + this.workingHours);
        }
        else
        {
            System.out.println("Working hours is not assigned yet");
        }
    }
}
```

8.2. Lecturer Class

```
public class Lecturer extends Teacher
{
    private String department;
    private int yearsOfExperience;
    private int gradedScore;
    private boolean hasGraded;

    Lecturer (int teacherid, String teacherName, String address, String workingType,
String employmentStatus, int workingHours, String department, int yearsOfExperience)
    {
        super(teacherid, teacherName, address, workingType, employmentStatus);
        setWorkingHours(workingHours);
        this.department=department;
        this.yearsOfExperience=yearsOfExperience;
        this.gradedScore=0;
        this.hasGraded=false;
    }

    public String getDepartment(){
        return this.department;
    }
    public int getYearsOfExperience(){
        return this.yearsOfExperience;
    }
    public int getGradedScore(){
        return this.gradedScore;
    }
    public boolean getHasGraded(){
        return this.hasGraded;
    }

    public void setGradedScore(int gradedScore)
    {
        this.gradedScore=gradedScore;
    }

    public void gradeAssignment(int gradedScore, String studentDepartment, int yearsOfExperience)
    {
        if (hasGraded==false && yearsOfExperience >= 5 && department.equals(studentDepartment))
        {
            if (gradedScore >= 70)
            {
                System.out.print("A");
            }
            else if (gradedScore >= 60)
            {
```

```
            System.out.print("B");
         }
         else if (gradedScore >=50)
         {
            System.out.print("C");
         }
         else if (gradedScore >= 40)
         {
            System.out.print("D");
         }
         else
         {
            System.out.print("E");
         }
         hasGraded = true;
      }
      else
      {
         System.out.print("Assignment is not graded yet.");
      }
   }

   public void display()
      {
         super.displayInfo();   // Calling the display method in the Teacher class
         System.out.println("The department of the teacher is " + this.department);
         System.out.println("The years of experience of the teacher is " + 
this.yearsOfExperience);
         if (hasGraded == true)
```

```
            {
                System.out.println("The graded score is " + this.gradedScore);
            }
            else
            {
                System.out.println("The score has not been graded yet");
            }
        }
    }
```

8.3. Tutor Class

```
public class Tutor extends Teacher
{
    // declaring the variables
    private double salary;
    private String specialization;
    private String academicQualifications;
    private int performanceIndex;
    private boolean isCertified;


    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);
        setWorkingHours(workingHours);
        this.salary = salary;
        this.specialization = specialization;
        this.academicQualifications = academicQualifications;
        this.performanceIndex = performanceIndex;
        this.isCertified = false;
    }

    public double getSalary(){
        return this.salary;
    }
    public String getSpecialization(){
        return this.specialization;
    }
    public String getAcademicQualifications(){
        return this.academicQualifications;
    }
    public int getPerformanceIndex(){
        return this.performanceIndex;
    }
    public boolean getIsCertified(){
        return this.isCertified;
    }

    public void setSalary (double newSalary, int newPerformanceIndex)
    {
        if (!isCertified && newPerformanceIndex > 5 && getWorkingHours() > 20)
        {
            double appraisal = 0;
            if (newPerformanceIndex >= 5 && newPerformanceIndex <= 7)
            {
                appraisal = 0.05;
            }
            else if (newPerformanceIndex >= 8 && newPerformanceIndex <=9)
            {
                appraisal = 0.1;
            }
            else if (newPerformanceIndex == 10)
            {
                appraisal = 0.2;
            }
            this.salary = salary + (appraisal * salary);
            this.performanceIndex = newPerformanceIndex;
            this.isCertified = true; // setting isCertified to true after appraisal
            System.out.println("Salary updated successfully to " + this.salary);
        }
        else
        {
            System.out.println("The salary cannot be approved. The tutor is not certified yet.");
        }
    }

    public void removeTutor()
     {
         if (!isCertified)
           {
               this.salary = 0;
```

```
                this.specialization = "";
                this.academicQualifications = "";
                this.performanceIndex = 0;
                this.isCertified = false;
            }
            else
            {
                System.out.println("Tutor is certified. Can't be removed");
            }
        }

    public void display()
        {
            if (isCertified)
            {
                super.displayInfo();    // Calling the display method in the Teacher class
            }
            else
            {
                super.displayInfo();    // Calling the display method in the Teacher class
                System.out.println("The salary of the tutor is " + this.salary);
                System.out.println("The specialization of the tutor is " + this.specialization);
                System.out.println("The academic qualifications of the tutor is " + 
this.academicQualifications);
                System.out.println("The performance index of the tutor is " + 
this.performanceIndex);
            }
        }
}
```

8.4. TeacherGUI Class

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JRadioButton;
import javax.swing.JTabbedPane;
import javax.swing.JOptionPane;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Color;
import java.util.ArrayList;

public class TeacherGUI implements ActionListener
{
    private JFrame mainFrame;

    private JPanel lecturerPanel;
    private JPanel tutorPanel;

    private JTabbedPane lecturerTabbedPane;
    private JTabbedPane tutorTabbedPane;



    private JLabel lecturertitle;
    private JLabel tutortitle;
    private JLabel teacherid;
    private JLabel teacherName;
    private JLabel address;
    private JLabel employmentStatus;
    private JLabel workingType;
    private JLabel workingHours;
    private JLabel department;
    private JLabel yearsOfExperience;
    private JLabel gradedScore;
    private JLabel salary;
    private JLabel specialization;
    private JLabel academicQualifications;
    private JLabel performanceIndex;

    private JTextField teacheridTextField1;
    private JTextField teacheridTextField2;
    private JTextField teacherNameTextField1;
    private JTextField teacherNameTextField2;
    private JTextField addressTextField1;
    private JTextField addressTextField2;
    private JTextField workingHoursTextField1;
    private JTextField workingHoursTextField2;
    private JTextField departmentTextField;
    private JTextField yearsOfExperienceTextField;
    private JTextField salaryTextField;
    private JTextField specializationTextField;
    private JTextField academicQualificationTextField;
    private JTextField performanceIndexTextField;
    private JTextField gradedScoreTextField;

    private JButton addLecturerButton;
    private JButton addTutorButton;
    private JButton gradeButton;
    private JButton setSalaryButton;
    private JButton removeTutorButton;
    private JButton lecturerclearButton;
    private JButton tutorclearButton;
    private JButton displayLecturerButton;
    private JButton displayTutorButton;

    private JComboBox workingTypeJComboBox1;
    private JComboBox workingTypeJComboBox2;
    private JComboBox statusComboBox1;
    private JComboBox statusComboBox2;

    private String [] comboBoxType = {"Part Time", "Full Time"};

    private String [] comboBoxStatus = {"Active", "Not Active"};

    ArrayList <Teacher> teacherArrayList = new ArrayList <> ();

    public void teacherGUI()
    {
        mainFrame = new JFrame("Teacher's Form");
        mainFrame.setSize(800,620);
```

```
mainFrame.setVisible(true);
mainFrame.setDefaultCloseOperation(mainFrame.EXIT_ON_CLOSE);

lecturerPanel = new JPanel();
lecturerPanel.setSize(800,620);
lecturerPanel.setLayout(null);
lecturerPanel.setBackground(new Color(230,230,255));

tutorPanel = new JPanel();
tutorPanel.setSize(800,620);
tutorPanel.setLayout(null);
tutorPanel.setBackground(new Color(230,230,255));

lecturerTabbedPane = new JTabbedPane();
lecturerTabbedPane.add("Lecturer",lecturerPanel);
lecturerTabbedPane.add("Tutor",tutorPanel);

mainFrame.add(lecturerTabbedPane);

Font font2 = new Font ("Franklin Gothic Book", Font.PLAIN,16);
Font font3 = new Font ("Franklin Gothic Book", Font.BOLD,23);
Font font4 = new Font ("Arial", Font.BOLD,14);

lecturertitle = new JLabel("Lecturer");
lecturertitle.setBounds(320,15,180,35);
lecturertitle.setFont(font3);
lecturerPanel.add(lecturertitle);
```

```
teacherid =  new JLabel("Teacher ID :");
teacherid.setBounds(50,85,150,30);
teacherid.setFont(font2);
lecturerPanel.add(teacherid);

teacherName = new JLabel("Teacher Name :");
teacherName.setBounds(400,85,150,30);
teacherName.setFont(font2);
lecturerPanel.add(teacherName);

address = new JLabel("Address :");
address.setBounds(50,145,150,30);
address.setFont(font2);
lecturerPanel.add(address);

employmentStatus= new JLabel("     Status :");
employmentStatus.setBounds(400,145,150,30);
employmentStatus.setFont(font2);
lecturerPanel.add(employmentStatus);

workingType = new JLabel("Working Type :");
workingType.setBounds(50,205,150,30);
workingType.setFont(font2);
lecturerPanel.add(workingType);

workingHours = new JLabel("Working Hours :");
workingHours.setBounds(400,200,150,30);
workingHours.setFont(font2);
```

```
lecturerPanel.add(workingHours);

gradedScore = new JLabel("Graded Score :");
gradedScore.setBounds(400,255,150,30);
gradedScore.setFont(font2);
lecturerPanel.add(gradedScore);

department = new JLabel("Department :");
department.setBounds(50,265,150,30);
department.setFont(font2);
lecturerPanel.add(department);

yearsOfExperience = new JLabel("Experience :");
yearsOfExperience.setBounds(50,320,150,30);
yearsOfExperience.setFont(font2);
lecturerPanel.add(yearsOfExperience);

teacheridTextField1 = new JTextField();
teacheridTextField1.setBounds(150,90,150,23);
teacheridTextField1.setFont(font4);
lecturerPanel.add(teacheridTextField1);

teacherNameTextField1 = new JTextField();
teacherNameTextField1.setBounds(520,90,150,23);
teacherNameTextField1.setFont(font4);
lecturerPanel.add(teacherNameTextField1);

addressTextField1 = new JTextField();
```

```
addressTextField1.setBounds(150,150,150,23);
addressTextField1.setFont(font4);
lecturerPanel.add(addressTextField1);

workingTypeJComboBox1 = new JComboBox(comboBoxType);
workingTypeJComboBox1.setBounds(160,210,125,25);
workingTypeJComboBox1.setFont(font2);
lecturerPanel.add(workingTypeJComboBox1);

statusComboBox1 = new JComboBox(comboBoxStatus);
statusComboBox1.setBounds(520,150,125,25);
statusComboBox1.setFont(font2);
lecturerPanel.add(statusComboBox1);

departmentTextField = new JTextField();
departmentTextField.setBounds(150,270,150,23);
departmentTextField.setFont(font4);
lecturerPanel.add(departmentTextField);

yearsOfExperienceTextField = new JTextField();
yearsOfExperienceTextField.setBounds(150,325,150,23);
yearsOfExperienceTextField.setFont(font4);
lecturerPanel.add(yearsOfExperienceTextField);

workingHoursTextField1 = new JTextField();
workingHoursTextField1.setBounds(520,205,150,23);
workingHoursTextField1.setFont(font4);
lecturerPanel.add(workingHoursTextField1);
```

```
gradedScoreTextField = new JTextField();
gradedScoreTextField.setBounds(520,260,150,23);
gradedScoreTextField.setFont(font4);
lecturerPanel.add(gradedScoreTextField);

addLecturerButton = new JButton("Add");
addLecturerButton.setBounds(410,330,110,30);
addLecturerButton.setFont(font4);
addLecturerButton.setBackground(new Color(255,253,208));
lecturerPanel.add(addLecturerButton);

gradeButton = new JButton("Grade");
gradeButton.setBounds(550,330,110,30);
gradeButton.setFont(font4);
gradeButton.setBackground(new Color(255,253,208));
lecturerPanel.add(gradeButton);

displayLecturerButton = new JButton("Display");
displayLecturerButton.setBounds(60,400,110,30);
displayLecturerButton.setFont(font4);
displayLecturerButton.setBackground(new Color(255,204,255));
lecturerPanel.add(displayLecturerButton);

lecturerclearButton = new JButton("Clear");
lecturerclearButton.setBounds(210,400,110,30);
lecturerclearButton.setFont(font4);
lecturerclearButton.setBackground(new Color(255,204,255));
```

```
lecturerPanel.add(lecturerclearButton);

tutortitle = new JLabel("Tutor");
tutortitle.setBounds(340,15,180,35);
tutortitle.setFont(font3);
tutorPanel.add(tutortitle);

teacherid =  new JLabel("Teacher ID :");
teacherid.setBounds(50,85,150,30);
teacherid.setFont(font2);
tutorPanel.add(teacherid);

teacherName = new JLabel("Teacher Name :");
teacherName.setBounds(400,85,150,30);
teacherName.setFont(font2);
tutorPanel.add(teacherName);

address = new JLabel("Address :");
address.setBounds(50,145,150,30);
address.setFont(font2);
tutorPanel.add(address);

employmentStatus = new JLabel("     Status :");
employmentStatus.setBounds(400,145,150,30);
employmentStatus.setFont(font2);
tutorPanel.add(employmentStatus);

workingType = new JLabel("Working Type :");
```

```
workingType.setBounds(50,205,150,30);
workingType.setFont(font2);
tutorPanel.add(workingType);

workingTypeJComboBox2 = new JComboBox(comboBoxType);
workingTypeJComboBox2.setBounds(160,210,125,25);
workingTypeJComboBox2.setFont(font2);
tutorPanel.add(workingTypeJComboBox2);

workingHours = new JLabel("Working Hours :");
workingHours.setBounds(400,205,150,30);
workingHours.setFont(font2);
tutorPanel.add(workingHours);

salary = new JLabel("Salary :");
salary.setBounds(50,265,150,30);
salary.setFont(font2);
tutorPanel.add(salary);

specialization = new JLabel("Specialization :");
specialization.setBounds(400,265,150,30);
specialization.setFont(font2);
tutorPanel.add(specialization);

academicQualifications = new JLabel("Qualification :");
academicQualifications.setBounds(50,325,150,30);
academicQualifications.setFont(font2);
tutorPanel.add(academicQualifications);
```

```
performanceIndex = new JLabel("Performance Index :");
performanceIndex.setBounds(400,325,150,30);
performanceIndex.setFont(font2);
tutorPanel.add(performanceIndex);

teacheridTextField2 = new JTextField();
teacheridTextField2.setBounds(150,90,150,23);
teacheridTextField2.setFont(font4);
tutorPanel.add(teacheridTextField2);

teacherNameTextField2 = new JTextField();
teacherNameTextField2.setBounds(520,90,150,23);
teacherNameTextField2.setFont(font4);
tutorPanel.add(teacherNameTextField2);

addressTextField2 = new JTextField();
addressTextField2.setBounds(150,150,150,23);
addressTextField2.setFont(font4);
tutorPanel.add(addressTextField2);

statusComboBox2 = new JComboBox(comboBoxStatus);
statusComboBox2.setBounds(520,150,125,25);
statusComboBox2.setFont(font2);
tutorPanel.add(statusComboBox2);

workingHoursTextField2 = new JTextField();
workingHoursTextField2.setBounds(520,210,150,23);
```

```
workingHoursTextField2.setFont(font4);
tutorPanel.add(workingHoursTextField2);

salaryTextField = new JTextField();
salaryTextField.setBounds(150,270,150,23);
salaryTextField.setFont(font4);
tutorPanel.add(salaryTextField);

specializationTextField = new JTextField();
specializationTextField.setBounds(520,270,150,23);
specializationTextField.setFont(font4);
tutorPanel.add(specializationTextField);

academicQualificationTextField = new JTextField();
academicQualificationTextField.setBounds(150,330,150,23);
academicQualificationTextField.setFont(font4);
tutorPanel.add(academicQualificationTextField);

performanceIndexTextField = new JTextField();
performanceIndexTextField.setBounds(550,330,150,23);
performanceIndexTextField.setFont(font4);
tutorPanel.add(performanceIndexTextField);

addTutorButton = new JButton("Add");
addTutorButton.setBounds(420,395,110,30);
addTutorButton.setFont(font4);
addTutorButton.setBackground(new Color(255,253,208));
tutorPanel.add(addTutorButton);
```

```
setSalaryButton = new JButton("Set Salary");
setSalaryButton.setBounds(570,395,110,30);
setSalaryButton.setFont(font4);
setSalaryButton.setBackground(new Color(255,253,208));
tutorPanel.add(setSalaryButton);

removeTutorButton = new JButton("Remove");
removeTutorButton.setBounds(490,460,110,30);
removeTutorButton.setFont(font4);
removeTutorButton.setBackground(new Color(255,253,208));
tutorPanel.add(removeTutorButton);

displayTutorButton = new JButton("Display");
displayTutorButton.setBounds(60,410,110,30);
displayTutorButton.setFont(font4);
displayTutorButton.setBackground(new Color(255,204,255));
tutorPanel.add(displayTutorButton);

tutorclearButton = new JButton("Clear");
tutorclearButton.setBounds(210,410,110,30);
tutorclearButton.setFont(font4);
tutorclearButton.setBackground(new Color(255,204,255));
tutorPanel.add(tutorclearButton);

addLecturerButton.addActionListener(this);
gradeButton.addActionListener(this);
displayLecturerButton.addActionListener(this);
```

```
        lecturerclearButton.addActionListener(this);
        addTutorButton.addActionListener(this);
        setSalaryButton.addActionListener(this);
        removeTutorButton.addActionListener(this);
        displayTutorButton.addActionListener(this);
        tutorclearButton.addActionListener(this);
    }

@Override
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==lecturerclearButton)
    {
        teacheridTextField1.setText("");
        teacherNameTextField1.setText("");
        addressTextField1.setText("");
        workingHoursTextField1.setText("");
        workingTypeJComboBox1.setSelectedIndex(-1);
        statusComboBox1.setSelectedIndex(-1);
        yearsOfExperienceTextField.setText("");
        gradedScoreTextField.setText("");
        departmentTextField.setText("");
    }
    else if(e.getSource()==tutorclearButton)
    {
        teacheridTextField2.setText("");
        teacherNameTextField2.setText("");
        addressTextField2.setText("");
        workingHoursTextField2.setText("");
        workingTypeJComboBox2.setSelectedIndex(-1);
        statusComboBox2.setSelectedIndex(-1);
        salaryTextField.setText("");
        specializationTextField.setText("");
        academicQualificationTextField.setText("");
        performanceIndexTextField.setText("");
    }
    else if(e.getSource()==addLecturerButton)
    {
        String teacherid = teacheridTextField1.getText();
        String teacherName = teacherNameTextField1.getText();
        String address = addressTextField1.getText();
        String workingType = (String) workingTypeJComboBox1.getSelectedItem();
        String employmentStatus = (String) statusComboBox1.getSelectedItem();
        String workingHours = workingHoursTextField1.getText();
        String department = departmentTextField.getText();
        String yearsOfExperience = yearsOfExperienceTextField.getText();
        boolean isAdded = false;
        try
        {
if(teacherid.isEmpty()||teacherName.isEmpty()||address.isEmpty()||workingHours.isEmp
ty()||
            yearsOfExperience.isEmpty())
            {
                throw new Exception();
            }
            else
```

```
         {
            int teacheridValue = Integer.parseInt(teacherid);
            int workingHoursValue = Integer.parseInt(workingHours);
            int yearsOfExperienceValue = Integer.parseInt(yearsOfExperience);
            if(department.matches(".*\\d.*")){
               JOptionPane.showMessageDialog(mainFrame,"Only Alphabetical
characters are allowed");
         }

         else
         {
            for (Teacher teacherList:teacherArrayList)
            {
               if (teacherList instanceof Lecturer)
               {
                  Lecturer lecturer = (Lecturer) teacherList;
                  if((lecturer.getTeacherId())==(teacheridValue))
                  {
                     isAdded = true;
                  }
               }
            }
            if (isAdded == true)
            {
               JOptionPane.showMessageDialog(mainFrame,"A teacher with ID " +
teacheridValue + " has already been added.");
            }
            else
            {
               Lecturer lecturer = new Lecturer(teacheridValue, teacherName, address, 
workingType, employmentStatus,
                                workingHoursValue, department, yearsOfExperienceValue);
               teacherArrayList.add(lecturer);
               JOptionPane.showMessageDialog(mainFrame,"A lecturer with ID " + 
teacheridValue + " is added successfully");


            }
         }
      }
      }


      catch(NumberFormatException exc)
      {
         JOptionPane.showMessageDialog(mainFrame,"Please Enter Only Numerical 
Value For teacher ID, working hours, and years of experience");
      }
      catch(Exception ex)
      {
         JOptionPane.showMessageDialog(mainFrame,"Please Do Not Leave Any Text 
Field Empty");
      }
   }
   else if(e.getSource()==addTutorButton)
    {
      String teacherid = teacheridTextField2.getText();
      String teacherName = teacherNameTextField2.getText();
      String address = addressTextField2.getText();
      String workingType = (String) workingTypeJComboBox2.getSelectedItem();
      String employmentStatus = (String) statusComboBox2.getSelectedItem();
    String workingHours = workingHoursTextField2.getText();
    String salary = salaryTextField.getText();
    String specialization = specializationTextField.getText();
    String academicQualification = academicQualificationTextField.getText();
    String performanceIndex = performanceIndexTextField.getText();
    boolean isAdded = false;
    try
    {
if(teacherid.isEmpty()||teacherName.isEmpty()||address.isEmpty()||workingHours.isEmp
ty()||salary.isEmpty()||
specialization.isEmpty()||academicQualification.isEmpty()||performanceIndex.isEmpty())
        {
            JOptionPane.showMessageDialog(mainFrame,"Please Do Not Leave Any
Text Field Empty.");
        }
        else
        {
            int teacheridValue = Integer.parseInt(teacherid);
            int workingHoursValue = Integer.parseInt(workingHours);
            double salaryValue = Double.parseDouble(salary);
            int performanceIndexValue = Integer.parseInt(performanceIndex);

            {
                for (Teacher teacherList:teacherArrayList)
                {
                if(teacherList instanceof Tutor)
                {
                    Tutor tutor = (Tutor) teacherList;
                    if((tutor.getTeacherId())==(teacheridValue))
                    {
                        isAdded = true;
                    }
                }
                }
            }
            if(isAdded==true)
            {
                JOptionPane.showMessageDialog(mainFrame,"A Tutor with ID " +
teacheridValue + " has already been added.");
            }
            else
            {
                Tutor tutor = new Tutor(teacheridValue, teacherName, address,
workingType, employmentStatus,
                            workingHoursValue, salaryValue,
academicQualification,specialization,performanceIndexValue);
                teacherArrayList.add(tutor);
                JOptionPane.showMessageDialog(mainFrame,"A Tutor with ID " +
teacheridValue + " is added successfully.");
            }
        }
    }
    catch(NumberFormatException exc)
    {
        JOptionPane.showMessageDialog(mainFrame,"Please Enter Only Numerical
Value for teacherid,workingHours,salary, and performanceIndex");
    }
}
```

```
    else if(e.getSource()==gradeButton)
    {
        String teacherid = teacheridTextField1.getText();
        String gradedScore = gradedScoreTextField.getText();
        String department = departmentTextField.getText();
        String yearsOfExperience = yearsOfExperienceTextField.getText();
        boolean isAdded = false;
        try
        {
if(teacherid.isEmpty()||gradedScore.isEmpty()||department.isEmpty()||yearsOfExperienc
e.isEmpty())
            {
                JOptionPane.showMessageDialog(mainFrame,"Please Do Not Leave Any
Text Field Empty.");
            }
            else
            {
                int teacheridValue = Integer.parseInt(teacherid);
                int yearsOfExperienceValue = Integer.parseInt(yearsOfExperience);
                int gradedScoreValue = Integer.parseInt(gradedScore);
                if(department.matches(".*\\d.*"))
                {
                    JOptionPane.showMessageDialog(mainFrame,"Only Alphabetical
characters are allowed");
                }
                else
                {
                    for(int i = 0; i<teacherArrayList.size(); i++)
                    {
```

```
                if((teacherArrayList.get(i).getTeacherId())==teacheridValue)
                {
                    isAdded = true;
                }
            }
        for (Teacher teacherList:teacherArrayList)
        {
            if (teacherList instanceof Lecturer)
            {
                Lecturer lecturer = (Lecturer) teacherList;
                if(lecturer.getTeacherId()==teacheridValue)
                {
                    if(lecturer.getHasGraded()==true)
                    {
                        JOptionPane.showMessageDialog(mainFrame,"The Assignment
has already been graded." );
                    }
                    else
                    {


                        lecturer.gradeAssignment(gradedScoreValue, department,
yearsOfExperienceValue);
                        JOptionPane.showMessageDialog(mainFrame,"The Assignment is
graded as " + gradedScoreValue + "." );
                    }
                }
            }
        }
    }
```

```
         }
         if(isAdded=false)
         {
            JOptionPane.showMessageDialog(mainFrame,"A lecturer with ID " +
teacherid + " is not certified.");
         }
      }
      catch(NumberFormatException exc)
      {
         JOptionPane.showMessageDialog(mainFrame,"Please Enter Only Numerical
Value For teacher ID, graded score, and years of experience");
      }
      catch(Exception ex)
      {
         JOptionPane.showMessageDialog(mainFrame,"Please Do Not Leave Any Text
Field Empty");
      }
   }
   else if(e.getSource()==displayLecturerButton)
   {

      if(teacherArrayList.size()==0)
      {
         JOptionPane.showMessageDialog(mainFrame,"The ArrayList is Empty.");
      }
      else
      {
         for(Teacher teacherList:teacherArrayList)
         {
            {
                Lecturer lecturer = (Lecturer) teacherList;
                lecturer.display();
            }
        }
    }
}
else if(e.getSource()==displayTutorButton)
{
    if(teacherArrayList.size()==0)
    {
        JOptionPane.showMessageDialog(mainFrame,"The ArrayList is Empty.");
    }
    else
    {
        for(Teacher teacherList:teacherArrayList)
        {
            if (teacherList instanceof Tutor)
            {
                Tutor tutor = (Tutor) teacherList;
                tutor.display();
            }
        }
    }
}
else if(e.getSource()==setSalaryButton)
{
    String teacherid = teacheridTextField2.getText();
```

```
        String salary = salaryTextField.getText();
        String performanceIndex = performanceIndexTextField.getText();
        boolean isAdded = false;

        try
        {
            if (teacherid.isEmpty() || salary.isEmpty() || performanceIndex.isEmpty())
            {
                JOptionPane.showMessageDialog(mainFrame, "Please Do Not Leave Any
Text Field Empty.");
            } else
            {
                int teacheridValue = Integer.parseInt(teacherid);
                double salaryValue = Double.parseDouble(salary);
                int performanceIndexValue = Integer.parseInt(performanceIndex);

                for (int i = 0; i < teacherArrayList.size(); i++)
                {
                    if (teacherArrayList.get(i).getTeacherId() == teacheridValue)
                    {
                        isAdded = true;
                    }
                }
                for(Teacher teacherList : teacherArrayList)
                {
                    if(teacherList instanceof Tutor)
                    {
                        Tutor tutor = (Tutor) teacherList;
                        if(tutor.getTeacherId()==teacheridValue)
                    {
                        if(tutor.getIsCertified()==true)
                        {
                            JOptionPane.showMessageDialog(mainFrame,"The salary has
already been updated.");
                        }
                        else
                        {
                            tutor.setSalary(salaryValue, performanceIndexValue);
                            JOptionPane.showMessageDialog(mainFrame,"The salary has
been updated.");
                        }
                    }
                }
            }
        }
        if (!isAdded)
        {
            JOptionPane.showMessageDialog(mainFrame, "A tutor with ID " + teacherid + "
has not been added.");
        }
    }
        catch (NumberFormatException exc)
        {
            JOptionPane.showMessageDialog(mainFrame, "Please Enter Only Numerical
Value For teacher ID, salary, and performance index.");
        }
        catch (Exception ex)
        {
            JOptionPane.showMessageDialog(mainFrame, "An error occurred: " + 
ex.getMessage());
        }
    }
    else if(e.getSource() == removeTutorButton)
    {
        String teacherId = teacheridTextField2.getText();

        try
        {
            if(teacherId.isEmpty())
            {
                JOptionPane.showMessageDialog(mainFrame,"Please enter the ID of the 
tutor to remove.");
            }
            else
            {
                int teacherIdValue = Integer.parseInt(teacherId);
                boolean isAdded = false;

                for(int i = 0; i < teacherArrayList.size(); i++)
                {
                    Teacher teacher = teacherArrayList.get(i);
                    if(teacher instanceof Tutor && teacher.getTeacherId() == teacherIdValue)
                    {
                        Tutor tutor = (Tutor) teacher;
                        if(tutor.getIsCertified())
                        {
```

```
                    JOptionPane.showMessageDialog(mainFrame,"The tutor with ID " +
teacherIdValue + " is certified and cannot be removed.");
                }
                else
                {
                    teacherArrayList.remove(i);
                    JOptionPane.showMessageDialog(mainFrame,"The tutor with ID " +
teacherIdValue + " has been removed successfully.");
                }
                isAdded = true;
                break; // Stop searching after finding the tutor
            }
        }
            if(!isAdded)
        {
            JOptionPane.showMessageDialog(mainFrame,"No tutor found with ID " +
teacherIdValue);
        }
    }
    }
    catch(NumberFormatException ex)
    {
        JOptionPane.showMessageDialog(mainFrame,"Please enter a valid ID.");
    }
  }
}
```

```
public static void main(String[]args)
{
    new TeacherGUI().teacherGUI();
}
}
```

9. Bibliography

(2024). https://www.simplilearn.com/tutorials/java-tutorial/exception-handling-in-java

Computer, H. (2020). https://www.computerhope.com/jargon/d/drawio.htm

DevX. (2023). https://www.devx.com/terms/java-swing/

GeeksforGeeks. (2024). https://www.geeksforgeeks.org/introduction-to-java/

javaguides. (n.d.). https://www.javaguides.net/2019/01/programming-errors-in-java-with-examples.html

javatpoint. (n.d.). https://www.javatpoint.com/semantic-error

LinkedIn. (n.d.). https://www.linkedin.com/learning/paths/master-microsoft-word

Mishra, M. (2022). https://www.scaler.com/topics/types-of-errors-in-java/

Roberts, S. (2023, April 13). https://www.theknowledgeacademy.com/blog/methods-in-java/

Scaler. (2023). https://www.scaler.com/topics/event-handling-in-java/

softonic. (2023). https://bluej.en.softonic.com/

Systems, S. (n.d.). https://sparxsystems.com/enterprise_architect_user_guide/16.1/modeling_languages/classdiagram.html

University, W. M. (2020). https://cs.wmich.edu/gupta/teaching/cs4310/lectureNotes_cs4310/PseudocodeBasics.pdf