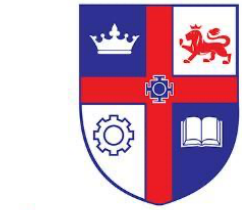
LONDON METROPOLITAN UNIVERSITY



islington college
(इस्लिङ्टन कलेज)

CS4001NI Programming

30% Individual Coursework

2023-24 Autumn

Student Name: Shivam Kumar Thakur

London Met ID: 23050396

College ID: NP01CP4A230301

Group: C11

Assignment Due Date: Friday, May 10, 2024

Assignment Submission Date: Friday, May 10, 2024

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

Table of Figure

Table of Tables

1. Introduction

1.1 Java

Java is a high-level object-oriented programming language developed by James Gosling and his team at Sun Microsystems in 1995 which was designed to be platform independent meaning that it will run on any device which has a Java Virtual Machine on it regardless of hardware and operating system. Despite after 20 years of its release it still runs on billions of devices worldwide and is currently ranked as 3rd most popular language in the world.

1.2 Java Swing

Java Swing is a lightweight Java graphical user interface (GUI) widget toolkit that includes a rich set of widgets. It is part of the Java Foundation Classes (JFC) and includes several packages for developing rich desktop applications in Java. Swing includes built-in controls such as trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, and text areas to display HTTP or rich text format (RTF). Swing components are written entirely in Java and thus are platform-independent. (Technopedia, 2011)

1.3 About this Project :

The objective of this Coursework was to make a Graphical User Interface (GUI) which would be an extension of the previous course work and will help to facilitate the management of teacher details. With the help of GUI user can interact with the system and input teacher details and perform various operations like adding both lecturers and tutors, grading assignments, setting tutor salaries, removing tutors, displaying relevant information, and clearing input fields. This Coursework is designed for building a user-friendly interface that facilitates effective data entry, manipulation, and retrieval, enhancing the overall usability and effectiveness of the teacher management system while following affective Java programming principles and (OOP) concept.

1.4 Tools Used in the Project :

1) BlueJ :

BlueJ is a Java integrated development environment designed for college and university students. It was developed by the University of Kent and Deakin University to teach object orientation in a Java development environment. BlueJ provides an easy-to-use teaching environment that helps first year students learn the Java programming language and then helps transition them to a world IDE (NetBeans). The BlueJ integrated development environment is highly interactive and encourages experimentation and exploration. (Java, 2024)

BlueJ was used in the project mainly for writing and compiling the codes because of its user-friendly interface and also for its great debugging feature which really helped in completion of the project.

2) Draw.io

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 Hope , 2020)

I used draw.io to make a Class Diagram for our project. With its many shapes and symbols, I was able to create a visually appealing representation how the classes are related and what are the elements in each one.

3) Microsoft Word

Microsoft word is a word processor software developed by Microsoft in 1983. It is the most commonly used word processor software. It is used to create professional quality documents, letters, reports, resumes, etc and also allows you to edit or modify your new or existing document. The file saved in Ms Word has .docx extension. (GeeksforGeeks, 2021)

2. Class Diagram

Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes in a system. UML is a standardized modeling language that helps in designing and documenting software systems. They are an integral part of the software development process, helping in both the design and documentation phases. (GeeksforGeeks, 2024)

2.1 Class Diagram for Teacher Class :

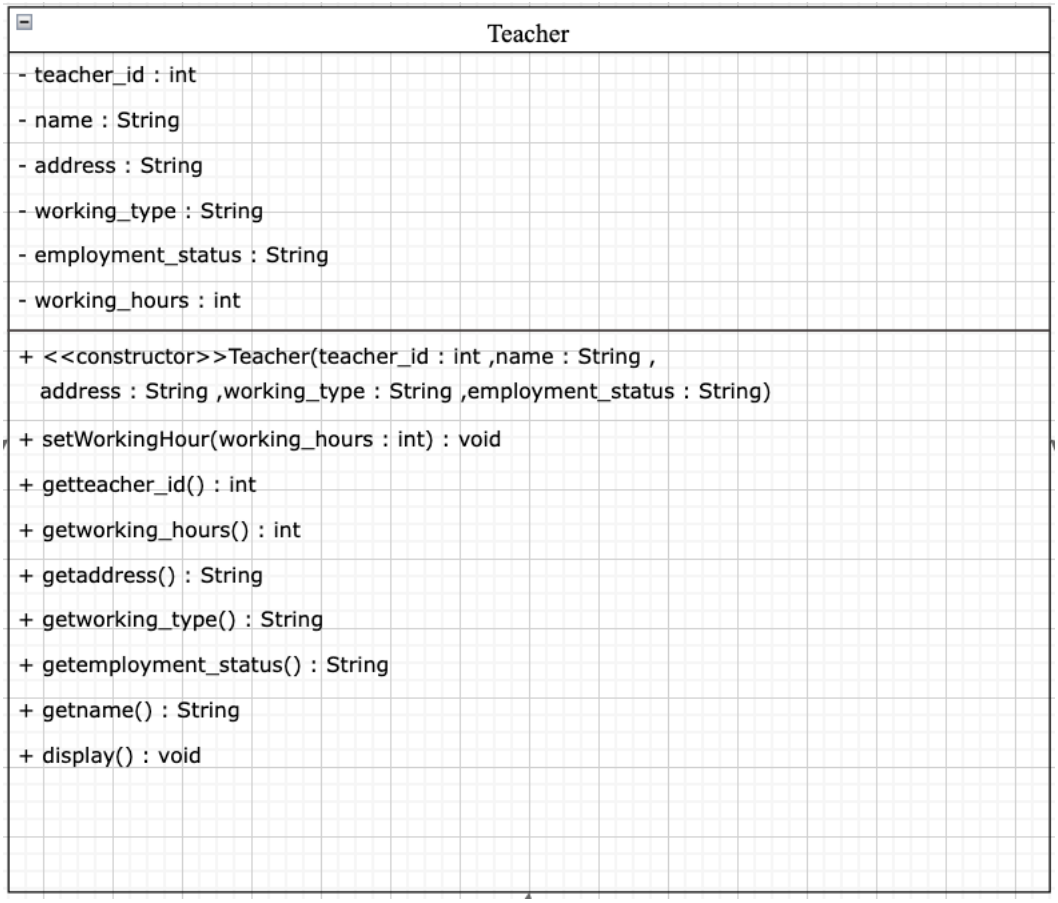


Figure 1: Class Diagram for Teacher Class

2.2 Class Diagram for Lecturer Class :

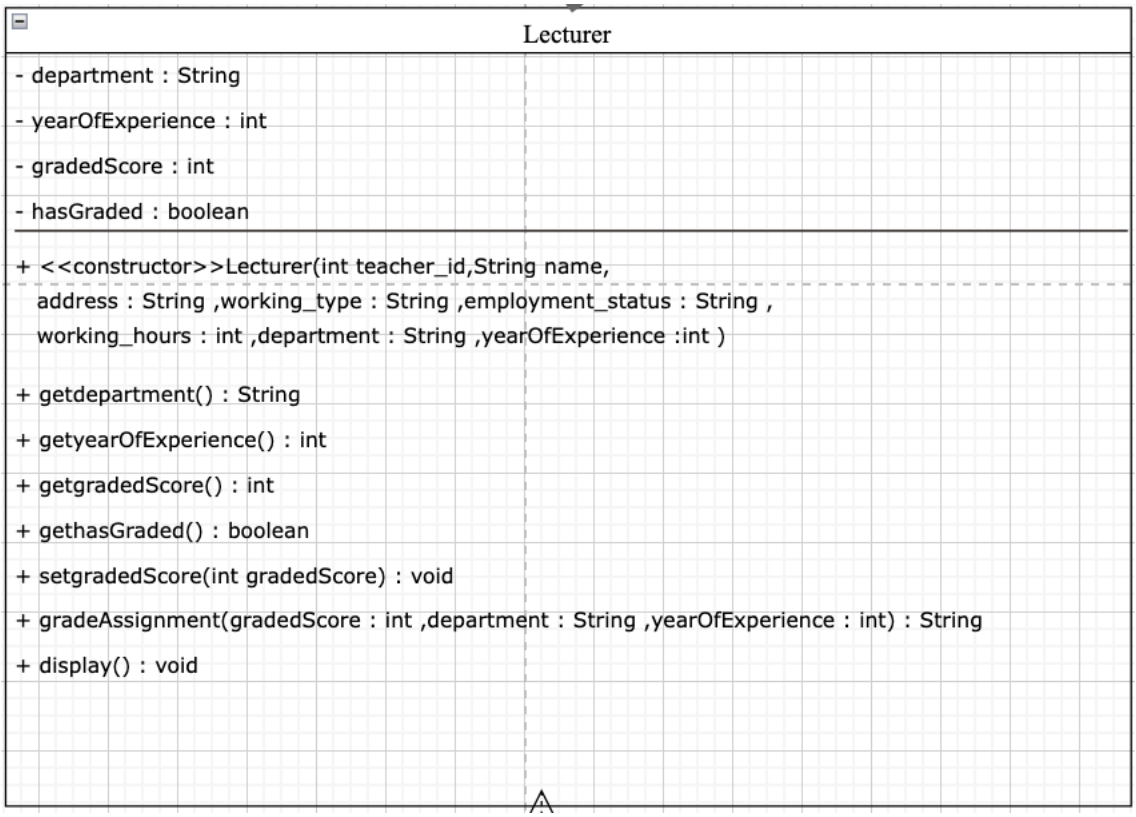


Figure 2 : Class Diagram for Lecturer Class

2.3 Class Diagram for Tutor Class :

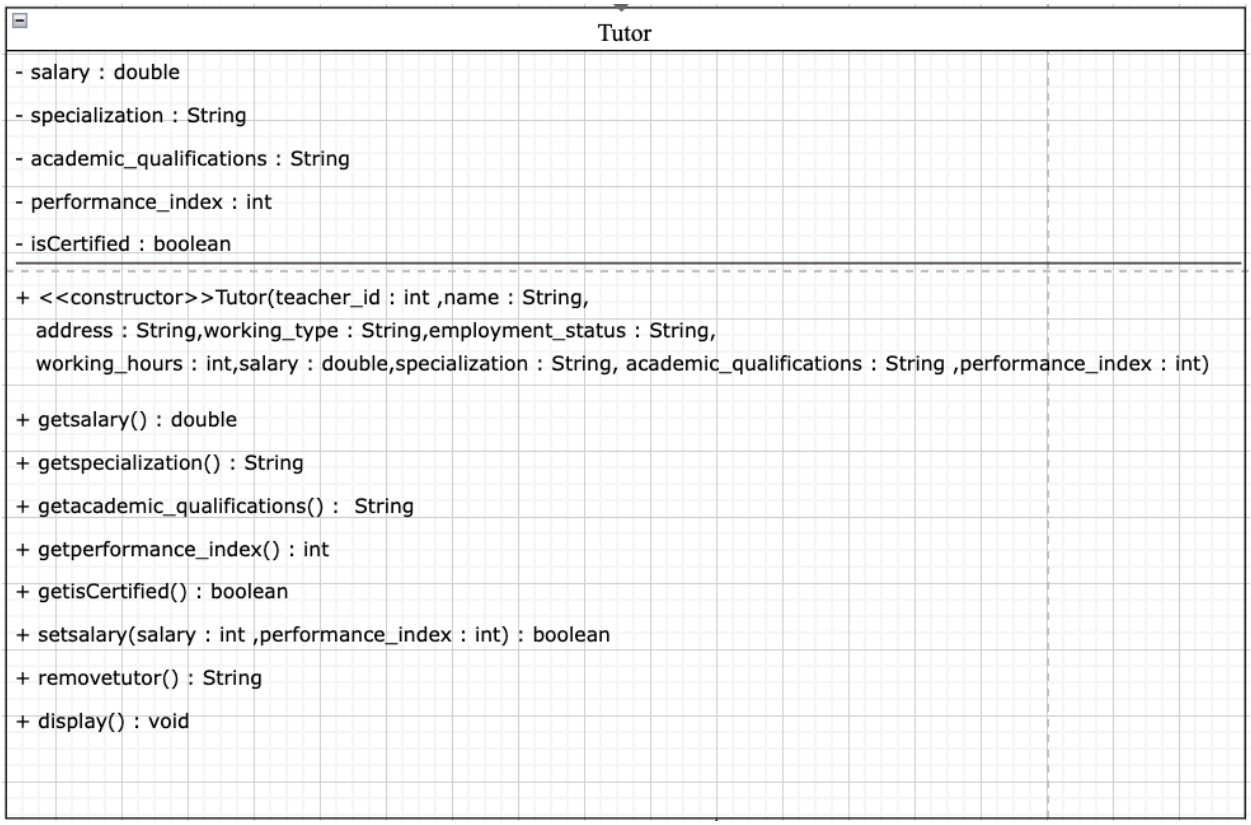


Figure 3 : Class Diagram for Tutor Class

2.4 Class Diagram for TeacherGUI Class:

Figure 4 : Class Diagram for TeacherGUI Class

2.5 Combined Class Diagram :

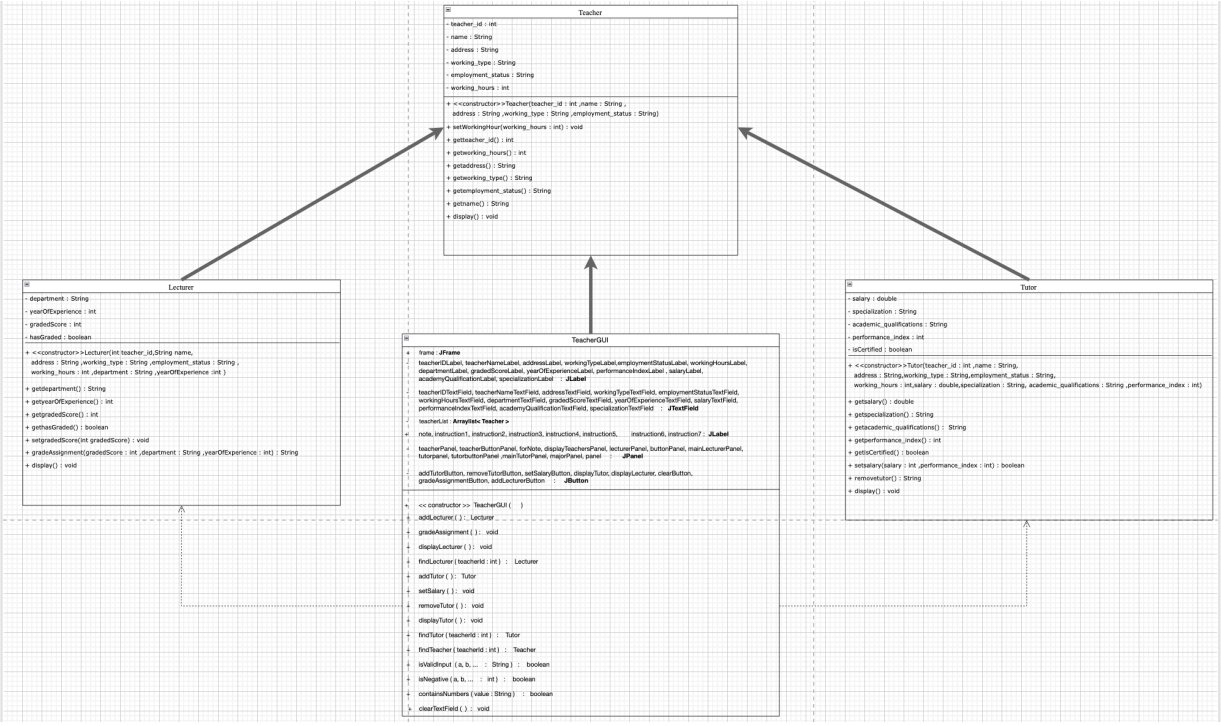


Figure 5 : Combined Detailed Class Diagram

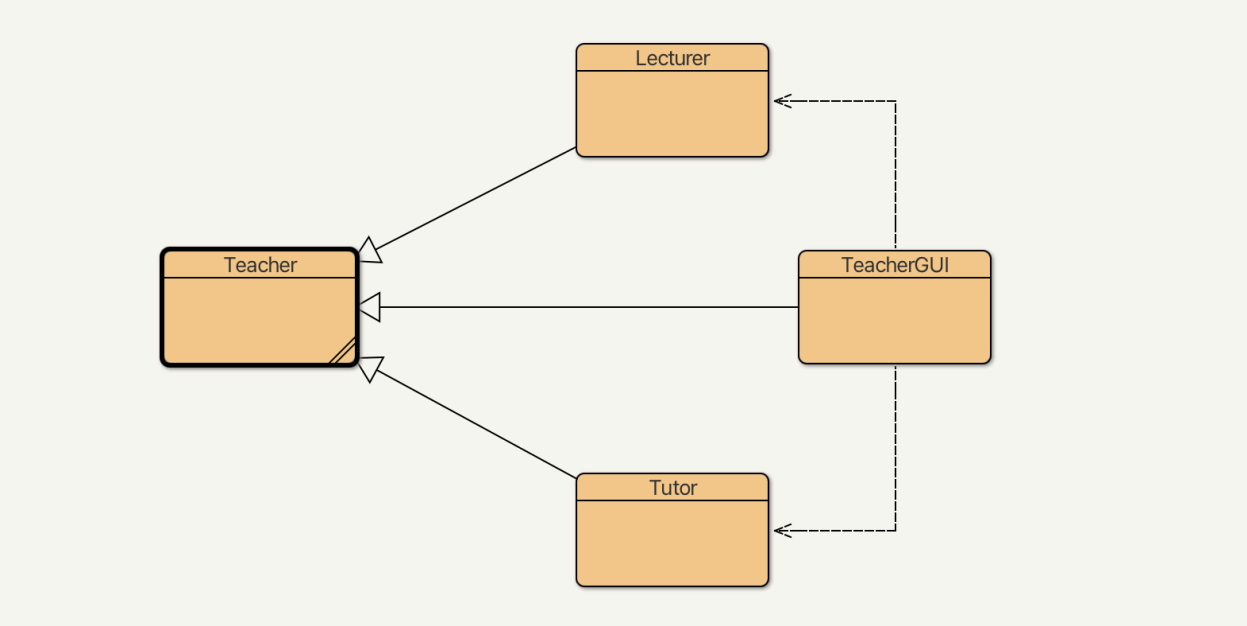


Figure 6 : Combined Class Diagram in BlueJ

3. Pseudocode

A Pseudocode is defined as a step-by-step description of an algorithm. Pseudocode does not use any programming language in its representation instead it uses the simple English language text as it is intended for human understanding rather than machine reading. Pseudocode is the intermediate state between an idea and its implementation(code) in a high-level language. (GeeksforGeeks, 2023)

3.1 Pseudocode for TeacherGUI Class :

```
IMPORT javax.swing.*
IMPORT java.awt.*
IMPORT java.awt.event.ActionEvent
IMPORT java.awt.event.ActionListener
IMPORT java.util.ArrayList

CREATE CLASS MyFrame EXTENDING JFrame

DECLARE ArrayList teachers

DECLARE JLabel teacherIDLabel, teacherNameLabel, addressLabel,
workingTypeLabel,
    employmentStatusLabel, workingHoursLabel, departmentLabel,
gradedScoreLabel,
    yearOfExperienceLabel, salaryLabel, performanceIndexLabel,
    academyQualificationLabel, specializationLabel

DECLARE JTextField teacherIDTextField, teacherNameTextField, addressTextField,
    workingTypeTextField, employmentStatusTextField, workingHoursTextField,
    departmentTextField, gradedScoreTextField, yearOfExperienceTextField,
    salaryTextField, performanceIndexTextField, academyQualificationTextField,
```

```
    specializationTextField


CREATE METHOD MyFrame()
   SET Title to "Teacher Database Portal"
   SET Default Close Operation to EXIT_ON_CLOSE
   SET Size to 1600x900
   SET Location to (130, 20)
   SET Layout of JFrame to GridLayout(3,4)
   INITIALIZE teachers as new ArrayList
   INITIALIZE GridBagConstraints
   CREATE teacherPanel JPanel
   SET Layout of teacherPanel to GridLayout(3, 2, 50, 5)
   SET Border of teacherPanel to createEmptyBorder(20, 20, 10, 20)



INITIALIZE TeacherIDLabel with text "Teacher ID"
INITIALIZE TeacherNameLabel with text "Teacher Name"
INITIALIZE AddressLabel with text "Address"
INITIALIZE WorkingTypeLabel with text "Working Type"
INITIALIZE EmploymentStatusLabel with text "Employment Status"
INITIALIZE WorkingHoursLabel with text "Working Hours"

SET Font of TeacherIDLabel to "Verdana" with fontsize 15
SET Font of TeacherNameLabel to "Verdana" with fontsize 15
SET Font of AddressLabel to "Verdana" with fontsize 15
SET Font of WorkingTypeLabel to "Verdana" with fontsize 15
SET Font of EmploymentStatusLabel to "Verdana" with fontsize 15
SET Font of WorkingHoursLabel to "Verdana" with fontsize 15



INITIALIZE TeacherIDJLabel
```

```
INITIALIZE teacherIDTextField JTextField with width 10
INITIALIZE teacherNameTextField JTextField with width 20
INITIALIZE addressTextField JTextField with width 15
INITIALIZE workingTypeTextField JTextField with width 10
INITIALIZE employmentStatusTextField JTextField with width 10
INITIALIZE workingHoursTextField JTextField with width 10



ADD teacherIDLabel to teacherPanel
ADD teacherIDTextField to teacherPanel
ADD teacherNameLabel to teacherPanel
ADD teacherNameTextField to teacherPanel
ADD addressLabel to teacherPanel
ADD addressTextField to teacherPanel
ADD workingTypeLabel to teacherPanel
ADD workingTypeTextField to teacherPanel
ADD employmentStatusLabel to teacherPanel
ADD employmentStatusTextField to teacherPanel
ADD workingHoursLabel to teacherPanel
ADD workingHoursTextField to teacherPanel


ADD teacherPanel to Frame at (0, 0) with gridwidth 4




CREATE JPanel teacherButtonPanel WITH BorderLayout()

// Buttons For Displaying lecturer and tutor details
```

```
CREATE JButton displayTutor WITH TEXT "Display Tutor Details"
CREATE JButton displayLecturer WITH TEXT "Display Lecturer Details"

// clear Button
CREATE JButton clearButton WITH TEXT "Clear All Input Fields"

// Changing Fonts And Backgrounds of buttons
SET Font of clearButton to "Consolas", Font.BOLD, 16
SET Foreground of clearButton to Color(60, 90, 111)
SET Background of clearButton to Color(199, 188, 161)
SET Font of displayLecturer to "Trebuchet MS", Font.BOLD, 15
SET Background of displayLecturer to Color(232, 246, 239)
SET Font of displayTutor to "Trebuchet MS", Font.BOLD, 15
SET Foreground of displayTutor to Color.BLACK
SET Background of displayTutor to Color(232, 246, 239)

// Will Contain different Instruction in the form of JLabels
CREATE JPanel forNote WITH GridLayout(10, 1)

// Instructions to be followed and will be added to forNote panel
CREATE JLabel note WITH TEXT " Instructions : "
CREATE JLabel instruction1 WITH TEXT "     1)   Please fill all the required input
fields with valid values only ."
CREATE JLabel instruction2 WITH TEXT "     2)   To add Lecturer provide with
Teacher Id, Teacher Name, Address, Working Type, Employment Status, Working
Hour, Year of Experience and Department Name.  "
CREATE JLabel instruction3 WITH TEXT "     3)   To add Tutor provide with Teacher
Id, Teacher Name, Address, Working Type, Employment Status, Working Hour,
Salary, Performance Index, "
CREATE JLabel subinstruction3 WITH TEXT "                Academic Qualification and
Specialization accurately."
```

```
CREATE JLabel instruction4 WITH TEXT "     4)   To assign grades provide with
Teacher Id, Graded Score, Year of Experience ( YOE ) and Department in their
respective input fields ."
CREATE JLabel instruction5 WITH TEXT "     5)   To set salary provide with correct
Teacher ID, Salary and Performance Index in their respective input fields accurately ."
CREATE JLabel instruction6 WITH TEXT "     6)   To remove Tutor provide with
correct Tutor ID in Teacher ID Field."
CREATE JLabel instruction7 WITH TEXT "     7)   To display Lecturer's or Tutor's
details kindly check console window ( Terminal ) . "

// Setting Font for the Instruction Portion
SET Font of note to "Consolas", Font.BOLD, 15
SET Font of instruction1 to "Verdana", Font.PLAIN, 13
SET Font of instruction2 to "Verdana", Font.PLAIN, 13
SET Font of instruction3 to "Verdana", Font.PLAIN, 13
SET Font of subinstruction3 to "Verdana", Font.PLAIN, 13
SET Font of instruction4 to "Verdana", Font.PLAIN, 13
SET Font of instruction5 to "Verdana", Font.PLAIN, 13
SET Font of instruction6 to "Verdana", Font.PLAIN, 13
SET Font of instruction7 to "Verdana", Font.PLAIN, 13

// adding instruction elements to instruction panel
ADD note TO forNote
ADD instruction1 TO forNote
ADD instruction2 TO forNote
ADD instruction3 TO forNote
ADD subinstruction3 TO forNote
ADD instruction4 TO forNote
ADD instruction5 TO forNote
ADD instruction6 TO forNote
ADD instruction7 TO forNote
```

```
// will includes 2 button for displayig tutor and lecturer
CREATE JPanel displayTeachersPanel WITH GridLayout(1, 2)

// Adding buttons to the panel
ADD clearButton TO teacherButtonPanel AT BorderLayout.NORTH
ADD displayLecturer TO displayTeachersPanel
ADD displayTutor TO displayTeachersPanel

// Adding instructions onto the panel
ADD displayTeachersPanel TO teacherButtonPanel AT BorderLayout.SOUTH
ADD forNote TO teacherButtonPanel AT BorderLayout.CENTER

// Adding the panel to the frame    ( 2nd Row )
ADD teacherButtonPanel TO frame



// For Lecturer Panel

// Initializing JLabels
CREATE JLabel departmentLabel WITH TEXT "Department    :"
CREATE JLabel gradedScoreLabel WITH TEXT "Graded Score   :"
CREATE JLabel yearOfExperienceLabel WITH TEXT "Year Of Experience   :"

SET Font of departmentLabel to "Verdana", Font.PLAIN, 15
SET Font of gradedScoreLabel to "Verdana", Font.PLAIN, 15
SET Font of yearOfExperienceLabel to "Verdana", Font.PLAIN, 15

// Initializing JTextFields
CREATE JTextField departmentTextField WITH COLUMN SIZE 15
CREATE JTextField gradedScoreTextField WITH COLUMN SIZE 10
```

```
CREATE JTextField yearOfExperienceTextField WITH COLUMN SIZE 10

// Buttons for Lecturer functions
CREATE JButton addLecturerButton WITH TEXT "Add Lecturer"
CREATE JButton gradeAssignmentButton WITH TEXT "Grade Assignment"

SET Font of addLecturerButton to "Trebuchet MS", Font.BOLD, 15
SET Background of addLecturerButton to Color(176, 197, 164)
SET Font of gradeAssignmentButton to "Trebuchet MS", Font.BOLD, 15
SET Background of gradeAssignmentButton to Color(255, 251, 218)

// Creating lecturerPanel to group different elements of Lecturer
CREATE JPanel lecturerPanel WITH GridLayout(3, 2, 50, 5)
ADD departmentLabel TO lecturerPanel
ADD departmentTextField TO lecturerPanel
ADD yearOfExperienceLabel TO lecturerPanel
ADD yearOfExperienceTextField TO lecturerPanel
ADD gradedScoreLabel TO lecturerPanel
ADD gradedScoreTextField TO lecturerPanel

// Gap around all elements
SET Border of lecturerPanel to BorderFactory.createEmptyBorder(20, 20, 10, 20)

// Creating panel to add add lecturer and grade assignment button
CREATE JPanel buttonPanel WITH GridLayout(1, 3, 5, 5)
ADD addLecturerButton TO buttonPanel
ADD gradeAssignmentButton TO buttonPanel

// Main panel to group button and text field of lecturer
CREATE JPanel mainLecturerPanel
SET Layout of mainLecturerPanel to BorderLayout()
```

```
ADD lecturerPanel TO mainLecturerPanel AT BorderLayout.CENTER
ADD buttonPanel TO mainLecturerPanel AT BorderLayout.SOUTH

// Adding Tutor Panel

// Initializing JLabels
CREATE JLabel salaryLabel WITH TEXT "Salary   :"
CREATE JLabel performanceIndexLabel WITH TEXT "Performance Index   :"
CREATE JLabel academyQualificationLabel WITH TEXT "Academy Qualification   :"
CREATE JLabel specializationLabel WITH TEXT "Specialization   :"

SET Font of salaryLabel to "Verdana", Font.PLAIN, 15
SET Font of performanceIndexLabel to "Verdana", Font.PLAIN, 15
SET Font of academyQualificationLabel to "Verdana", Font.PLAIN, 15
SET Font of specializationLabel to "Verdana", Font.PLAIN, 15

// Initializing JTextFields
CREATE JTextField salaryTextField WITH COLUMN SIZE 10
CREATE JTextField performanceIndexTextField WITH COLUMN SIZE 10
CREATE JTextField academyQualificationTextField WITH COLUMN SIZE 15
CREATE JTextField specializationTextField WITH COLUMN SIZE 15

// Tutor Functionality Buttons
CREATE JButton addTutorButton WITH TEXT "Add Tutor"
CREATE JButton removeTutorButton WITH TEXT "Remove Tutor"
CREATE JButton setSalaryButton WITH TEXT "Set Salary"

// Setting fonts and background of the buttons
SET Font of addTutorButton to "Trebuchet MS", Font.BOLD, 15
SET Background of addTutorButton to Color(176, 197, 164)
SET Font of setSalaryButton to "Trebuchet MS", Font.BOLD, 15
```

```
SET Background of setSalaryButton to Color(170, 215, 217)
SET Font of removeTutorButton to "Trebuchet MS", Font.BOLD, 15
SET Background of removeTutorButton to Color(255, 128, 128)

// Tutor panel for grouping the buttons, text fields, and labels
CREATE JPanel tutorPanel WITH GridLayout(4, 2, 50, 5)

// Gap around all elements
SET Border of tutorPanel to BorderFactory.createEmptyBorder(10, 20, 10, 20)

// Adding elements to tutor panel
ADD salaryLabel TO tutorPanel
ADD salaryTextField TO tutorPanel
ADD performanceIndexLabel TO tutorPanel
ADD performanceIndexTextField TO tutorPanel
ADD academyQualificationLabel TO tutorPanel
ADD academyQualificationTextField TO tutorPanel
ADD specializationLabel TO tutorPanel
ADD specializationTextField TO tutorPanel

// Button panel with GridLayout
CREATE JPanel tutorButtonPanel WITH GridLayout(1, 3, 5, 5)
ADD addTutorButton TO tutorButtonPanel
ADD removeTutorButton TO tutorButtonPanel
ADD setSalaryButton TO tutorButtonPanel

// Main Tutor Panel consists of buttons and text fields
CREATE JPanel mainTutorPanel
SET Layout of mainTutorPanel to BorderLayout()
ADD tutorPanel TO mainTutorPanel AT BorderLayout.CENTER
ADD tutorButtonPanel TO mainTutorPanel AT BorderLayout.SOUTH
```

```
// Comprises of both lecturer and tutor
CREATE JPanel majorPanel WITH GridLayout(1, 2, 50, 20)
ADD mainLecturerPanel TO majorPanel
ADD mainTutorPanel TO majorPanel

// Adding majorPanel to the frame (3rd Row)
SET Grid Position of majorPanel to (0, 2)
SET Grid Width of majorPanel to 4

ADD majorPanel TO frame

PACK frame    // will help consume extra spacing on the frame

// Changing the default visibility of frame & making frame visible
SET Visibility of frame to true

// For clear button
ADD ActionListener to clearButton
   @Override
   CREATE METHOD actionPerformed(ActionEvent e)
      DISPLAY Confirmation dialog with message "Are you sure you want to clear all
the fields?"
      IF user selects "Yes" THEN
         CALL clearTextField() method
      END IF
   END METHOD

// For display lecturer button
ADD ActionListener to displayLecturer
   @Override
```

```
    CREATE METHOD actionPerformed(ActionEvent e)
        CALL displayLecturer() method
    END METHOD

// For display tutor button
ADD ActionListener to displayTutor
    @Override
    CREATE METHOD actionPerformed(ActionEvent e)
        CALL displayTutor() method
    END METHOD

// For add lecturer button
ADD ActionListener to addLecturerButton
    @Override
    CREATE METHOD actionPerformed(ActionEvent e)
        CREATE Teacher teacher1 = CALL addLecturer() method
        IF teacher1 is not null THEN
            ADD teacher1 to teachers list
        END IF
    END METHOD

// For add tutor button
ADD ActionListener to addTutorButton
    @Override
    CREATE METHOD actionPerformed(ActionEvent e)
        CREATE Teacher teacher2 = CALL addTutor() method
        IF teacher2 is not null THEN
            ADD teacher2 to teachers list
        END IF
    END METHOD
```

```
// For grade assignment button
ADD ActionListener to gradeAssignmentButton
    @Override
    CREATE METHOD actionPerformed(ActionEvent e)
        CALL gradeAssignment() method
    END METHOD

// For set salary button
ADD ActionListener to setSalaryButton
    @Override
    CREATE METHOD actionPerformed(ActionEvent e)
        CALL setSalary() method
    END METHOD

// For remove tutor button
ADD ActionListener to removeTutorButton
    @Override
    CREATE METHOD actionPerformed(ActionEvent e)
        CALL removeTutor() method
    END METHOD



// For addLecturer method
CREATE METHOD addLecturer()
    // Retrieving input values from text fields and trim leading/trailing whitespace
    INITIALIZE String teacherID = CALL getText().trim() on teacherIDTextField
    INITIALIZE String teacherName = CALL getText().trim() on teacherNameTextField
    INITIALIZE String address = CALL getText().trim() on addressTextField
    INITIALIZE String workingType = CALL getText() on workingTypeTextField
    INITIALIZE String employmentStatus = CALL getText().trim() on
employmentStatusTextField
```

```
    INITIALIZE String workingHoursStr = CALL getText().trim() on
workingHoursTextField
    INITIALIZE String department = CALL getText().trim() on departmentTextField
    INITIALIZE String yearOfExperienceStr = CALL getText().trim() on
yearOfExperienceTextField

    TRY
        IF NOT CALL isValidInput() with parameters teacherID, teacherName, address,
workingType, employmentStatus, workingHoursStr, department, yearOfExperienceStr
THEN
            DISPLAY JOptionPane message dialog with error message: "Oops! It looks
like some fields are empty or contain invalid data.\nBefore proceeding, please ensure
all required fields are filled properly.\n\nNote: See instructions for help."
            RETURN null
        END IF

        // Check if teacher with the same ID already exists
        IF CALL findTeacher() with parameter Integer.parseInt(teacherID) is null THEN
            INITIALIZE int teacherId = CALL parseInt(teacherID)
            INITIALIZE int workingHours = CALL parseInt(workingHoursStr)
            INITIALIZE int yearOfExperience = CALL parseInt(yearOfExperienceStr)

            // Check for negative values in input fields
            IF CALL isNegative() with parameters teacherId, workingHours,
yearOfExperience THEN
                DISPLAY JOptionPane message dialog with error message: "Invalid
input!!\nInput Field cannot have negative values."
                RETURN null
            END IF

            // Check for numeric values in non-numeric fields
```

```
        IF containsNumbers(department) OR containsNumbers(employmentStatus)
OR containsNumbers(workingType)
                OR containsNumbers(address) OR containsNumbers(teacherName)
THEN
            DISPLAY JOptionPane message dialog with error message: "Invalid
input!\nNumeric values are not allowed in name, address, working type, and
employment status fields.\nPlease correct your input.\n\nNote: See instructions for
help."
            RETURN null
        END IF

        IF yearOfExperience > 100 THEN
            DISPLAY JOptionPane message dialog with error message: "Year Of
Experience must be valid."
            RETURN null
        END IF

        IF workingHours > 168 THEN
            DISPLAY JOptionPane message dialog with error message: "Working
Hours must be in the range from 1 to 168."
            RETURN null
        END IF

        // Creating new Lecturer object
        INITIALIZE Lecturer lecturer = CALL Lecturer constructor with parameters
teacherId, teacherName, address, workingType, employmentStatus, workingHours,
department, yearOfExperience
        // Displaying success message, clear text fields, and return newly created
Lecturer object
        DISPLAY JOptionPane message dialog with success message: "Lecturer
added successfully."
```

```
            CALL clearTextField()
            RETURN lecturer
        ELSE
            // Display error message if teacher with the same ID already exists
            DISPLAY JOptionPane message dialog with error message: "Teacher with ID: " + Integer.parseInt(teacherID) + " already exists."
        END IF
    CATCH NumberFormatException e
        // Display error message for invalid numerical input
        DISPLAY JOptionPane message dialog with error message: "Oops! It seems you've entered invalid input.\nPlease provide valid numerical values for\n1) Teacher ID\n2) Working Hours\n3) Year of Experience"
    CATCH Exception y
        // Displaying a standard error message for other exceptions
        DISPLAY JOptionPane message dialog with error message: "Invalid input! Tutor was not added."
    END TRY
END METHOD




// For addTutor method
CREATE METHOD addTutor()
    // Retrieve input values from text fields
    INITIALIZE String teacherID = CALL getText().trim() on teacherIDTextField
    INITIALIZE String teacherName = CALL getText().trim() on teacherNameTextField
    INITIALIZE String address = CALL getText().trim() on addressTextField
    INITIALIZE String workingType = CALL getText() on workingTypeTextField
```

```
    INITIALIZE String employmentStatus = CALL getText().trim() on
employmentStatusTextField
    INITIALIZE String workingHoursStr = CALL getText().trim() on
workingHoursTextField
    INITIALIZE String salaryStr = CALL getText().trim() on salaryTextField
    INITIALIZE String performanceIndexStr = CALL getText().trim() on
performanceIndexTextField
    INITIALIZE String academyQualification = CALL getText().trim() on
academyQualificationTextField
    INITIALIZE String specialization = CALL getText().trim() on specializationTextField

    // Check if all input fields are valid i.e they are not empty strings
    IF NOT CALL isValidInput() with parameters teacherID, teacherName, address,
workingType, employmentStatus, workingHoursStr, salaryStr, performanceIndexStr,
academyQualification, specialization THEN
        // Displaying error message for invalid or empty fields
        DISPLAY JOptionPane message dialog with error message: "Oops! It looks like
some fields are empty or contain invalid data. Please fill in all required fields with
accurate information.\n\nNote: See instructions for help."
        RETURN null
    END IF

    TRY
        // Checking if teacher with the same ID already exists
        IF CALL findTeacher() with parameter Integer.parseInt(teacherID) is null THEN
            // Parsing String input values to integer data types
            INITIALIZE int teacherId = CALL parseInt(teacherID)
            INITIALIZE int workingHours = CALL parseInt(workingHoursStr)
            INITIALIZE int salary = CALL parseInt(salaryStr)
            INITIALIZE int performanceIndex = CALL parseInt(performanceIndexStr)
```

```
        // Checking if any numeric field is negative
        IF CALL isNegative() with parameters teacherId, workingHours, salary,
performanceIndex THEN
            DISPLAY JOptionPane message dialog with error message: "Invalid input!
Input Field cannot have Negative Values"
            RETURN null
        END IF

        // Checking if numeric values are present in non-numeric fields
        IF containsNumbers(specialization) OR
containsNumbers(academyQualification)
            OR containsNumbers(employmentStatus) OR
containsNumbers(workingType)
            OR containsNumbers(address) OR containsNumbers(teacherName)
THEN
            DISPLAY JOptionPane message dialog with error message: "Invalid input!
Numeric values are not allowed in name, address, working type, employment status,
specialization, and academic qualification fields. Please correct your input.\n\nNote:
See instructions for help."
            RETURN null
        END IF

        IF performanceIndex > 10 THEN
            DISPLAY JOptionPane message dialog with error message: "Performance
Index value ranges from 1 to 10."
            RETURN null
        END IF

        IF workingHours > 168 THEN
            DISPLAY JOptionPane message dialog with error message: "Working
Hours value ranges from 1 to 168."
```

```
            RETURN null
        END IF

        // Creating new Tutor object
        INITIALIZE Tutor tutor = CALL Tutor constructor with parameters teacherId, teacherName, address, workingType, employmentStatus, workingHours, salary, specialization, academyQualification, performanceIndex

        // Displaying success message
        DISPLAY JOptionPane message dialog with success message: "Tutor added successfully!"
        CALL clearTextField() // Clearing TextFields after adding tutors
        RETURN tutor
    ELSE
        // Displaying error message if teacher with the same ID already exists
        DISPLAY JOptionPane message dialog with error message: "Teacher with ID: " + Integer.parseInt(teacherID) + " already exists."
    END IF
  CATCH NumberFormatException e
    // Displaying error message for invalid numerical input
    DISPLAY JOptionPane message dialog with error message: "Oops! It seems you've entered invalid input. Please provide valid numerical values for:\n1) Teacher ID\n2) Working Hours\n3) Salary\n4) Performance Index"
  CATCH Exception y
    // Displaying a standard error message for other exceptions
    DISPLAY JOptionPane message dialog with error message: "Invalid input! Tutor was not added."
  END TRY
  RETURN null
END METHOD
```

```
// For gradeAssignment method
CREATE METHOD gradeAssignment()
   // Retrieve input values from text fields and trim leading/trailing whitespace
   INITIALIZE String teacherID = CALL getText().trim() on teacherIDTextField
   INITIALIZE String gradeStr = CALL getText().trim() on gradedScoreTextField
   INITIALIZE String department = CALL getText().trim() on departmentTextField
   INITIALIZE String yearStr = CALL getText().trim() on yearOfExperienceTextField

   TRY
      // Input validation
      IF NOT CALL isValidInput() with parameters teacherID, gradeStr, yearStr,
department THEN
         DISPLAY JOptionPane message dialog with error message: "Invalid input!!
Please check your entries for Teacher ID, Graded Score, Department, and Year of
Experience.\n\nNote: See instructions for help."
         RETURN
      END IF

      // Parsing String input values to integer data types
      INITIALIZE int teacherId = CALL parseInt(teacherID)
      INITIALIZE int gradedScore = CALL parseInt(gradeStr)
      INITIALIZE int yearsOfExperience = CALL parseInt(yearStr)

      // Checking for negative values
      IF CALL isNegative() with parameters teacherId, gradedScore,
yearsOfExperience THEN
         DISPLAY JOptionPane message dialog with error message: "Invalid input!!
Input Field cannot have negative values."
```

```
            RETURN
        END IF

        // Check for numeric values in the department field
        IF containsNumbers(department) THEN
            DISPLAY JOptionPane message dialog with error message: "Invalid input!
Numeric values are not allowed in the department field. Please correct your input."
            RETURN
        END IF

        IF gradedScore > 100 THEN
            DISPLAY JOptionPane message dialog with error message: "Invalid input!!
Graded Score must be in the range from 0 to 100."
            RETURN
        END IF

        // Checking if lecturer exists
        INITIALIZE Lecturer lecturer = CALL findLecturer() with parameter teacherId
        IF lecturer is not null THEN
            // Calling gradeAssignment method from Lecturer class
            INITIALIZE String result = CALL lecturer.gradeAssignment() with parameters
gradedScore, department, yearsOfExperience
            IF result is not equal to "Teacher Not Eligible For Grade Assignment" THEN
                DISPLAY JOptionPane message dialog with success message: "( " + result
+ " ) Assigned and Grade assignment successful From lecturer " + teacherID +
"\nTeacher ID : " + teacherId + "\nGraded Score : " + gradedScore + "\nDepartment : "
+ department + "\nYears of Experience : " + yearsOfExperience
                CALL clearTextField() // Clearing text fields after gradingAssignment
            ELSE
                DISPLAY JOptionPane message dialog with error message: result
            END IF
```

```
        ELSE
            DISPLAY JOptionPane message dialog with error message: "Lecturer with ID
" + teacherID + " not found."
        END IF
    CATCH NumberFormatException g
        // Display error message for invalid numerical input
        DISPLAY JOptionPane message dialog with error message: "Oops! It seems
you've entered invalid input. Please provide valid numerical values for :\n1) Teacher
ID\n2) Graded Score\n3) Year Of Experience"
    CATCH Exception z
        // Displaying standard error message for other exceptions
        DISPLAY JOptionPane message dialog with error message: "Invalid input!
Grade cannot be assigned."
    END TRY
END METHOD




// For removeTutor method
CREATE METHOD removeTutor()
    // Retrieve teacher ID from text field and trim leading/trailing whitespace
    INITIALIZE String teacherID = CALL getText().trim() on teacherIDTextField

    TRY
        // Check if teacher ID input is empty string or not
        IF NOT CALL isValidInput() with parameter teacherID THEN
            DISPLAY JOptionPane message dialog with error message: "\nInvalid
input!!\nPlease check your entries for Teacher ID\n\nNote: See instructions for help."
            RETURN
        END IF
```

```
        // Parsing teacher ID to integer
        INITIALIZE int teacherId = CALL parseInt(teacherID)

        // Checking for negative values in teacher ID
        IF CALL isNegative() with parameter teacherId THEN
            DISPLAY JOptionPane message dialog with error message: "Invalid input!!
Input Field cannot have negative values."
            RETURN
        END IF

        // Asking for confirmation before removing the tutor
        INITIALIZE int option = DISPLAY JOptionPane showConfirmDialog() with
parameters null, "Are you sure you want to proceed?", "Confirmation",
JOptionPane.YES_NO_OPTION
        IF option is equal to 0 THEN
            // Finding if Tutor object with given ID already exists or not
            INITIALIZE Tutor tutor = CAST findTutor() with parameter teacherId
            IF tutor is not null THEN
                // If it exists, proceed to remove it
                INITIALIZE String removeornot = CALL tutor.removetutor()
                IF removeornot is not equal to "Tutor is cerified & cannot be removed"
THEN
                    CALL teachers.remove() with parameter tutor // Removing the tutor
object from the arrayList
                    DISPLAY JOptionPane message dialog with success message: "Tutor
with ID " + teacherID + " removed successfully."
                    CALL clearTextField() // Clearing text fields after removing tutor
                ELSE
                    DISPLAY JOptionPane message dialog with error message:
removeornot
```

```
                RETURN
            END IF
        ELSE
            DISPLAY JOptionPane message dialog with error message: "Tutor with ID "
+ teacherID + " not found."
        END IF
    END IF
  CATCH NumberFormatException h
    // Display error message for invalid numerical input
    DISPLAY JOptionPane message dialog with error message: "Oops! It seems
you've entered invalid input. Please provide valid numerical values for : \n1) Teacher
ID\n\nNote: See instructions for help."
  CATCH Exception y
    // Displaying standard error message for other exceptions
    DISPLAY JOptionPane message dialog with error message: "Invalid input! Tutor
was not removed." + y
  END TRY
END METHOD




// For setSalary method
CREATE METHOD setSalary()
  // Retrieve input values from text fields and trim leading/trailing whitespace
  INITIALIZE String teacherID = CALL getText().trim() on teacherIDTextField
  INITIALIZE String salaryStr = CALL getText().trim() on salaryTextField
  INITIALIZE String performanceStr = CALL getText().trim() on
performanceIndexTextField

  TRY
```

```
        // Check if teacher ID, salary, or performance index input is empty string or not
        IF NOT CALL isValidInput() with parameters teacherID, salaryStr,
performanceStr THEN
            // Display suitable message in information dialog if any of the input fields is
empty string
            DISPLAY JOptionPane message dialog with error message: "\nInvalid input!!
\nPlease check your entries for Teacher ID,\nSalary and Performance Index.\n\nNote:
See instructions for help."
            RETURN
        END IF

        // Parsing input strings to integers
        INITIALIZE int teacherId = CALL parseInt(teacherID)
        INITIALIZE int salary = CALL parseInt(salaryStr)
        INITIALIZE int performanceIndex = CALL parseInt(performanceStr)

        // Checking for negative values
        IF CALL isNegative() with parameters teacherId, salary, performanceIndex
THEN
            // Display suitable message in information dialog if any of the input fields is a
negative value
            DISPLAY JOptionPane message dialog with error message: "Invalid input!!
Input Field cannot have negative values."
            RETURN
        END IF

        // Finding tutor to know if they exist already or not
        INITIALIZE Tutor tutor = CALL findTutor() with parameter teacherId
        IF tutor is not null THEN
            // Calling set salary method and storing the returned boolean value onto a
variable
```

```
        INITIALIZE boolean isIncreased = CALL tutor.setsalary() with parameters
salary, performanceIndex

        // Check if salary was set and show new salary and performance index in a
dialog box
        IF isIncreased is true THEN
            INITIALIZE int appraisal = CAST tutor.getsalary() - salary
            DISPLAY JOptionPane message dialog with success message: "Salary and
performance index updated for Tutor ID : " + teacherID + "\nNew Salary : " + CAST
tutor.getsalary() + "\nUpdated Performance Index : " + performanceIndex + "\nBonus
Amount : " + appraisal
            CALL clearTextField() // Clearing text fields after setting salary
        ELSE
            // If salary was not set then display suitable information dialog
            DISPLAY JOptionPane message dialog with error message: "Requirements
not met for salary increment."
        END IF
      ELSE
        DISPLAY JOptionPane message dialog with error message: "Tutor with ID " +
teacherID + " not found."
      END IF
   CATCH NumberFormatException f
      // Display error message for invalid numerical input
      DISPLAY JOptionPane message dialog with error message: "Oops! It seems
you've entered invalid input. \nPlease provide valid numerical values for :\n1) Teacher
ID \n2) Salary \n3) Performance Index"
   CATCH Exception y
      // Displaying standard error message for other exceptions
      DISPLAY JOptionPane message dialog with error message: "Invalid input!
Salary was not set."
   END TRY
```

```
END METHOD



CREATE METHOD displayLecturer()
   INITIALIZE int count = 0

   // Check if there exists any Lecturer type of object in ArrayList
   FOR EACH Teacher teacher in teachers DO
      IF teacher is an instance of Lecturer THEN
         // If any instance of Lecturer found, increase the count value and break the
loop
         INCREASE count by 1
         BREAK the loop
      END IF
   END FOR

   IF count is less than or equal to 0 THEN
      // If no lecturers found, display a message dialog
      DISPLAY JOptionPane message dialog with message: "There are currently no
Lecturers to display."
      RETURN
   ELSE
      // Create a new frame to display lecturer details
      INITIALIZE JFrame displayLecturerFrame
      SET displayLecturerFrame title to "Lecturer Details"
      SET displayLecturerFrame default close operation to
JFrame.DISPOSE_ON_CLOSE
      SET displayLecturerFrame size to 940x280
      SET displayLecturerFrame location to center of the screen

      INITIALIZE JPanel panel with BorderLayout
```

```
    ADD panel to displayLecturerFrame

    INITIALIZE String array columnNames with column names
    INITIALIZE 2D String array rowData with size
[teachers.size()][columnNames.length]

    INITIALIZE int rowIndex to 0
    // Iterate over the teachers list to populate the table data
    FOR EACH Teacher teacher in teachers DO
      IF teacher is an instance of Lecturer THEN
        INITIALIZE Lecturer lecturer as the casted instance of teacher
        // Populate rowData with lecturer details
        SET rowData[rowIndex][0] to String value of lecturer.getteacher_id()
        SET rowData[rowIndex][1] to lecturer.getname()
        SET rowData[rowIndex][2] to lecturer.getaddress()
        SET rowData[rowIndex][3] to lecturer.getworking_type()
        SET rowData[rowIndex][4] to lecturer.getemployment_status()
        SET rowData[rowIndex][5] to String value of lecturer.getworking_hours()
        SET rowData[rowIndex][6] to String value of lecturer.getdepartment()
        SET rowData[rowIndex][7] to String value of lecturer.getyearOfExperience()
        INCREASE rowIndex by 1
      END IF
    END FOR

    // Create a JTable with the rowData and columnNames
    INITIALIZE JTable table with rowData and columnNames
    SET table auto resize mode to JTable.AUTO_RESIZE_OFF // Disable auto-
resizing of columns

    // Set preferred column widths for each column
    SET preferred width of column 0 to 50 // ID
```

```
        SET preferred width of column 1 to 125 // Tutor Name
        SET preferred width of column 2 to 160 // Address
        SET preferred width of column 3 to 110 // Working Type
        SET preferred width of column 4 to 130 // Employment Status
        SET preferred width of column 5 to 100 // Working Hour
        SET preferred width of column 6 to 140 // Department
        SET preferred width of column 7 to 115 // Year Of Experience

        // Create a JScrollPane to contain the table
        INITIALIZE JScrollPane scrollPane with table
        ADD scrollPane to panel at BorderLayout.CENTER

        // Set the visible property of displayLecturerFrame to true
        SET displayLecturerFrame visible to true
    END IF
END METHOD




CREATE METHOD displayTutor()
    INITIALIZE int count to 0 // starting from zero

    // Iterate over all the ArrayList elements and check if there exists any Tutor type of
object in ArrayList
    FOR EACH Teacher teacher IN teachers DO
        IF teacher is an instance of Tutor THEN
            // If any instance of Tutor found, increase the count value and break the loop
            INCREASE count by 1
            BREAK the loop
        END IF
```

```
    END FOR

    IF count is less than or equal to 0 THEN // Check if there exists at least one object of tutor in the list or not
        // If there are not even 1 object of tutor in the list, display appropriate message and return
        DISPLAY JOptionPane message dialog with message: "There are currently no Tutors to display."
        RETURN
    ELSE
        // Create a new frame to display Tutor details
        INITIALIZE JFrame displayTutorFrame
        SET displayTutorFrame title to "Tutor Details"
        SET displayTutorFrame default close operation to JFrame.DISPOSE_ON_CLOSE
        SET displayTutorFrame size to 1135x280
        SET displayTutorFrame location to center of the screen

        // Creating a panel for the frame and adding it onto the frame
        INITIALIZE JPanel panel with BorderLayout
        ADD panel to displayTutorFrame

        // Creating an array for holding column names
        INITIALIZE String array columnNames with column names

        // Creating a two-dimensional array to hold the data for each teacher in the table
        INITIALIZE 2D String array rowData with size [teachers.size()][columnNames.length]

        INITIALIZE int rowIndex to 0
        // Iterate over the teachers list to populate the table data
```

```
    FOR EACH Teacher teacher IN teachers DO
        IF teacher is an instance of Tutor THEN
            INITIALIZE Tutor tutor as the casted instance of teacher
            // Extract data from each Tutor object and populate the table row by row
            SET rowData[rowIndex][0] to String value of tutor.getteacher_id()
            SET rowData[rowIndex][1] to tutor.getname()
            SET rowData[rowIndex][2] to tutor.getaddress()
            SET rowData[rowIndex][3] to tutor.getworking_type()
            SET rowData[rowIndex][4] to tutor.getemployment_status()
            SET rowData[rowIndex][5] to String value of tutor.getworking_hours()
            SET rowData[rowIndex][6] to String value of (int) tutor.getsalary()
            SET rowData[rowIndex][7] to tutor.getspecialization()
            SET rowData[rowIndex][8] to tutor.getacademic_qualifications()
            SET rowData[rowIndex][9] to String value of tutor.getperformance_index()
            INCREASE rowIndex by 1
        END IF
    END FOR

    // Initialize the JTable variable which will contain all the values of the Tutor
    INITIALIZE JTable table with rowData and columnNames
    SET table auto resize mode to JTable.AUTO_RESIZE_OFF // Disable auto-
resizing of columns

    // Set preferred column widths for each column
    SET preferred width of column 0 to 45 // ID
    SET preferred width of column 1 to 120 // Tutor Name
    SET preferred width of column 2 to 160 // Address
    SET preferred width of column 3 to 110 // Working Type
    SET preferred width of column 4 to 120 // Employment Status
    SET preferred width of column 5 to 100 // Working Hour
    SET preferred width of column 6 to 80 // Salary
```

```
        SET preferred width of column 7 to 130 // Specialization
        SET preferred width of column 8 to 140 // Academic Qualification
        SET preferred width of column 9 to 120 // Performance Index

        // Create a JScrollPane to contain the table
        INITIALIZE JScrollPane scrollPane with table
        ADD scrollPane to panel at BorderLayout.CENTER

        // Set the visible property of displayTutorFrame to true
        SET displayTutorFrame visible to true
    END IF
END METHOD




CREATE METHOD clearTextField()
    // Clear each text field in the frame
    SET teacherIDTextField text to empty string
    SET teacherNameTextField text to empty string
    SET addressTextField text to empty string
    SET workingTypeTextField text to empty string
    SET employmentStatusTextField text to empty string
    SET workingHoursTextField text to empty string
    SET departmentTextField text to empty string
    SET gradedScoreTextField text to empty string
    SET yearOfExperienceTextField text to empty string
    SET salaryTextField text to empty string
    SET performanceIndexTextField text to empty string
    SET academyQualificationTextField text to empty string
    SET specializationTextField text to empty string
```

```
END METHOD



CREATE METHOD isValidInput(String a)
   IF a is empty string THEN
      RETURN false
   ELSE
      RETURN true
   END IF
END METHOD

CREATE METHOD isValidInput(String a, String b, String c)
   IF a or b or c is empty string THEN
      RETURN false
   ELSE
      RETURN true
   END IF
END METHOD

CREATE METHOD isValidInput(String a, String b, String c, String d)
   IF a or b or c or d is empty string THEN
      RETURN false
   ELSE
      RETURN true
   END IF
END METHOD

CREATE METHOD isValidInput(String a, String b, String c, String d, String e, String f,
String g, String h)
   IF any of the parameters is an empty string THEN
```

```
        RETURN false
    ELSE
        RETURN true
    END IF
END METHOD

CREATE METHOD isValidInput(String a, String b, String c, String d, String e, String f,
String g, String h,
            String i, String j)
    IF any of the parameters is an empty string THEN
        RETURN false
    ELSE
        RETURN true
    END IF
END METHOD




CREATE METHOD isNegative(int a)
    IF a < 0 THEN
        RETURN true
    ELSE
        RETURN false
    END IF
END METHOD

CREATE METHOD isNegative(int a, int b, int c)
    IF any of a, b, or c is less than 0 THEN
        RETURN true
    ELSE
```

```
        RETURN false
    END IF
END METHOD

CREATE METHOD isNegative(int a, int b, int c, int d)
    IF any of a, b, c, or d is less than 0 THEN
        RETURN true
    ELSE
        RETURN false
    END IF
END METHOD



CREATE METHOD findTeacher(int teacherId)
    FOR EACH teacher IN teachers
        IF teacher is an instance of Teacher AND teacher's ID is equal to teacherId
THEN
            RETURN teacher
        END IF
    END FOR
    RETURN null
END METHOD

CREATE METHOD findLecturer(int teacherId)
    FOR EACH teacher IN teachers
        IF teacher is an instance of Lecturer AND teacher's ID is equal to teacherId
THEN
            RETURN teacher CASTED TO Lecturer
        END IF
    END FOR
    RETURN null
```

```
END METHOD

CREATE METHOD findTutor(int teacherId)
   FOR EACH teacher IN teachers
      IF teacher is an instance of Tutor AND teacher's ID is equal to teacherId THEN
         RETURN teacher CASTED TO Tutor
      END IF
   END FOR
   RETURN null
END METHOD

CREATE METHOD containsNumbers(String value)
   FOR EACH character IN value
      IF character is a digit THEN
         RETURN true
      END IF
   END FOR
   RETURN false
END METHOD



CREATE CLASS TeacherGUI EXTENDING Teacher
   CREATE METHOD main(String[] args)
      // Creating an object of MyFrame class to display the frame
               INITIALIZE MyFrame frame
               CALL MyFrame() method and ASSIGN the returned value to frame
   END METHOD
END CLASS
```

Table 1 : Pseudocode for TeacherGUI Class

4. Method Description :

A Method Description is a short explanation of what each function & method does and what's their roles are when a button's Action Listeners is triggered . It also explains how one function or method connects with other methods and variables. It's handy for quickly understanding how a method or function is performing in a creation and functioning of a class.

4.1 Method Description for Add Lecturer Button

The Add Lecturer Button triggers the addLecturer( ) method which extracts the values from the text fields and conducts validation checks. This validation includes ensuring that the provided values are not empty, negative, or containing alphabetic characters. In case the validation fails, an appropriate information dialog is displayed showcasing where the error is while filling up the text field letting user to correct their mistakes. if it passes all the validation, it creates a new object of Lecturer with the value obtained in the text field via getText() method. This Lecturer object is then added to the list, which is of type Teacher. The addLecturer( ) method performs validation with the help of various other methods, which are briefly introduced below.

Method Name	Description
findLecturer (int teacherId)	This method iterate through all the teacher object present in the Arraylist and compare every teacher object's teacher id using getter method with the value passed in the parameter and If a match is found, the method returns the corresponding Lecturer object.
isValidInput (String … values)	This method returns "true" if any text field is not empty and "false" otherwise. It does this by comparing all of the values provided in the parameter with an empty String value i.e. (""). This approach uses method overloading to make it simpler and reduce needless method naming.
findTeacher (int teacherId)	This method loop through the arraylist of Teacher and check if any teacher present in the list already have the teacher id provided in the parameter, ensuring that the teacher id unique.
isNegative (int … values)	This method checks if the provided values in the parameter are negative or not. If the values are negative it returns 'true' otherwise 'false'.
containsNumbers (String value)	This method checks if the provided String in the parameter contains number or not. It will return 'true' if the provided String contains number , 'false' otherwise . It helps to assist users in preventing accidental entry a non-alphabetic value in the text field mistakenly where only alphabetic values are allowed.
clearTextField ( )	This method will clear all the input text fields after the Lecturer is added to the list to give user a better user experience.

Table 2 : Method Description for Add Lecturer Button

4.2 Method Description for Add Tutor Button

The "Add Tutor" Button triggers the addTutor( ) method, which retrieves information from the text fields and conducts validation checks. These checks ensure that the provided values are not empty, negative, or containing alphabetic characters. If the validation fails, an appropriate information dialog is displayed, indicating the error location in the text field and allowing the user to correct their mistakes. And if all validation checks pass, a new Tutor object is created using the value obtained from the text field via the getText( ) method. This Tutor object is then added to the list, which is of type Teacher. The addTutor( ) method performs validation with the assistance of various other methods, briefly introduced below.

Method Name	Description
findTutor (int teacherId)	This method iterates through each teacher object in the Arraylist, compares each one's teacher id with the value supplied in the parameter using the getter method, and returns the matching Tutor object if a match is found.
isValidInput (String … values)	This method checks if there is any required text field empty by comparing all the values passed in the parameter with an empty String value and return ‘false’ if any one text field is empty and, ‘true’ otherwise. Method overloading is used for this method for simplicity and to reduce unnecessary naming of methods.
findTeacher (int teacherId)	This method checks that the teacher id is unique by looping through the Teacher arraylist and determining whether any teachers have been created with the teacher id provided in the input parameter.
isNegative (int … values)	This method checks if the values provided in the parameter are negative. It returns "true" if the values are negative; else, it returns false.

containsNumbers (String value)	This method determines whether or not a number is present in the provided String in the parameter. If the provided String has a number, it will return "true," otherwise it will return "false." It helps users from accidently entering a non-alphabetic value when only alphabetic values are permitted in the text field.
clearTextField ( )	After adding a tutor to the list, this method clears all input text fields to enhance the user experience.

Table 3 : Method Description for Add Tutor Button

4.3 Method Description for Grade Assignment Button

The " Grade Assignment " Button triggers the gradeAsignment( ) method, which retrieves the Teacher ID, Department, Year Of Experience and Graded Score value from the text fields and conducts validation checks. These checks ensure that the values are not empty, negative, or containing alphabetic characters. If the validation fails, an appropriate information dialog is displayed, indicating the error location in the text field and allowing the user to correct their mistakes. And if all validation checks pass, the Grade Assignment procedure is carried out from a Lecturer where gradeAsignment( ) method is called with the values as a parameter which will check if the Lecturer is eligibility of lecturer and if the lecturer is eligible it will perform gradeAssignment with the value obtained from the text field via the getText( ) method and will show the grade in the Information dialog else it will show that the specific teacher is not eligible and thus the grade was not assigned. The gradeAssignment( ) method performs these different function with the help of various other methods which are briefly mentioned below:

Method Name	Description
findLecturer (int teacherId)	This method iterates through all the teacher object present in the Arraylist and compare every teacher object's teacher id using getter method with the value passed in the parameter and If a match is found, the method returns the corresponding Lecturer object for grade Assignment.
isValidInput (String … values)	This function returns "false" otherwise and "true" if any text field is not empty. To do this, it compares each value provided in the parameter to an empty String value.
isNegative (int … values)	This method checks if the provided values in the parameter are negative or not. If the values are negative it returns 'true' otherwise 'false'.
containsNumbers (String department)	This method checks if the provided department value in the parameter contains number or not. It will return 'true' if the

	provided department value contains number , ‘false’ otherwise. It helps to assist users in preventing accidental entry a non alphabetic value in the text field mistakenly where only alphabetic values are allowed.
clearTextField ( )	This method will clear all the input text fields after the Lecturer is added to the list to give user a better user experience.

Table 4 : Method Description for Grade Assignment Button

4.4 Method Description for Remove Tutor Button

The " Remove Tutor " Button triggers the removeTutor( ) method, which retrieves the Teacher ID value from the text fields and conducts validation checks. These checks ensure that the Teacher ID is not empty, negative, or containing alphabetic characters. If the validation fails, an appropriate information dialog is displayed, indicating that the value of the Teacher ID in the text field is not appropriate. And if all validation checks pass, It will check if the Tutor exists or not and if the tutor exists then the removal procedure of tutor is carried on where removetutor() method is called of the Tutor class first with the teacherID as a parameter which will set all the instance of tutor to default value and also then the respective Tutor is removed from the list. After removing the tutor from the list it will display an information dialog with message saying “Teacher was removed Succesfully”.The removeTutor( ) method performs these different function with the help of various other methods which are briefly mentioned below:

Method Name	Description
findTutor (int teacherId)	This function loops over every teacher object in the Arraylist and uses the getter method to check each teacher object's teacher id with the value provided in as a parameter. If a match is found, the method returns the corresponding Tutor object for the grade assignment.
isValidInput (String teacherID)	This function returns "false" otherwise and "true" if Teacher ID text field is not empty. To do this, it compares teacherID provided in the parameter to an empty String value.
isNegative (int teacherID)	This method checks if the provided Teacher ID in the input field are negative or not. If the values are negative it returns ‘true’ otherwise ‘false’.
clearTextField ( )	This method will clear all the input text fields after the Tutor is removed from the list to give user a better user experience.

Table 5 : Method Description for Remove Tutor Button

4.5 Method Description for Set Salary Button

The "Set Salary" Button triggers the setSalary( ) method, retrieving Teacher ID, Salary, and Performance Index values from text fields and conducts validation checks. Validation checks ensure non-empty, non-negative, and numeric inputs. If validation fails, an error message prompts the user to correct their input. Upon successful validation, the setsalary( ) method is called of Tutor class which attempts to update the salary and performance index for the Tutor. If successful, a confirmation dialog displays a message saying that “Salary was Updated “ along with the updated salary, performance index, and any bonus amount. If not, a information dialog with message informs the user of the failure saying “Requirements not met for salary increment.”. The setSalary ( ) method performs these different function with the help of various other methods which are briefly mentioned below:

Method Name	Description
findTutor (int teacherId)	This function loops over every teacher object in the Arraylist and uses the getter method to check each teacher object's teacher id with the value provided in as a parameter. If a match is found, the method returns the corresponding Tutor object for the grade assignment.
isValidInput (String... values)	This function returns "false" otherwise and "true" if values provided in the parameter is not empty. To do this, it compares values provided in the parameter to an empty String value.
isNegative (int… values)	This method checks if the provided values in parameter are negative or not. If the values are negative it returns ‘true’ otherwise ‘false’.
clearTextField ( )	This method will clear all the input text fields after the Tutor’s salary is set to give user a better user experience.

Table 6 : Method Description for Set Salary Button

4.6 Method Description for Display Lecturer Button

The "Display Lecturer" Button triggers the displayLecturer( ) method. This method retrieves details of each Lecturer instance from the list by iterating through all objects present in the list. It then populates a JTable with the lecturer's ID, Lecturer Name, Address, Working Type, Employment Status, Working Hour, Department, Year Of Experience, organizing them into rows and columns. Upon calling this method, a new JFrame is displayed on top of the existing frame, containing all lecturer information in a tabular format. This allows users to conveniently view all lecturer details in a single window enhancing the program's user-friendliness.

4.7 Method Description for Display Tutor Button

The "Display Tutor" Button activates the displayTutor( ) method. This method retrieves details of each Tutor instance from the list by iterating through all objects present. It then fills a JTable with the tutor's ID, Name, Address, Salary, Specialization, Academic Qualification, Performance Index and organizing them into rows and columns. When this method is called, a new JFrame appears over the existing frame, presenting all tutor information in a tabular layout. This facilitates users in conveniently accessing all tutor details within one window, making the program more user-friendly.

4.8 Method Description for Clear Button

The "Clear" Button triggers the clearTextFields( ) method, which is designed to reset all text fields within the graphical user interface (GUI) to empty values. Upon calling, this method prompts the user for confirmation before proceeding. If confirmed, it efficiently clears all input fields, ensuring a clean and organized interface for further interactions. This feature helps to maintain a and user-friendly environment for further engagements with the program.

5. Testing

5.1 Test 1:

Test that the program can be compiled and run using the command prompt, including a screenshot like Figure 1 from the command prompt learning aid.

Test Number	1
Objective	Testing if the program can be compiled and run from the command prompt
Action taken	1) Opened Terminal 2) Selected folder path i.e. /Users/shivamthakur/Documents/Java\ S02\ 3) Compiled file first using - javac TeacherGUI.java 4) Executed file using command - java TeacherGUI
Expected Results	Program must be compiled and run using the command prompt
Actual Results	Program was successfully compiled and executed.
Conclusion	The test passed successfully

Table 7 : Testing 01

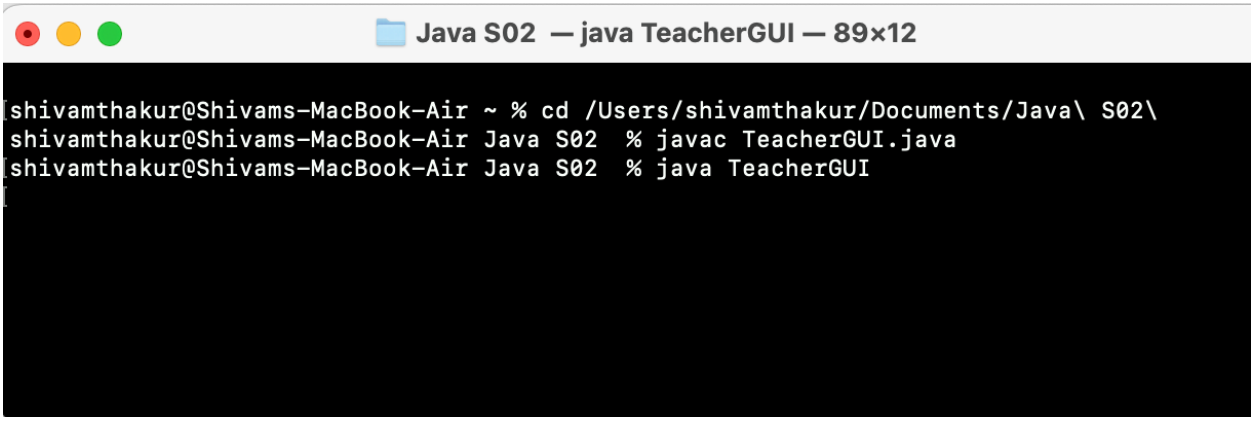


Figure 7 : Testing 01 Screenshot 1

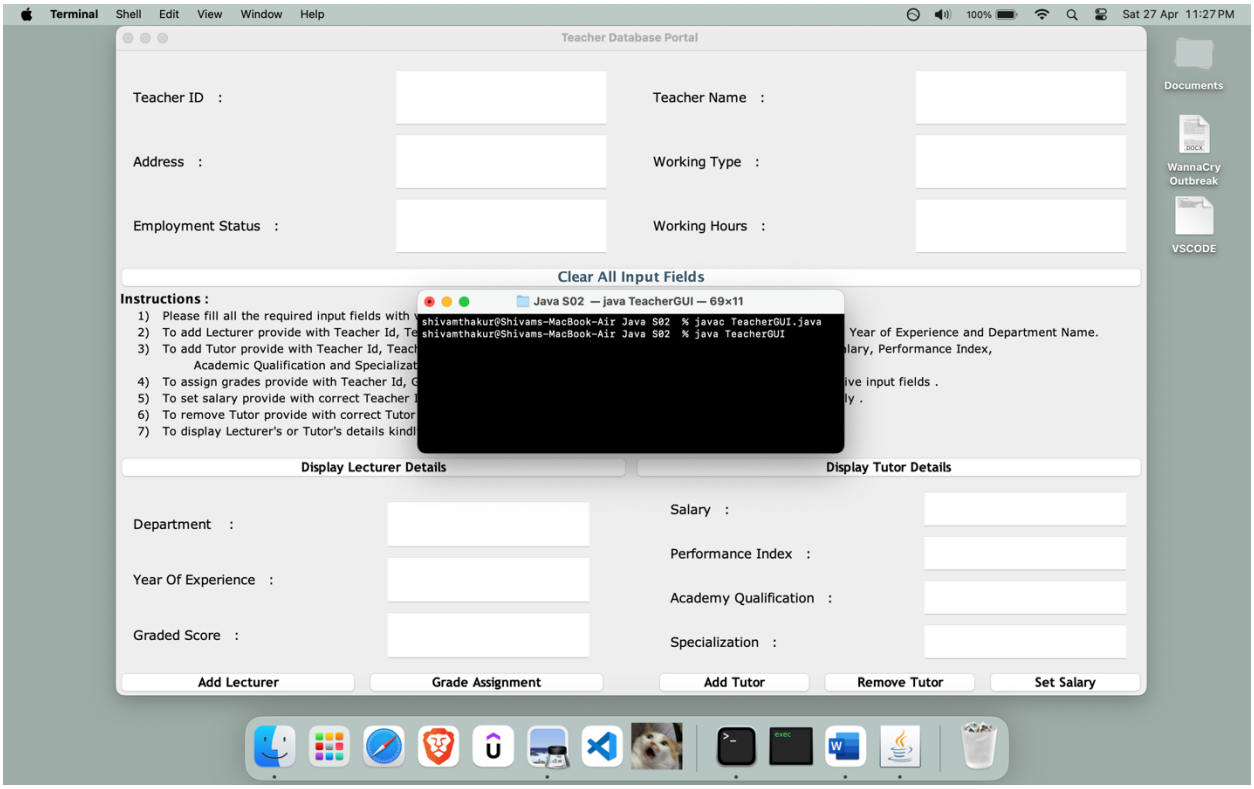


Figure 8 : Testing 01 Screenshot 2

5.2 Test 2 :

Evidence should be shown of :

5.2.1 Test 2.1

Adding the Lecturer to the Array List

Test Number	2.1
Objective	Creating an object of Lecturer by passing appropriate input in text field and pressing the add Lecturer button and adding it to the Array list of Teacher type.
Actions Taken	Step 1 : Compiled and ran TeacherGUI file Step 2 : Entered appropriate values required for creating an object of Lecturer type i.e. • Teacher ID = 1; • Teacher Name = “Pushkar Sah”; • Teacher Address = “Kalanki, Kathmandu”; • Employment Status = “Full Time”; • Working type = “Active”; • Working Hours = 52; • Department = “Information System” • Year Of Experience = 8; Step 3 : Pressed on “Add Lecturer” Button Step 4 : Closed Information Dialog Step 5 : Clicked Display Lecturer Button
Expected Results	After adding the lecturer, when clicked upon display Lecturer button it should pop up the information of the Lecturer in the tabular form which was previously added.

Actual Results	After adding the lecturer, the display lecturer button worked perfectly fine and displayed the whole information of the lecturer added.
Conclusion	The test passed successfully.

Table 8 : Testing 2.1

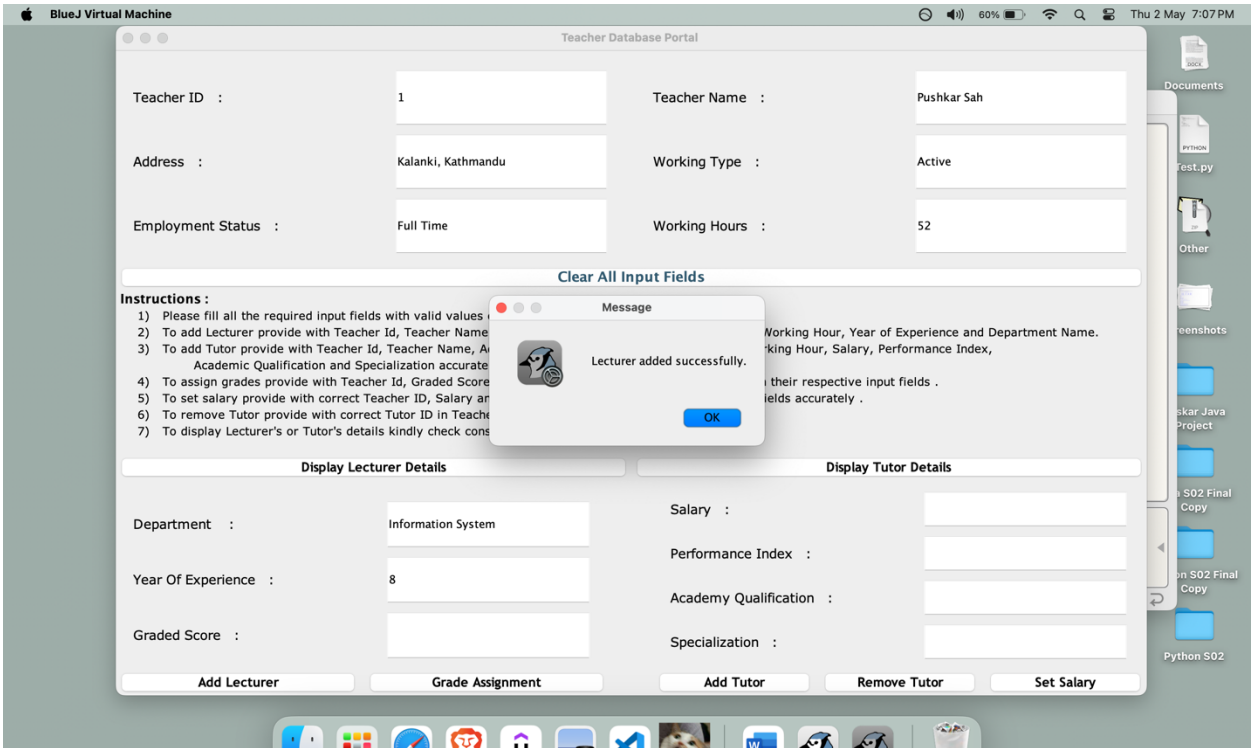


Figure 9 : Testing 2.1 Screenshot 1

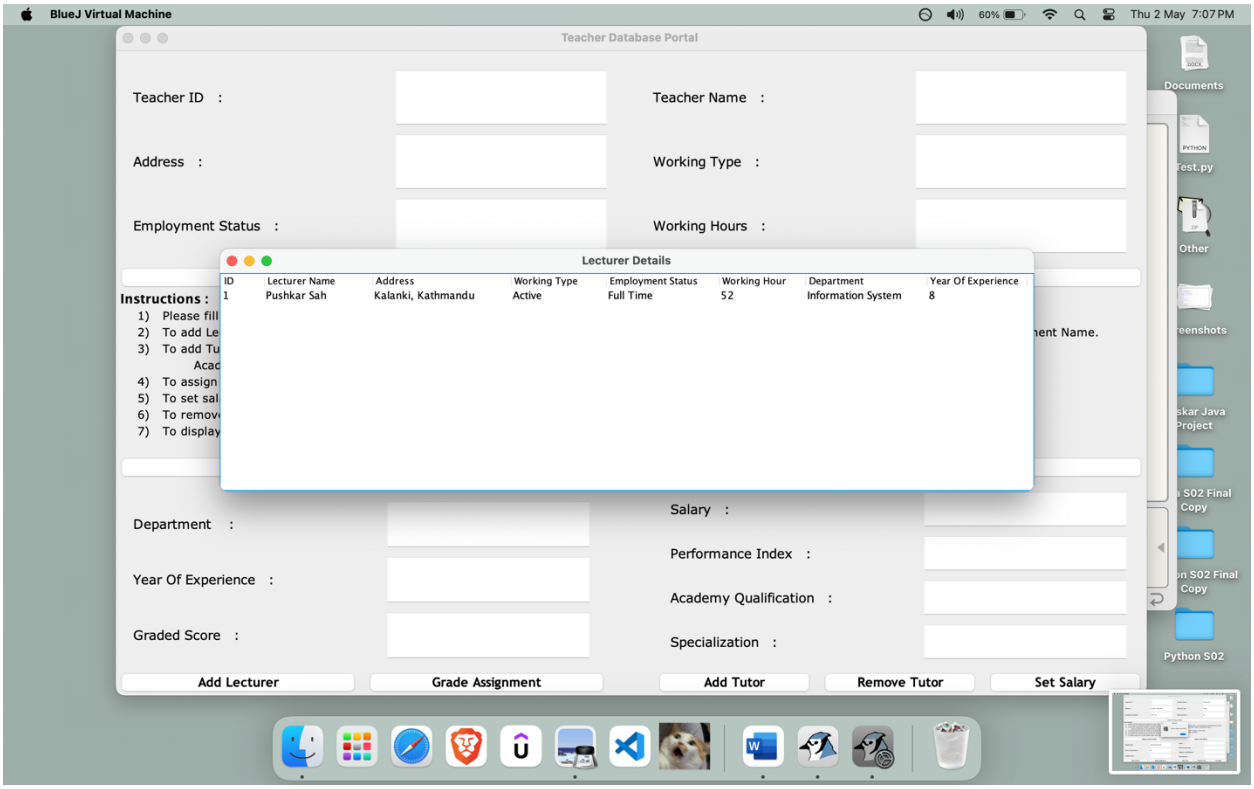


Figure 10 : Testing 2.1 Screenshot 2

5.2.2 Test 2.2

Adding Tutor object to the list.

Test Number	2.2
Objective	Creating an object of Tutor by passing appropriate input in text field and pressing the add Tutor button and adding it to the Array list of Teacher type.
Actions Taken	Step 1 : Compiled and ran TeacherGUI file Step 2 : Entered appropriate values required for creating an object of Tutor type i.e. • Teacher ID = 2; • Teacher Name = “Binayak Prajapati”; • Teacher Address = “Gwarkho, Lalitpur”; • Employment Status = “Part Time”; • Working type = “Inactive”; • Working Hours = 36; • Salary = 45000; • Academic Qualification : “BSc Computing “; • Specialization : “Hardware Architecture” ; • Performance Index = 7 ; Step 3 : Pressed on “Add Tutor” Button Step 4 : Closed Information Dialog Step 5 : Clicked Display Tutor Button
Expected Results	After adding the Tutor, when clicked upon display Tutor button it should pop up the information of the Tutor which was previously added.
Actual Results	After adding the tutor, the display Tutor button worked perfectly fine and displayed the whole information of the Tutor which was recently added.
Conclusion	The test passed successfully.

Table 9 : Testing 2.1

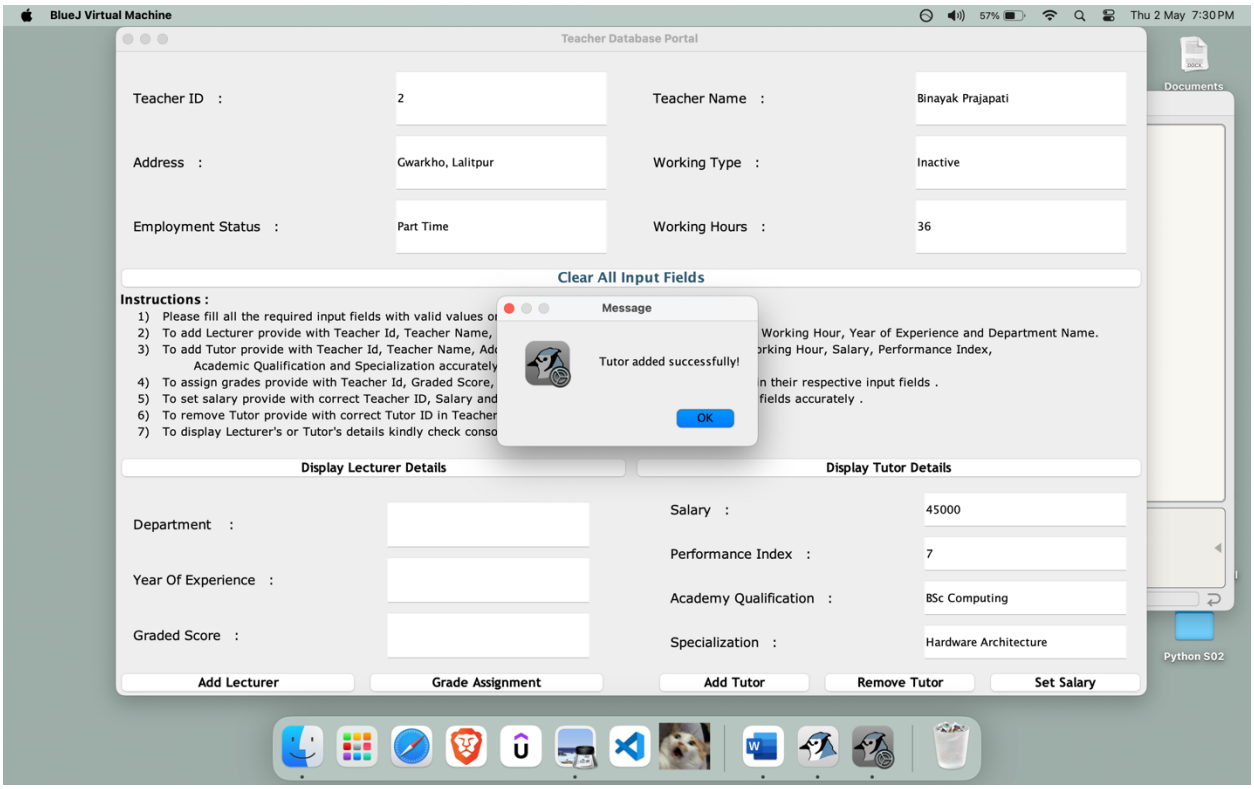


Figure 11 : Testing 2.2 Screenshot 1

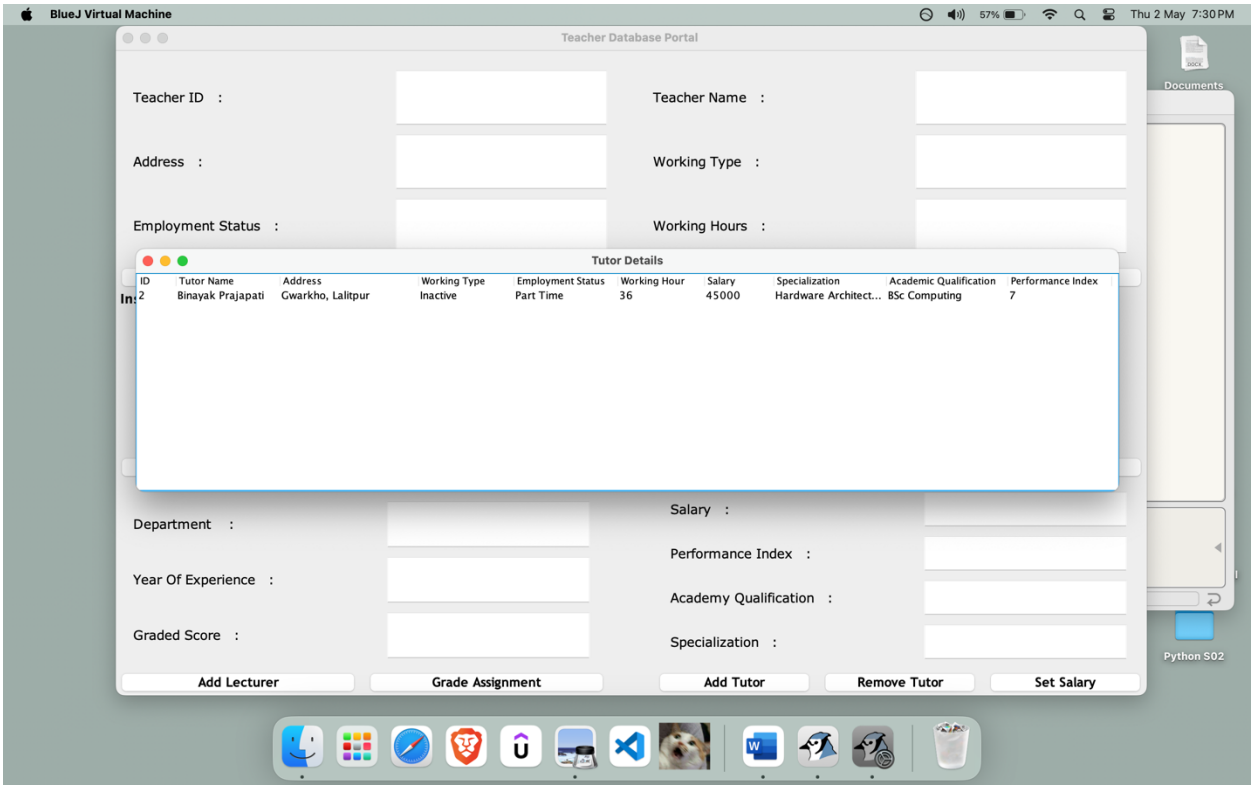


Figure 12: Testing 2.2 Screenshot 2

5.2.3 Test 2.3

Grade Assignment from Lecturer

Test Number	2.3
Objective	Grade Assignment from Lecturer
Actions Taken	Step 1 : Compiled and ran TeacherGUI file Step 2 : Entered appropriate values required for creating an object of Lecturer type i.e. • Teacher ID = 1; • Teacher Name = “Pushkar Sah”; • Teacher Address = “Kalanki, Kathmandu”; • Employment Status = “Full Time”; • Working type = “Active”; • Working Hours = 52; • Department = “Information System” • Year Of Experience = 8; Step 3 : Closed Information Dialog Step 4 : Entered teacher Id, graded score, department, and year of experience (YOE) in their appropriate text fields for grade assignment . i.e. • Teacher ID = 1; • Graded Score = 89; • Department = “Information System”; • Year Of Experience (YOE) = 8; Step 5: Clicked on Grade Assignment Button. Step 6 : Checked Information Dialog for Grades Step 7 : Closed Information Dialog

Expected Results	After grade assignment button is pressed, it should have the information of the grade obtained and other details related to grade assignment in it .
Actual Results	The grade assignment button worked as expected and **(Grade A)** was assigned and the information dialog contained all the necessary information of Teacher related to Grade Assignments .
Conclusion	The test passed successfully.

Table 10 : Testing 2.3

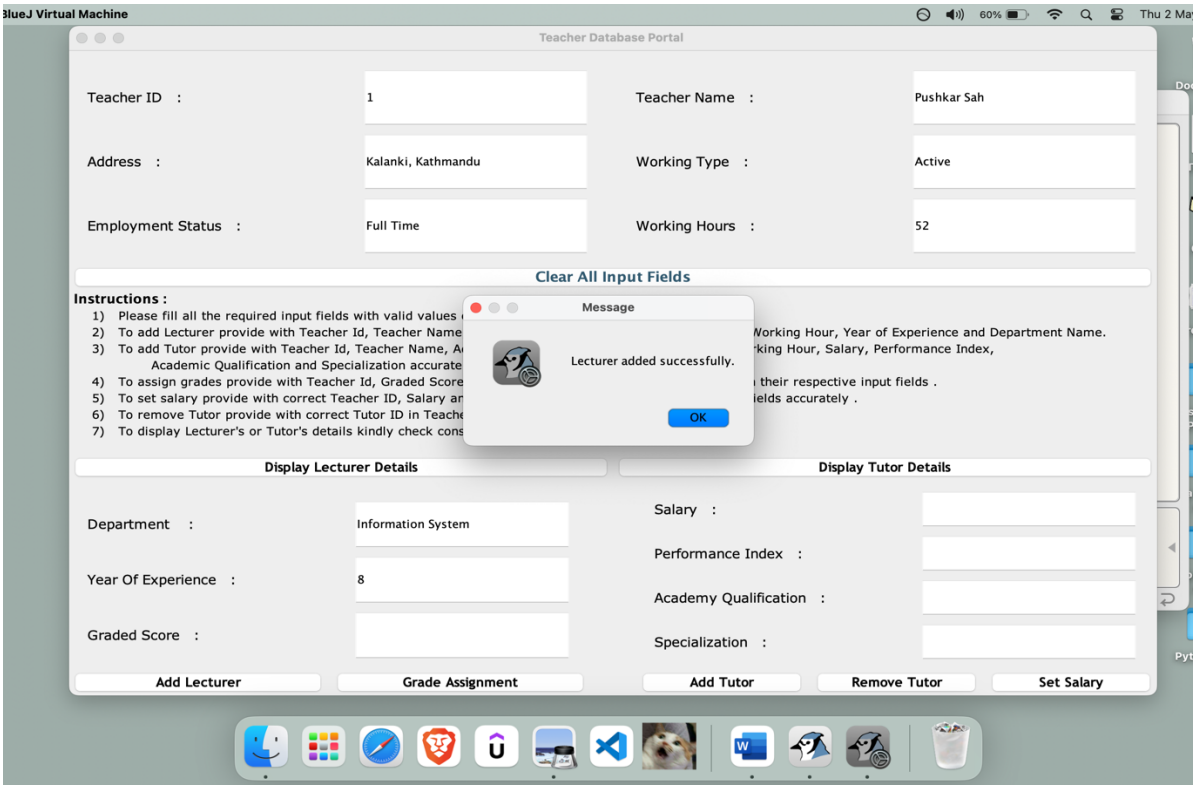


Figure 13 : Testing 2.3 Screenshot 1

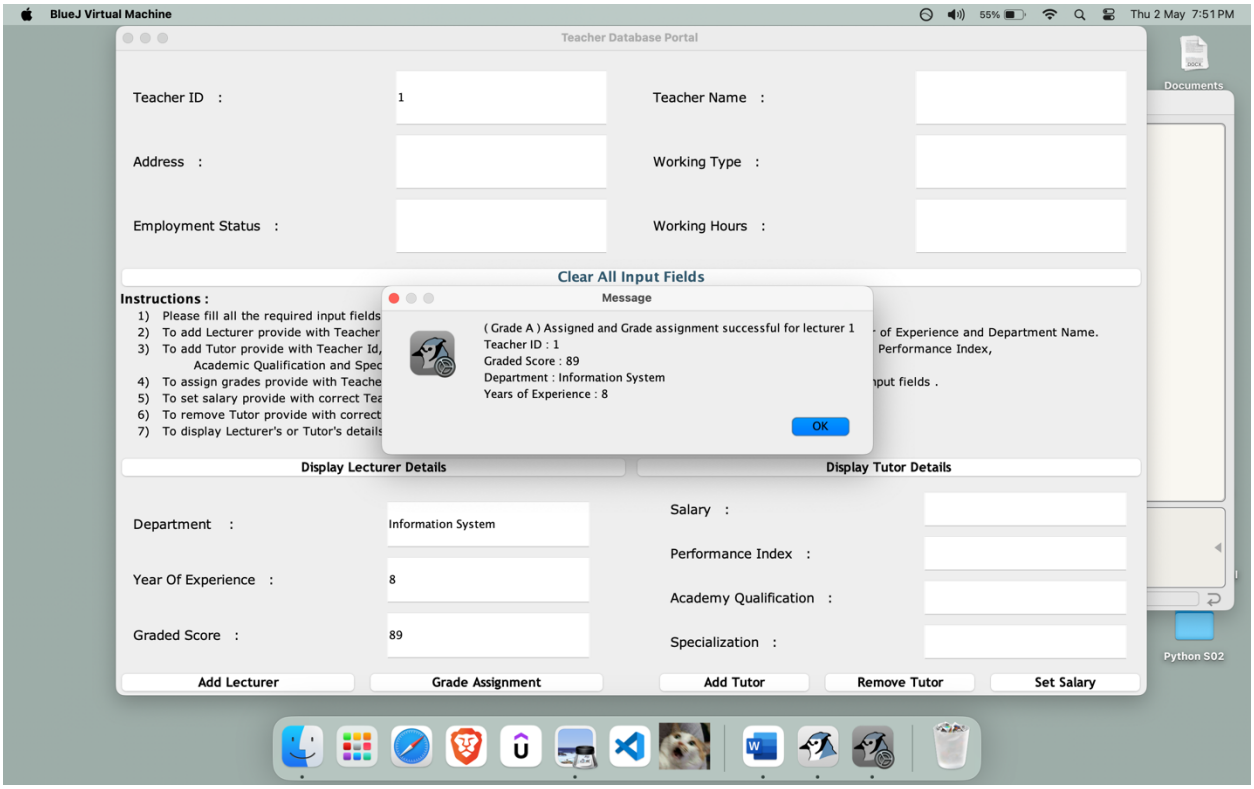


Figure 14 : Testing 2.3 Screenshot 2

5.2.4 Test 2.4

Setting the salary of Tutor Object

Test Number	2.4
Objective	Setting the salary of the Tutor
Actions Taken	Step 1 : Compiled and ran TeacherGUI file Step 2 : Entered appropriate values required for creating an object of Tutor type i.e. • Teacher ID = 2; • Teacher Name = “Binayak Prajapati”; • Teacher Address = “Gwarkho, Lalitpur”; • Employment Status = “Part Time”; • Working type = “Inactive”; • Working Hours = 36; • Salary = 45000; • Academic Qualification : “BSc Computing “; • Specialization : “Hardware Architecture” ; • Performance Index = 7 ; Step 3 : Closed Information Dialog Step 4 : Clicked on Display Tutor Button to show tutor details whose salary we are about to set. Step 5 : Entered Tutor Id in the respective field .i.e. • Teacher ID = 1 ; • Salary = 80000; • Performance Index = 9 ; Step 6 : Clicked on “Set Salary” Button. Step 7 : Checked information dialog for success message. Step 8 : Clicked on Display Tutor Button to check if Tutor’s salary was updated in the list or not.

Expected Results	After "Set Salary" button is pressed, the tutor's class ".setsalary()" method must me called and the tutor's salary must be updated and bonus must be added if he/she is eligible according to the performance index and when pressed on display tutor button the respective tutor's salary must be replaced with the new salary in the JTable.
Actual Results	The "Set Salary" button worked perfectly fine, and the Tutor's salary was updated in the list.
Conclusion	The test passed successfully.

Table 11 : Testing 2.4

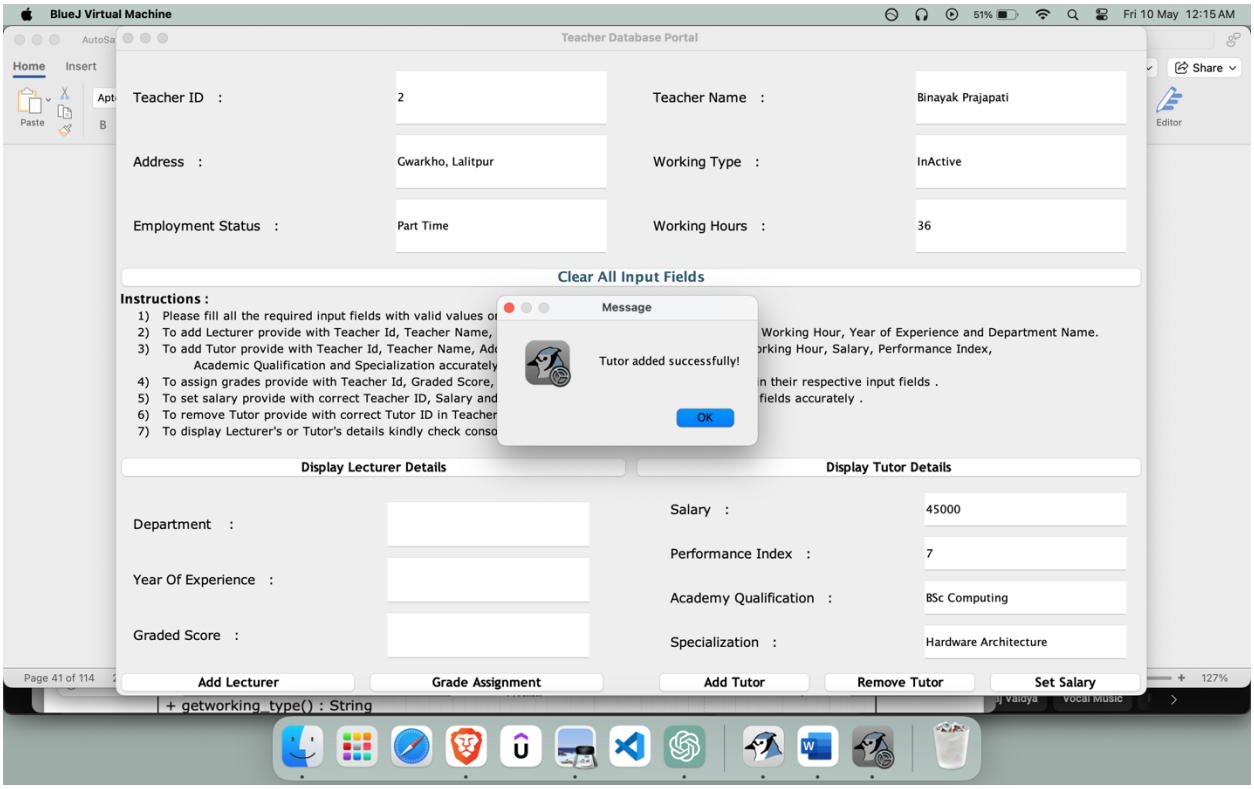


Figure 15 : Testing 2.4 Screenshot 1

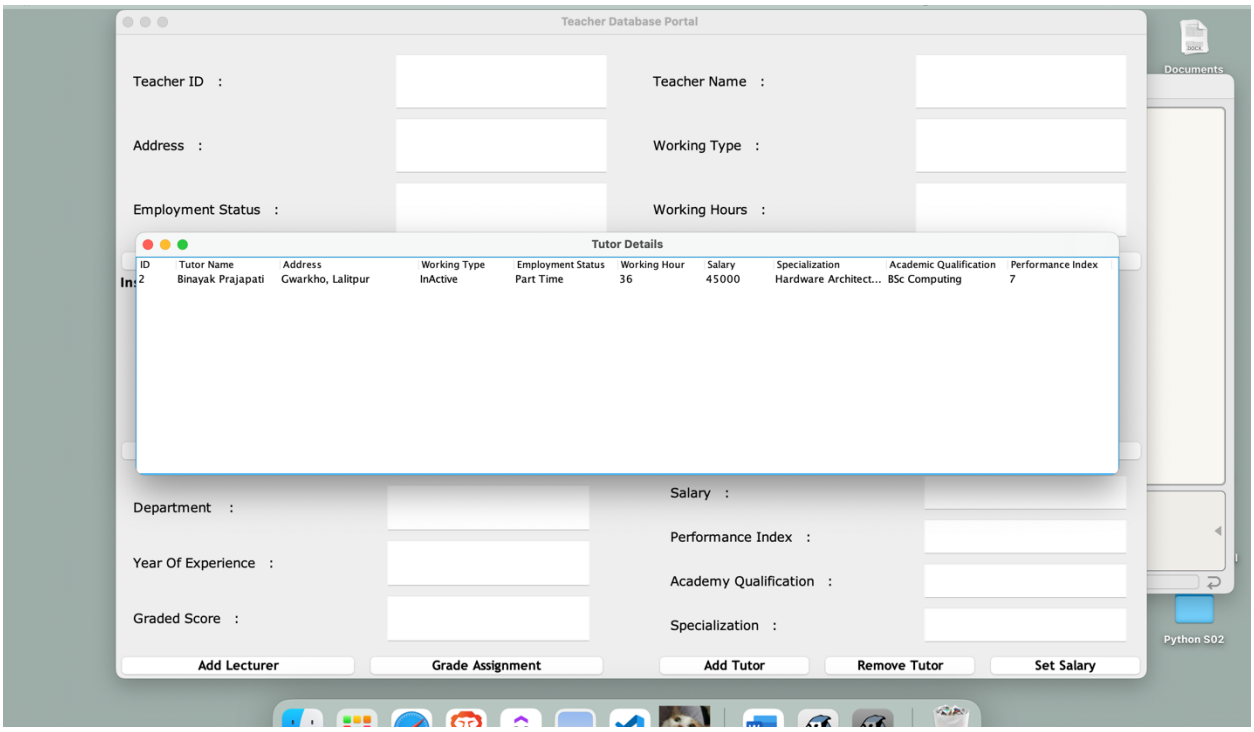


Figure 16 : Testing 2.4 Screenshot 2

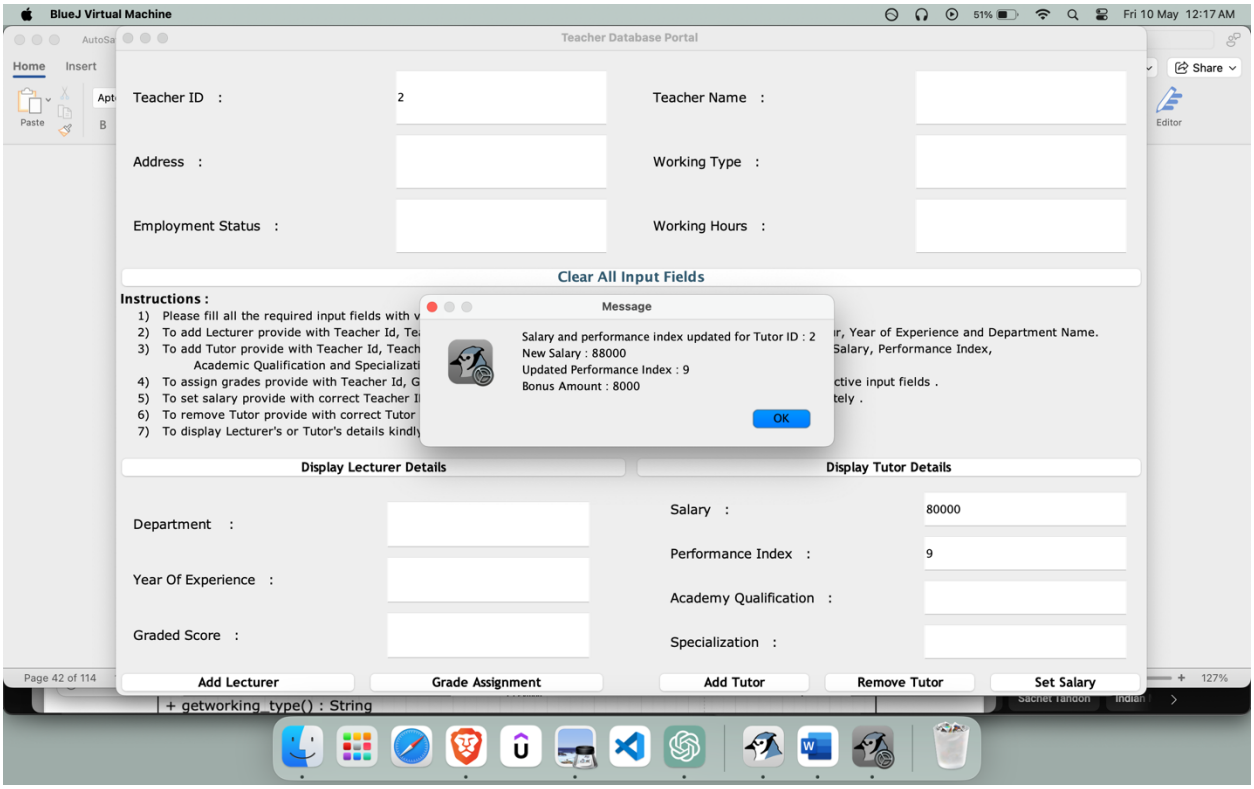


Figure 17 : Testing 2.4 Screenshot 3

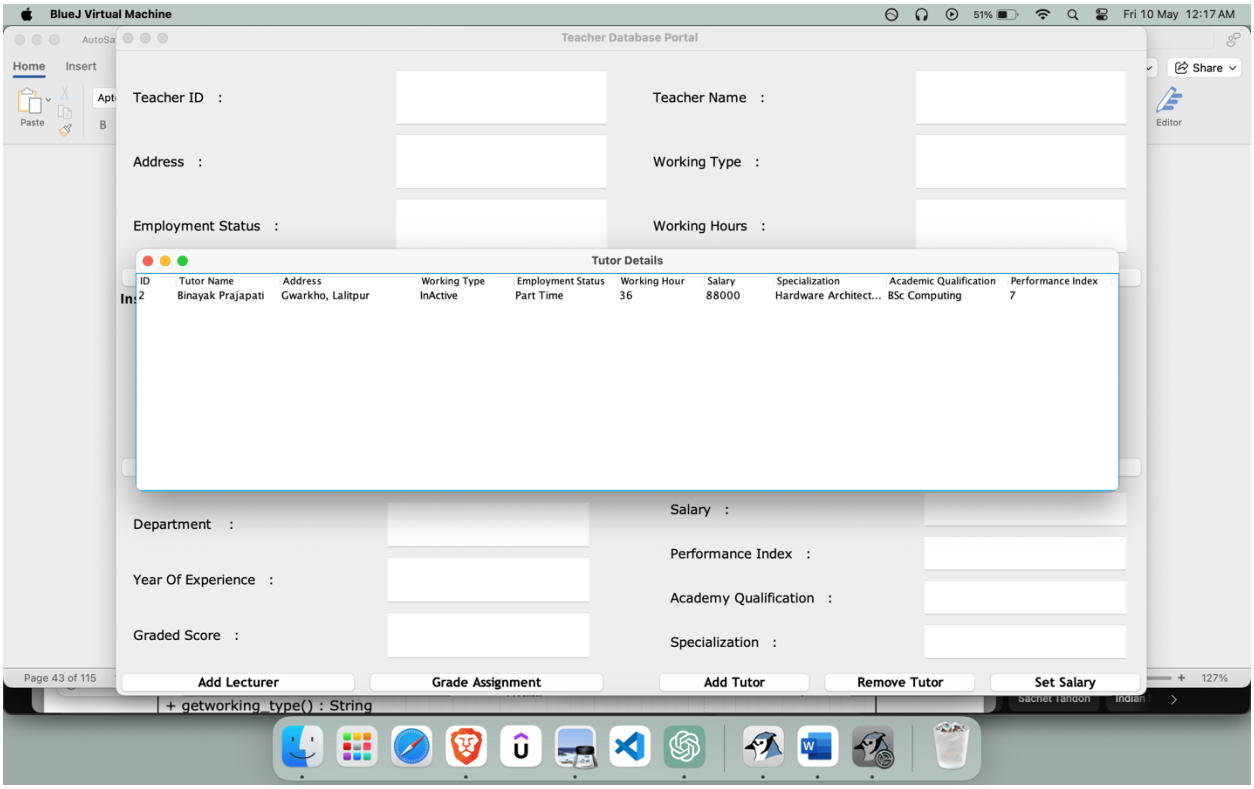


Figure 18 : Testing 2.4 Screenshot 4

5.2.5 Test 2.5

Removing the Tutor object from the list

Test Number	2.5
Objective	Removing a Tutor
Actions Taken	Step 1 : Compiled and ran TeacherGUI file Step 2 : Entered appropriate values required for creating an object of Tutor type i.e. • Teacher ID = 2; • Teacher Name = “Binayak Prajapati”; • Teacher Address = “Gwarkho, Lalitpur”; • Employment Status = “Part Time”; • Working type = “Inactive”; • Working Hours = 36; • Salary = 45000; • Academic Qualification : “BSc Computing “; • Specialization : “Hardware Architecture” ; • Performance Index = 7 ; Step 3 : Closed Information Dialog Step 4 : Clicked on Display Tutor Button to show tutor details which we are about to remove from the list. Step 5 : Entered Tutor Id in the respective field .i.e. • Teacher ID = 1 ; Step 6 : Clicked on Remove Tutor Button. Step 7 : Checked information dialog for success message. Step 8 : Clicked on Display Tutor Button to check if tutor was removed from the list or not.

Expected Results	After remove tutor button is pressed, the tutor's class ".removetutor()" method must me called and the tutor must be removed from the list and when pressed on display tutor button the respective tutor which was removed must not be present in the JTable.
Actual Results	The remove tutor button worked perfectly fine and the tutor was removed from the list and the list was empty.
Conclusion	The test passed successfully.

Table 12 : Testing 2.5

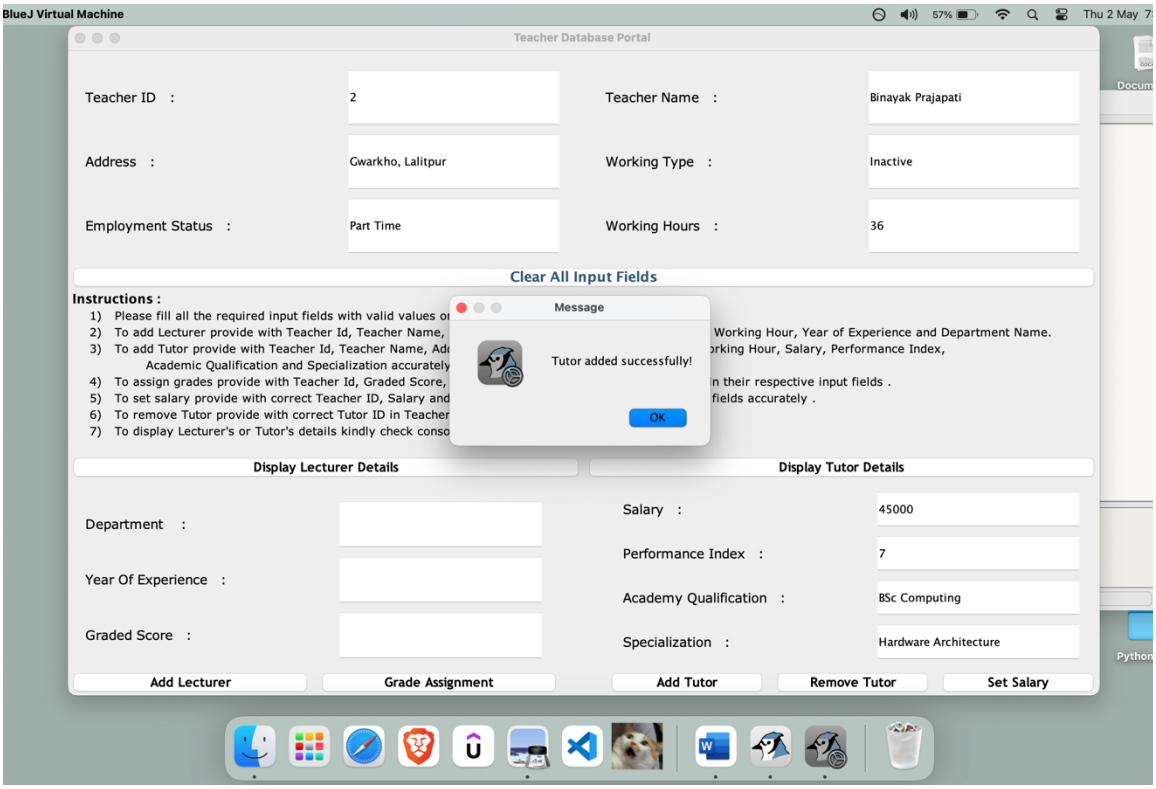


Figure 19 : Testing 2.5 Screenshot 1

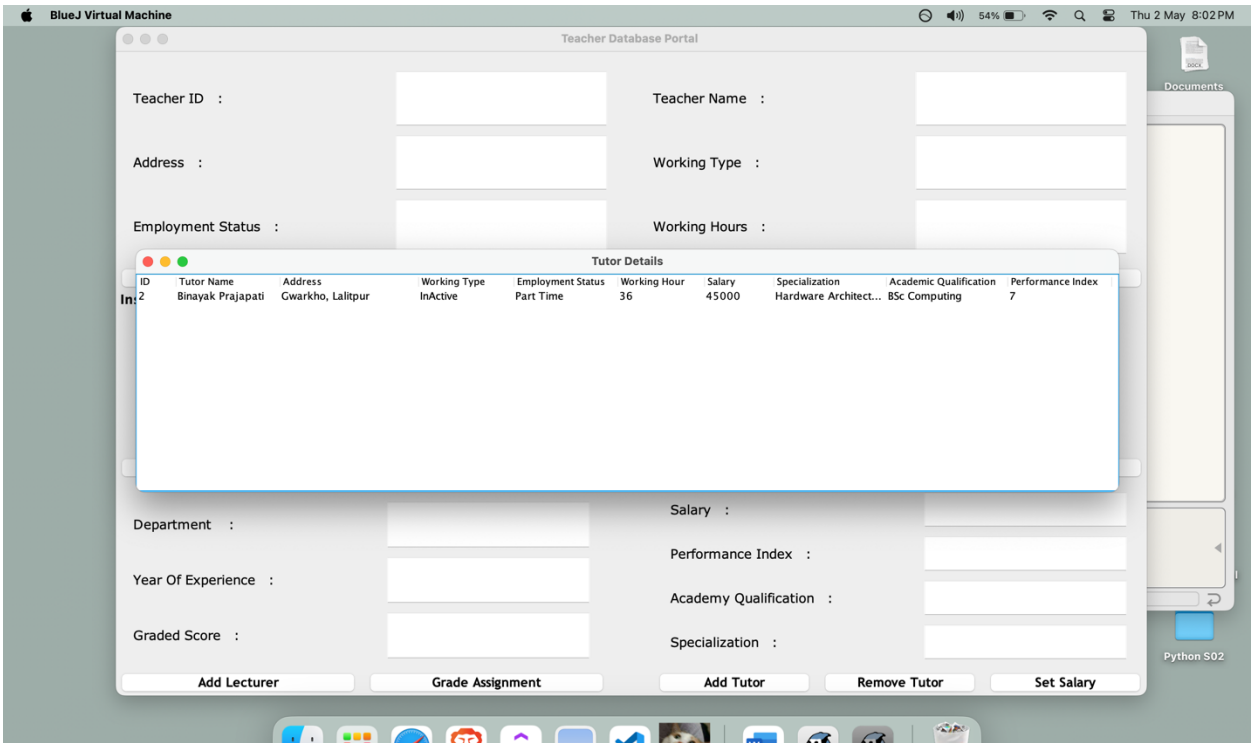


Figure 20 : Testing 2.5 Screenshot 2

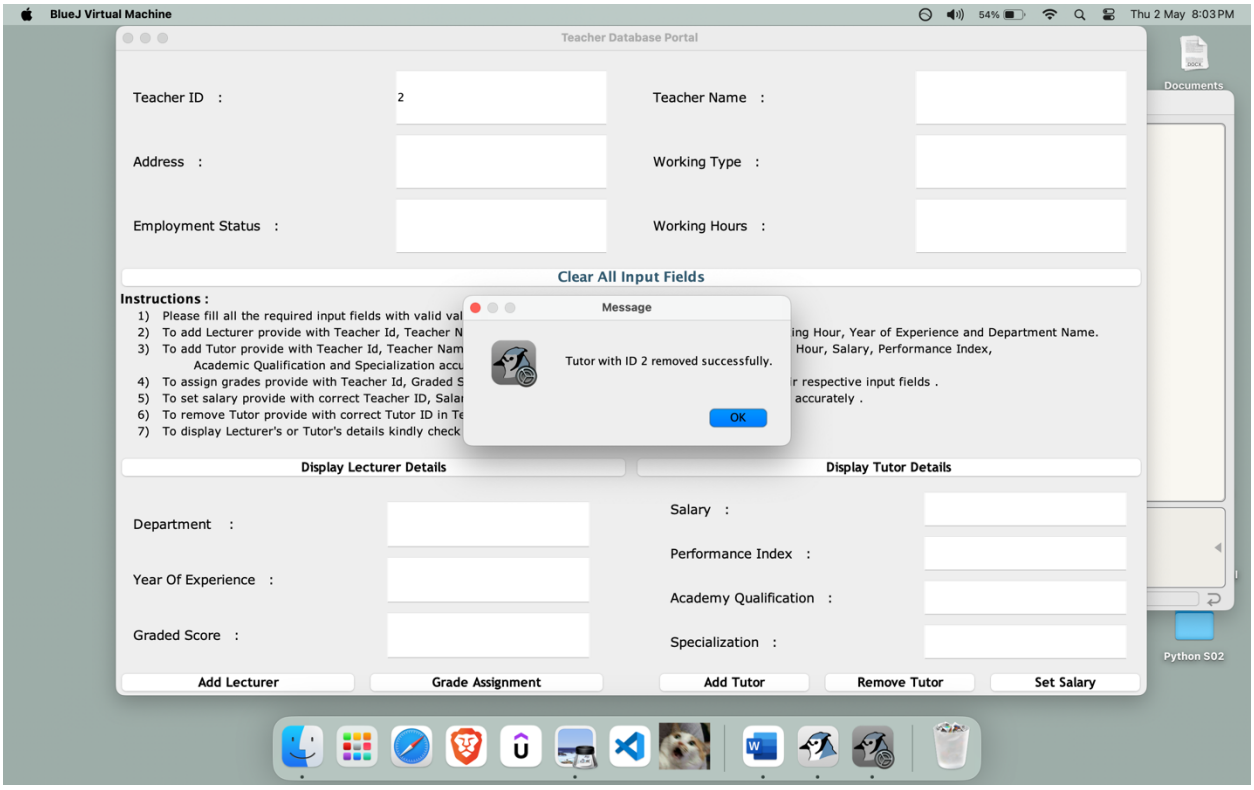


Figure 21 : Testing 2.5 Screenshot 3

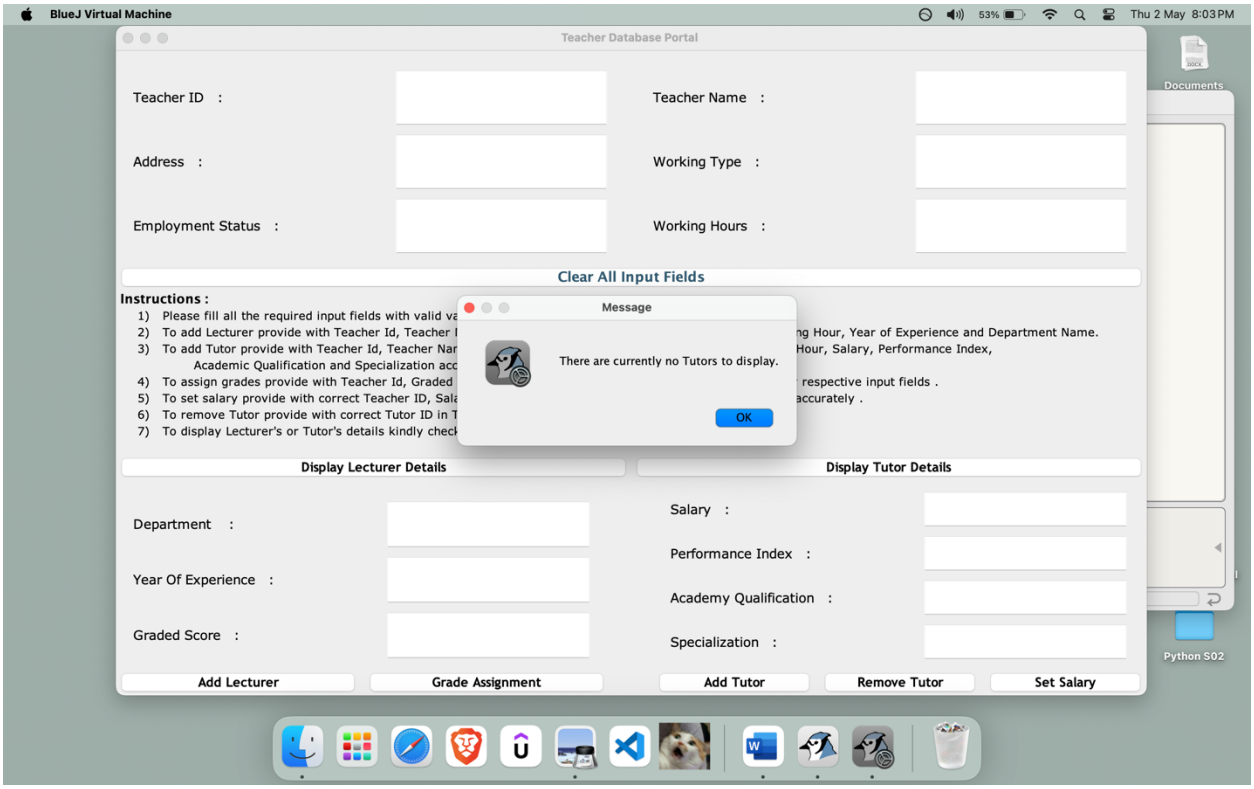


Figure 22 : Testing 2.5 Screenshot 4

5.3 Test 3

Test that appropriate dialog boxes appear when unsuitable values are entered for the Teacher ID.

Test Number	3
Objective	Testing if appropriate dialog boxes appear or not if unsuitable values are entered for Teacher ID.
Action taken	Step 1 : Compiled and ran TeacherGUI file Step 2 : Entered appropriate values for everything except for Teacher ID • **Teacher ID =”first”;** • Teacher Name = “Shivam Thakur”; • Teacher Address = “Patan, Lalitpur”; • Employment Status = “Full Time”; • Working type = “Active”; • Working Hours = 48; • Department = “Programming” • Year Of Experience = 6;
Expected Results	An appropriate dialog box with a suitable message showcasing that an inappropriate input was entered in the Teacher ID or any input field.
Actual Results	An appropriate dialog box appeared when an incorrect datatype or alphabetic value was entered in the teacher id input text field which primarily accepts only Integer or numeric value.
Conclusion	The test passed successfully.

Figure 23 : Testing 3

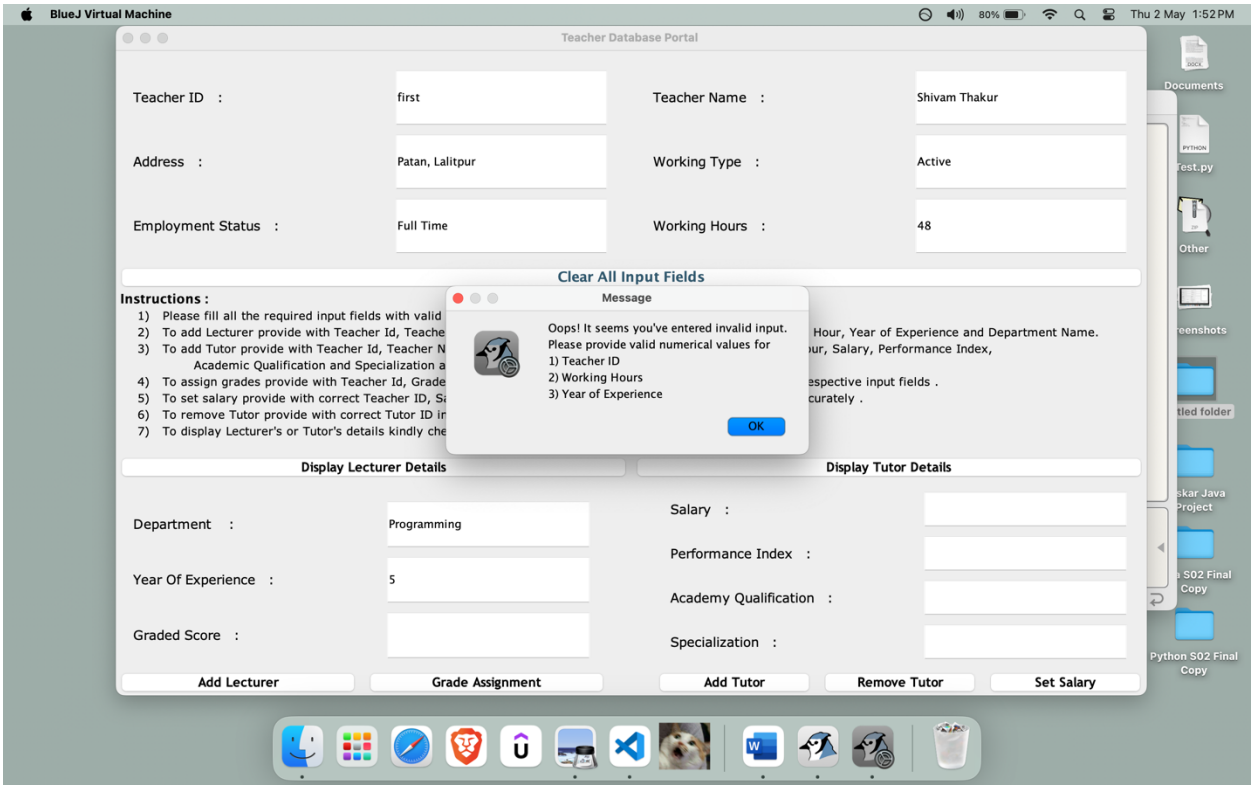


Figure 24 : Testing 3 Screenshot 1

6. Errors Encountered

6.1 Syntax Error

A syntax Error occurs when the rule of programming is violated during the coding process. In Simple terms, it is encountered when the code is not written in the way in which the compiler or interpreter can understand. It can be generally detected at compile time and will throw errors which needs to be compulsorily fixed before the program can be executed. Examples of syntax errors include missing semicolons, mismatched parentheses, and misspelled keywords.

While engaged in the coursework, I stumbled upon various syntax errors among which one was while attempting to use JOptionPane. At first, I thought it takes only one parameter which was the Information message string but as I encountered it and started to do some research on it I figured it takes at least two parameters and 5 at most. So I learned about all about the parameters and corrected the error by adding the parent component and message string as required.

Error Detection :

```
if (isNegative(teacherId, salary, performanceIndex)) {
    JOptionPane.showMessageDialog(
            "Invalid input! Input Field cannot have Negative Values");
    return;
}
```

Figure 25 : Syntax Error Occurrence

Error Correction :

```
if (isNegative(teacherId, salary, performanceIndex)) {
    JOptionPane.showMessageDialog(null,
            "Invalid input! Input Field cannot have Negative Values");
    return;
}
```

Figure 26 : Syntax Error Removal

6.2 Logical Error

A logical error occurs when the code executes without any problem or any syntax or runtime errors but does not give desired output due to the flaw in logic or algorithm of the program. It is also known as Semantic Error and does not give any runtime or compile time errors but fixing it requires lots of debugging and understanding of how the program is intended to work. Examples of logical errors include incorrect calculations, infinite loop, and incorrect comparisons.

Throughout my coursework, I stumbled upon many logical errors in which most common were returning different value other than intended value and incorrect comparisons. But it was fun debugging all those and really helped me in improving my debugging skills. In one of those error, I was adding a value which was returned by addTutor() method to the Array List of type Teacher if the value returned was null but it should have been added to the list if the value was not equal to null. So after getting some Null Pointer Exceptions and debugging I figured out the problem and corrected it.

Error Detection :

```
// for addtutorbutton
addTutorButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Teacher teacher2 = addTutor();
        if (teacher2 != null) {
            teachers.add(teacher2);
        }
    }
});
```

Figure 27 : Logical Error Occurrence

Error Correction :

```
// for addtutorbutton
addTutorButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Teacher teacher2 = addTutor();
        if (teacher2 == null) {
            teachers.add(teacher2);
        }
    }
});
```

Figure 28 : Logical Error Removal

6.3 Run Time Error

A run time error occurs when a running program runs into issues while executing some operations due to various reasons like division by zero, attempting to access an invalid memory location, or trying to perform an operation on an incompatible data type. Runtime errors often lead to the termination of the program or unexpected behavior.

During my coursework, I ran into a run time error which was typically a Number Format Exception and program was crashing because of it. After careful consideration I figured out that when I was converting a string with some alphabet to an Integer value with the help of Integer.parseInt( ) method to do any operations it was throwing that error so as mentioned in the Coursework Guidelines I used try catch block to catch the Number Format Exception and display suitable message in JOptionPane if string is entered in integer field which solved the issue and also added one more exception to catch all other remaining exceptions if encountered.

Error Detection :

```
public Lecturer addLecturer() {
    String teacherID = teacherIDTextField.getText().trim();
    String teacherName = teacherNameTextField.getText().trim();
    String address = addressTextField.getText().trim();
    String workingType = workingTypeTextField.getText();
    String employmentStatus = employmentStatusTextField.getText().trim();
    String workingHoursStr = workingHoursTextField.getText().trim();
    String department = departmentTextField.getText().trim();
    String yearOfExperienceStr = yearOfExperienceTextField.getText().trim();

    if (!isValidInput(teacherID, teacherName, address, workingType, employmentStatus, workingHoursStr, department,
            yearOfExperienceStr)) {
        JOptionPane.showMessageDialog(null, "Invalid input! Please check for missing input fields.");
        return null;
    }

        if (findTeacher(Integer.parseInt(teacherID)) == null) {

            int teacherId = Integer.parseInt(teacherID);
            int workingHours = Integer.parseInt(workingHoursStr);
            int yearOfExperience = Integer.parseInt(yearOfExperienceStr);

            if (isNegative(teacherId, workingHours, yearOfExperience)) {
                JOptionPane.showMessageDialog(null,
                        "Invalid input! Input Field cannot have Negative Values");
                return null;
            }
            // new Lecturer object
            Lecturer lecturer = new Lecturer(teacherId, teacherName, address, workingType, employmentStatus,
                    workingHours, department, yearOfExperience);

            JOptionPane.showMessageDialog(null, "Lecturer added successfully!");
            clearTextField();
            return lecturer;

        } else {
            JOptionPane.showMessageDialog(null,
                    "Teacher with ID : " + Integer.parseInt(teacherID) + " already exists.");
        }

    return null;
}
```

Figure 29 : Runtime Error Occurrence

Error Correction :

```
public Lecturer addLecturer() {
    String teacherID = teacherIDTextField.getText().trim();
    String teacherName = teacherNameTextField.getText().trim();
    String address = addressTextField.getText().trim();
    String workingType = workingTypeTextField.getText();
    String employmentStatus = employmentStatusTextField.getText().trim();
    String workingHoursStr = workingHoursTextField.getText().trim();
    String department = departmentTextField.getText().trim();
    String yearOfExperienceStr = yearOfExperienceTextField.getText().trim();

    if (!isValidInput(teacherID, teacherName, address, workingType, employmentStatus, workingHoursStr, department,
            yearOfExperienceStr)) {
        JOptionPane.showMessageDialog(null, "Invalid input! Please check for missing input fields.");
        return null;
    }

    try {
        if (findTeacher(Integer.parseInt(teacherID)) == null) {

            int teacherId = Integer.parseInt(teacherID);
            int workingHours = Integer.parseInt(workingHoursStr);
            int yearOfExperience = Integer.parseInt(yearOfExperienceStr);

            if (isNegative(teacherId, workingHours, yearOfExperience)) {
                JOptionPane.showMessageDialog(null,
                        "Invalid input! Input Field cannot have Negative Values");
                return null;
            }
            // new Lecturer object
            Lecturer lecturer = new Lecturer(teacherId, teacherName, address, workingType, employmentStatus,
                    workingHours, department, yearOfExperience);

            JOptionPane.showMessageDialog(null, "Lecturer added successfully!");
            clearTextField();
            return lecturer;

        } else {
            JOptionPane.showMessageDialog(null,
                    "Teacher with ID : " + Integer.parseInt(teacherID) + " already exists.");
        }
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null,
                "Invalid input! Please fill input fields with their specific values only.");
        return null;
    } catch (Exception y) {
        JOptionPane.showMessageDialog(null,
                "Invalid input! Lecturer was not added.");
        return null;
    }

    return null;
}
```

Figure 30 : Runtime Error Removal

6.4 Semantic Error

A semantic error is when our code has a problem that confuses the computer, even though we followed the programming rules correctly. This confusion can make the program perform unexpectedly or stop working, even though we didn't make any obvious mistakes in our code.

During my coursework, I encountered a semantic error in my code. Initially, in my method containsNumber( ) to check if a string contains numbers, I mistakenly used Character.isLetter( ) instead of Character.isDigit( ) to identify digits. This semantic error caused the function to incorrectly identify letters as numbers, leading to unexpected outcome. After realizing my mistake, I corrected the code by using the appropriate method i.e. isDigit( ) to accurately identify numeric digits.

Error Detection :

```
/**
 * Checks if the given string contains any numeric digits.
 * takes input and cheak each character using charat and compare it with number using isDigit() method
 * returns true if if contains number, otherwise false
 */
public static boolean containsNumbers(String value) {

    // Iterating over each character in value
    for (int i = 0; i < value.length(); i++) {
        char character = value.charAt(i);
        // Checking if the character is the digit
        if (Character.isLetter(character)) {
            return true;        // Return true if a digit is found
        }
    }
    // Return false if no digits are found
    return false;
}
```

Figure 31 : Semantic Error Occurrence

Error Correction :

```java
/**
 * Checks if the given string contains any numeric digits.
 * takes input and cheak each character using charat and compare it with number using isDigit() method
 * returns true if if contains number, otherwise false
 */
public static boolean containsNumbers(String value) {

    // Iterating over each character in value
    for (int i = 0; i < value.length(); i++) {
        char character = value.charAt(i);
        // Checking if the character is the digit
        if (Character.isDigit(character)) {
            return true;        // Return true if a digit is found
        }
    }
    // Return false if no digits are found
    return false;
}
```

Figure 32 : Semantic Error Removal

7. Conclusion

In conclusion, I believe that working on this project has taught me a lot, especially about building Java Swing GUI development and incorporating it with the Object Oriented Concept of Java. I had a lot of difficulties when I first started using Swing, which forced me to go outside of my comfort zone and learn new ideas and come up with numerous ways of solving that problem.

One of the main lessons I learned was how to build a GUI interface using Swing components. At first, I had trouble figuring out how to use layout and arrange everything according on the Frame and use Table to contain details of the Lecturer and Tutor. But after researching a lot, practicing, and watching tutorials online on YouTube to learn more about organizing layouts and adding tables to the frame along with the help I get from my tutor. I came up with some creative solutions and was able to understand all the concept very clearly and use the GUI component to make the interface more unique and user-friendly.

As I look forward, I appreciate the valuable skills and knowledge I've gained from this project. Working hands-on with Swing GUI components and ArrayLists has given me a strong foundation for what lies ahead of me in my career. I'm sure that working on this project has not only improved my GUI development skills and but equipped me to be prepared and work with more advanced concepts and tools in the industry.

I am immensely grateful for the guidance and support provided by **Mr.Ujjwal Adhikari** and **Mr Pramodh Tuladhar** for helping me and guiding me in this course work and helping me debug some of my errors and bugs in my coding section which I really appreciate and would be thankful for the rest of my life.

Overall, I'm really grateful for the chance to learn and grow through this project. Although it was not a complete real-world project, it was a big step towards applying what we learned in college to real-life situations. As I move on to other projects, I'll take with me the lessons I've learned from this coursework.

8. Bibliography

Technopedia, 2011. *What is Java Swing? - Definition from Technopedia.* [Online]
Available at: https://www.techopedia.com/definition/26102/java-swing
[Accessed 27 4 2024].

Computer Hope , 2020. *What is Draw.io ?.* [Online]
Available at:
https://www.computerhope.com/jargon/d/drawio.htm#:~:text=Designed%20by%20Seibert%20Media%2C%20draw,%2Dof%2Da%2Dkind.
[Accessed 9 5 2024].

GeeksforGeeks, 2022. *Introduction of BlueJ -GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-of-bluej/
[Accessed 9 5 2024].

GeeksforGeeks, 2021. *Introduction to Microsoft Word - GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-to-microsoft-word/
[Accessed 9 5 2024].

Java, 2024. *Java in Action : Java + BlueJ.* [Online]
Available at: https://www.java.com/en/java_in_action/bluej.jsp
[Accessed 9 5 2024].

GeeksforGeeks, 2024. *Class Diagram | Unified Modeling Language (UML).* [Online]
Available at: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/
[Accessed 9 5 2024].

GeeksforGeeks, 2023. *What is PseudoCode: A Complete Tutorial - GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/what-is-pseudocode-a-complete-tutorial/
[Accessed 10 5 2024].

9. Appendix

9.1 Code for Teacher Class

Teacher.java

```
public class Teacher {
    // Decleration Of Instance Variables
    private int teacher_id;
    private String name;
    private String address;
    private String working_type;
    private String employment_status;
    private int working_hours;

    // Constructor Creation For Assigning The Values Of Instance Variable
    public Teacher(int teacher_id, String name, String address, String working_type,
String employment_status) {

        this.teacher_id = teacher_id;
        this.name = name;
        this.address = address;
        this.working_type = working_type;
        this.employment_status = employment_status;

    }

    public Teacher() {

    }
```

```
// Getter Method to Access All The Instance Variable in Child Classes
public int getteacher_id() {
    return this.teacher_id;
}

public int getworking_hours() {
    return this.working_hours;
}

public String getaddress() {
    return this.address;
}

public String getworking_type() {
    return this.working_type;
}

public String getemployment_status() {
    return this.employment_status;
}

public String getname() {
    return this.name;
}

// Setter Method For Setting The Value Of Instance Variable i.e. (working_hours)
public void setWorkingHour(int working_hours) {
    this.working_hours = working_hours;
}

// Display Method To Display All The Values Of Instance Variables With Suitable
```

```
    // Annotation
    public void display() {

        System.out.println("Teacher_ID = " + this.getteacher_id());
        System.out.println("Teacher_Name = " + this.getname());
        System.out.println("Teacher_Address = " + this.getaddress());
        System.out.println("Working Type = " + this.getworking_type());
        System.out.println("Employee_Status = " + this.getemployment_status());

        // Conditions For Cheaking If Working Hours Has Been Set Or Not
        if (working_hours == 0) // If Working Hour Has Been Set Then Print Out The Set
Working Hour With
                                // Suitable Annotation
        {
            System.out.println("Working Hour Has Not Been Set Yet");

        } else { // If Working Hour Has Been Set Then Print Out The Set Working Hour
With
                 // Suitable Annotation
            System.out.println("Working_Hour = " + this.getworking_hours());
        }

    }

}
```

Table 13 : Code for Teacher Class

9.2 Code for Lecturer Class

Lecturer.java

```
public class Lecturer extends Teacher {
    // Declaring Instance Variables
    private String department;
    private int yearOfExperience;
    private int gradedScore = 0;
    private boolean hasGraded = false;

    // Constructor Creation For Assigning The Values Of Instance Variable
    public Lecturer(int teacher_id, String name, String address, String working_type,
String employment_status,
            int working_hours, String department, int yearOfExperience) {

        super(teacher_id, name, address, working_type, employment_status);
        super.setWorkingHour(working_hours);
        this.department = department;
        this.yearOfExperience = yearOfExperience;

    }

    // Getter Method to Access All The Instance Variable
    public String getdepartment() {
        return this.department;
    }

    public int getyearOfExperience() {
        return this.yearOfExperience;
    }
```

```
    public int getgradedScore() {
        return this.gradedScore;
    }

    public boolean gethasGraded() {
        return this.hasGraded;
    }

    // Setter Method For Setting The Value Of Instance Variable i.e. (gradedScore)
    public void setgradedScore(int gradedScore) {
        this.gradedScore = gradedScore;
    }

    // Method For Assigning The Grade Of Student
    public String gradeAssignment(int gradedScore, String department, int
yearOfExperience) {

        // Check If The YOE Is Greater Than Or Equal to 5 & Teacher's Department Is
Also
        // The Same
        if ((yearOfExperience >= 5) && (this.department.equals(department))) // If It
Satisfies Condition Then Do
        {

            // Check The Value Of GradeScore And Then Assign Grade According To The
Marks
            if (gradedScore > 70) {
                this.gradedScore = gradedScore;
                this.hasGraded = true;// Assign true To hasGraded After Grade Assignment
                return "Grade A";
```

```
        } else if (gradedScore > 60) {
            this.gradedScore = gradedScore;
            this.hasGraded = true;// Assign true To hasGraded After Grade Assignment
            return "Grade B";
        } else if (gradedScore > 50) {
            this.gradedScore = gradedScore;
            this.hasGraded = true;// Assign true To hasGraded After Grade Assignment
            return "Grade C";
        } else if (gradedScore > 40) {
            this.gradedScore = gradedScore;
            this.hasGraded = true;// Assign true To hasGraded After Grade Assignment
            return "Grade D";
        } else if (gradedScore < 40) {
            this.gradedScore = gradedScore;
            this.hasGraded = true;// Assign true To hasGraded After Grade Assignment
            return "Grade E";
        }

    }

    // If YOE And Teacher's Assignment Doesn't Match
    return "Teacher Not Eligible For Grade Assignment";

}

// Display Method To Display The Details Of The Lecturer
public void display() {
    super.display();// Calling Method from Parent to Child

    System.out.println("Department = " + getdepartment());
    System.out.println("Year Of Experience = " + getyearOfExperience());
```

```
        // Checking If Grading Score Has Been Set Or Not
        if (this.gradedScore == 0) // If Its Not Set Display A Suitable Message
        {
            System.out.println("Graded Score Not Assigned  ");
        } else// If Its Set Already Just Print The Output With Suitable Notation
        {
            System.out.println("Graded Score = " + getgradedScore());
        }
    }
}
```

Table 14 : Code For Lecturer Class

9.3 Code for Tutor Class

Tutor.java

```
public class Tutor extends Teacher {
    // Declaring Instance Variables
    private double salary;
    private String specialization;
    private String academic_qualifications;
    private int performance_index;
    private boolean isCertified = false;

    // Constructor Creation For Assigning The Values Of Instance Variable
    public Tutor(int teacher_id, String name, String address, String working_type,
String employment_status,
            int working_hours, double salary, String specialization, String
academic_qualifications,
            int performance_index) {
        super(teacher_id, name, address, working_type, employment_status);
        super.setWorkingHour(working_hours);
        this.salary = salary;
        this.specialization = specialization;
        this.academic_qualifications = academic_qualifications;
        this.performance_index = performance_index;

    }

    // Getter Method to Access All The Instance Variable
    public double getsalary() {
        return this.salary;
    }
```

```
public String getspecialization() {
    return this.specialization;
}

public String getacademic_qualifications() {
    return this.academic_qualifications;
}

public int getperformance_index() {
    return this.performance_index;
}

public boolean getisCertified() {
    return this.isCertified;
}

// Setter Method For Setting The Value Of Instance Variable i.e. (salary) After
// Giving Appraisal As Per Their Work
public boolean setsalary(int salary, int performance_index) {

    if ((this.performance_index > 5) && (this.getworking_hours() > 20)) {
        if ((performance_index >= 5) && (performance_index <= 7)) {
            this.salary = salary + ((double) 5 / 100) * salary;
            this.isCertified = true;
            return true;
        } else if ((performance_index >= 8) && (performance_index <= 9)) {
            this.salary = salary + ((double) 10 / 100) * salary;
            this.isCertified = true;
            return true;
        } else if (performance_index == 10) {
```

```
            this.salary = salary + ((double) 20 / 100) * salary;
            this.isCertified = true;
            return true;
        } else {
            return false;
        }

    } else {
        // System.out.println("Salary Has Not Been Approved");
        return false;
    }

}

// Method For RemovingTutor If Tutor Has Not Been Certified Yet
public String removetutor() {

    if (isCertified == false) // Check The Value Of (isCertified) If Its False Then
Assign Values To The Tutor
    {
        // Assigning the Values Of The Instance Variable To None And Zero
        this.salary = 0d;
        this.specialization = "";
        this.academic_qualifications = "";
        this.performance_index = 0;

        this.isCertified = false; // Assigning The Value Of isCertified To False
        return "Tutor Removed Successfully"; // Displaying A Suitable Message If
Tutor Is Removed
    } else {
```

```
            return "Tutor is cerified & cannot be removed";// Displaying A Suitable
Message To Convey That Tutor Was Not
                                                // Removed
        }
    }

    // Method Overwriting To Display With Suitable Annotation
    public void display() {
        if (isCertified) {
            // Calling Display Method Of Teacher To Display The Information Of Teacher
            super.display();

            // Display Additional Information
            System.out.println("Salary = " + this.getsalary());
            System.out.println("Specialization = " + this.getspecialization());
            System.out.println("Academic Qualification = " +
this.getacademic_qualifications());
            System.out.println("Performance Index = " + this.getperformance_index());

        } else {
            super.display(); // Display Information Of Teacher
        }
    }

}
```

Table 15 : Code For Tutor Class

9.4 Code for TeacherGUI Class

TeacherGUI.java

```
// Importing Essentials for GUI
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JTable;

class MyFrame extends JFrame {



   // ArrayList Declaration to Store Lecturer and Tutors Of Teacher Class
   private ArrayList<Teacher> teachers = new ArrayList<>();

   // decleration of every JLabel Elements used in program
   private JLabel teacherIDLabel, teacherNameLabel, addressLabel,
workingTypeLabel,
         employmentStatusLabel, workingHoursLabel, departmentLabel,
gradedScoreLabel,
         yearOfExperienceLabel, salaryLabel, performanceIndexLabel,
         academyQualificationLabel, specializationLabel;

   // decleration of every JLabel Elements used in program
   private JTextField teacherIDTextField, teacherNameTextField, addressTextField,
         workingTypeTextField, employmentStatusTextField, workingHoursTextField,
         departmentTextField, gradedScoreTextField, yearOfExperienceTextField,
         salaryTextField, performanceIndexTextField, academyQualificationTextField,
```

```
        specializationTextField;



   MyFrame() { // constructor
      // setting up the JFrame
      setTitle("Teacher Database Portal ");       // for frame title
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   // for frame close button
behaviour
      setSize(1600, 900);     // for frame size
      setLocation(130, 20);           // frame location on the screen

      GridLayout gridLayout = new GridLayout(3, 4);

      // for positioning elements onto frame with points
      GridBagConstraints gbc = new GridBagConstraints();
      // Grid layout for main frame
      setLayout(gridLayout);




      // For 1st Row of frame
      // For Teacher Input Fields **(Mandatory)

      // Initializing JLabels
      teacherIDLabel = new JLabel("Teacher ID   :");
      teacherNameLabel = new JLabel("Teacher Name   :");
      addressLabel = new JLabel("Address   :");
      workingTypeLabel = new JLabel("Working Type   :");
      employmentStatusLabel = new JLabel("Employment Status   :");
      workingHoursLabel = new JLabel("Working Hours   :");
```

```
teacherIDLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
teacherNameLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
workingTypeLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
addressLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
employmentStatusLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
workingHoursLabel.setFont(new Font("Verdana", Font.PLAIN, 15));

// Initializing JTextFields
teacherIDTextField = new JTextField(10);
teacherNameTextField = new JTextField(20);
addressTextField = new JTextField(15);
workingTypeTextField = new JTextField(10);
employmentStatusTextField = new JTextField(10);
workingHoursTextField = new JTextField(10);

// will store mandatory teacher fields
JPanel teacherPanel = new JPanel();
GridLayout g = new GridLayout(3, 2, 50, 5);
teacherPanel.setLayout(g);

// creating gap around all elements
teacherPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 10, 20));

// Add labels and text fields to the panel
teacherPanel.add(teacherIDLabel);
teacherPanel.add(teacherIDTextField);
teacherPanel.add(teacherNameLabel);
teacherPanel.add(teacherNameTextField);
teacherPanel.add(addressLabel);
teacherPanel.add(addressTextField);
teacherPanel.add(workingTypeLabel);
```

```
        teacherPanel.add(workingTypeTextField);
        teacherPanel.add(employmentStatusLabel);
        teacherPanel.add(employmentStatusTextField);
        teacherPanel.add(workingHoursLabel);
        teacherPanel.add(workingHoursTextField);

        // Adding the panel to the frame and positioning it. (  1st Row   )
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 4;
        this.add(teacherPanel, gbc);









        // For 2nd Row of Frame



        // Creating panel for 2nd row of frame to include instructions,displaytutor and
displaylecturer button and clear field
        JPanel teacherButtonPanel = new JPanel(new BorderLayout());

        // Buttons For Displaying lecturer and tutor details
        JButton displayTutor = new JButton("Display Tutor Details");
```

```
        JButton displayLecturer = new JButton("Display Lecturer Details");

        // clear Button
        JButton clearButton = new JButton("Clear All Input Fields");

        // Changing Fonts And Backgrounds of buttons
        clearButton.setFont(new Font("Consolas", Font.BOLD, 16));
        clearButton.setForeground(new Color(60, 90, 111));
        clearButton.setBackground(new Color(199, 188, 161));
        displayLecturer.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
        displayLecturer.setBackground(new Color(232, 246, 239));
        displayTutor.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
        displayTutor.setForeground(Color.BLACK);
        displayTutor.setBackground(new Color(232, 246, 239));

        // Will Contain different Instruction in the form of JLabels
        JPanel forNote = new JPanel(new GridLayout(10, 1));

        // Instructions to be followed and will be added to forNote panel
        JLabel note = new JLabel(" Instructions : ");
        JLabel instruction1 = new JLabel(
                "    1)   Please fill all the required input fields with valid values only .");
        JLabel instruction2 = new JLabel(
                "    2)   To add Lecturer provide with Teacher Id, Teacher Name, Address,
Working Type, Employment Status, Working Hour, Year of Experience and
Department Name.  ");
        JLabel instruction3 = new JLabel(
                "    3)   To add Tutor provide with Teacher Id, Teacher Name, Address,
Working Type, Employment Status, Working Hour, Salary, Performance Index, ");
        JLabel subinstruction3 = new JLabel(
                "                 Academic Qualification and Specialization accurately.");
```

```
        JLabel instruction4 = new JLabel(
                "    4)  To assign grades provide with Teacher Id, Graded Score, Year of
Experience ( YOE ) and Department in their respective input fields .");
        JLabel instruction5 = new JLabel(
                "    5)  To set salary provide with correct Teacher ID, Salary and
Performance Index in their respective input fields accurately .");
        JLabel instruction6 = new JLabel(
                "    6)  To remove Tutor provide with correct Tutor ID in Teacher ID
Field.");
        JLabel instruction7 = new JLabel(
                "    7)  To display Lecturer's or Tutor's details kindly check console window
( Terminal ) . ");

        // Setting Font for the Instruction Portion
        note.setFont(new Font("Consolas", Font.BOLD, 15));
        instruction1.setFont(new Font("Verdana", Font.PLAIN, 13));
        instruction2.setFont(new Font("Verdana", Font.PLAIN, 13));
        instruction3.setFont(new Font("Verdana", Font.PLAIN, 13));
        subinstruction3.setFont(new Font("Verdana", Font.PLAIN, 13));
        instruction4.setFont(new Font("Verdana", Font.PLAIN, 13));
        instruction5.setFont(new Font("Verdana", Font.PLAIN, 13));
        instruction6.setFont(new Font("Verdana", Font.PLAIN, 13));
        instruction7.setFont(new Font("Verdana", Font.PLAIN, 13));

        // adding instruction elements to instruction panel
        forNote.add(note);

        forNote.add(instruction1);

        forNote.add(instruction2);
```

```
        forNote.add(instruction3);

        forNote.add(subinstruction3);

        forNote.add(instruction4);

        forNote.add(instruction5);

        forNote.add(instruction6);

        forNote.add(instruction7);


        // will includes 2 button for displayig tutor and lecturer
        JPanel displayTeachersPanel = new JPanel(new GridLayout(1, 2));


        displayTeachersPanel.add(displayLecturer);
        displayTeachersPanel.add(displayTutor);

        // Adding buttons to the panel
        teacherButtonPanel.add(clearButton, BorderLayout.NORTH);
        teacherButtonPanel.add(displayTeachersPanel, BorderLayout.SOUTH);

        // adding instructions onto the panel
        teacherButtonPanel.add(forNote, BorderLayout.CENTER);

        // Adding the panel to the frame    ( 2nd Row )
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.gridwidth = 4;
```

```
this.add(teacherButtonPanel);









// For 3rd Row of Frame



// For Lecturer Panel

// Initializing JLabels
departmentLabel = new JLabel("Department    :");
gradedScoreLabel = new JLabel("Graded Score   :");
yearOfExperienceLabel = new JLabel("Year Of Experience   :");

departmentLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
gradedScoreLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
yearOfExperienceLabel.setFont(new Font("Verdana", Font.PLAIN, 15));

// Initializing JTextFields
departmentTextField = new JTextField(15);
gradedScoreTextField = new JTextField(10);
yearOfExperienceTextField = new JTextField(10);
```

```
// Buttons for Lecturer functions
JButton addLecturerButton = new JButton("Add Lecturer");
JButton gradeAssignmentButton = new JButton("Grade Assignment");

addLecturerButton.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
addLecturerButton.setBackground(new Color(176, 197, 164));
gradeAssignmentButton.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
gradeAssignmentButton.setBackground(new Color(255, 251, 218));

// Creating lecturerPanel to group different elemets of Lecturer
JPanel lecturerPanel = new JPanel(new GridLayout(3, 2, 50, 5));
lecturerPanel.add(departmentLabel);
lecturerPanel.add(departmentTextField);
lecturerPanel.add(yearOfExperienceLabel);
lecturerPanel.add(yearOfExperienceTextField);
lecturerPanel.add(gradedScoreLabel);
lecturerPanel.add(gradedScoreTextField);

// gap around all elements
lecturerPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 10, 20));

// Creating panel to add add lecturer and gradeassignment button
JPanel buttonPanel = new JPanel(new GridLayout(1, 3, 5, 5));
buttonPanel.add(addLecturerButton);
buttonPanel.add(gradeAssignmentButton);

// Main panel to group button and txtfield of lecturer
JPanel mainLecturerPanel = new JPanel();
mainLecturerPanel.setLayout(new BorderLayout());
mainLecturerPanel.add(lecturerPanel, BorderLayout.CENTER);
```

```
mainLecturerPanel.add(buttonPanel, BorderLayout.SOUTH);



// Adding Tutor Panel

// Initializing JLabels
salaryLabel = new JLabel("Salary   :");
performanceIndexLabel = new JLabel("Performance Index   :");
academyQualificationLabel = new JLabel("Academy Qualification   :");
specializationLabel = new JLabel("Specialization   :");

salaryLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
performanceIndexLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
academyQualificationLabel.setFont(new Font("Verdana", Font.PLAIN, 15));
specializationLabel.setFont(new Font("Verdana", Font.PLAIN, 15));

// Initializing JTextFields
salaryTextField = new JTextField(10);
performanceIndexTextField = new JTextField(10);
academyQualificationTextField = new JTextField(15);
specializationTextField = new JTextField(15);

// Tutor Functionality Buttons
JButton addTutorButton = new JButton("Add Tutor");
JButton removeTutorButton = new JButton("Remove Tutor");
JButton setSalaryButton = new JButton("Set Salary");

// setting fonts and background of the buttons
addTutorButton.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
addTutorButton.setBackground(new Color(176, 197, 164));
```

```
setSalaryButton.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
setSalaryButton.setBackground(new Color(170, 215, 217));
removeTutorButton.setFont(new Font("Trebuchet MS", Font.BOLD, 15));
removeTutorButton.setBackground(new Color(255, 128, 128));

// tutorpanel panel for grouping the buttons,txtfield and labels
JPanel tutorpanel = new JPanel(new GridLayout(4, 2, 50, 5));

// for gap around all elements
tutorpanel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

// adding elements to tutorpanel
tutorpanel.add(salaryLabel);
tutorpanel.add(salaryTextField);
tutorpanel.add(performanceIndexLabel);
tutorpanel.add(performanceIndexTextField);
tutorpanel.add(academyQualificationLabel);
tutorpanel.add(academyQualificationTextField);
tutorpanel.add(specializationLabel);
tutorpanel.add(specializationTextField);

// Button panel with GridLayout
JPanel tutorbuttonPanel = new JPanel(new GridLayout(1, 3, 5, 5));
tutorbuttonPanel.add(addTutorButton);
tutorbuttonPanel.add(removeTutorButton);
tutorbuttonPanel.add(setSalaryButton);

// mainTutorPanel i.e consists of buttons and txtfields
JPanel mainTutorPanel = new JPanel();
mainTutorPanel.setLayout(new BorderLayout());
mainTutorPanel.add(tutorpanel, BorderLayout.CENTER);
```

```
        mainTutorPanel.add(tutorbuttonPanel, BorderLayout.SOUTH);

        // comprises of both lecturer and tutor
        JPanel majorPanel = new JPanel(new GridLayout(1, 2, 50, 20));
        majorPanel.add(mainLecturerPanel);
        majorPanel.add(mainTutorPanel);

        // Adding majorPanel to the frame(  3rd Row   )
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.gridwidth = 4; // let it take all the column space

        this.add(majorPanel, gbc);

        this.pack();    // will help consume extra spacig on the frame

        // changing the default visibility of frame & making frame visible
        setVisible(true);









/*

            <<<  ---------  -----------   -----------   ----------     Action Listeners Portions  -------
--  -----------   -----------  ----------    >>>

 */
```

```
        // for clear button
        clearButton.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                int option = JOptionPane.showConfirmDialog(null, "Are you sure you want
to clear all the fields?",
                "Confirmation", JOptionPane.YES_NO_OPTION);


                    if(option == 0){
                        clearTextField();

                    }
            }
        });

        // for displaymethod
        displayLecturer.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                displayLecturer();
```

```
        }
    });
    // for displaytutorbutton
    displayTutor.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            // displayTutor();
            displayTutor();

        }

    });

    // foraddlecturerbutton
    addLecturerButton.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            Teacher teacher1 = addLecturer();
            if (teacher1 != null) {
                teachers.add(teacher1);
            }
        }
    });

    // for addtutorbutton
    addTutorButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
```

```
            Teacher teacher2 = addTutor();
            if (teacher2 != null) {
                teachers.add(teacher2);
            }
        }

    });




/**
 * ActionListener for the gradeAssignmentButton.
 * Assigns a grade to a Lecturer based on the provided teacher ID, graded score,
department, and years of experience.
 */


        gradeAssignmentButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
              gradeAssignment();

            }
         });
```

```
/**
 * ActionListener for the setSalaryButton.
 * Sets the salary and performance index for a Tutor based on the provided teacher
ID.
 */



 setSalaryButton.addActionListener(new ActionListener() {
          @Override

          public void actionPerformed(ActionEvent e) {
              setSalary();
          }
       });









/**
 * ActionListener for the removeTutorButton.
 * Removes a Tutor from the list based on the provided teacher ID.
 */
       removeTutorButton.addActionListener(new ActionListener() {
```

```
            @Override
            public void actionPerformed(ActionEvent e) {
               removeTutor();
            }

        });
    }




/*

            <<< --------- ----------- ----------- ---------- Important Method & Functions
Section --------- ----------- ----------- ---------- >>>

 */









/**
 * Method to add a new lecturer.
 * Retrieves input values from text fields, validates them, and creates a new Lecturer
object.
```

```
 * Displays appropriate error messages if input is invalid or if a lecturer with the same
ID already exists.
 * Returns the newly created Lecturer object if successful, otherwise returns null.
 */

   public Lecturer addLecturer() {
       // Retrieving input values from text fields and trim leading/trailing whitespace
       String teacherID = teacherIDTextField.getText().trim();
       String teacherName = teacherNameTextField.getText().trim();
       String address = addressTextField.getText().trim();
       String workingType = workingTypeTextField.getText();
       String employmentStatus = employmentStatusTextField.getText().trim();
       String workingHoursStr = workingHoursTextField.getText().trim();
       String department = departmentTextField.getText().trim();
       String yearOfExperienceStr = yearOfExperienceTextField.getText().trim();

       try{
       if (!isValidInput(teacherID, teacherName, address, workingType,
employmentStatus, workingHoursStr, department,
               yearOfExperienceStr)) {
           JOptionPane.showMessageDialog(null,
                   "Oops! It looks like some fields are empty or contain invalid data.\n"+
                   "Before proceeding, please ensure all required fields are filled
properly.\n\n"+
                   "Note : See instructions for help." );
           return null;
       }

           // Check if teacher with the same ID already exists
           if (findTeacher(Integer.parseInt(teacherID)) == null) {
```

```
            int teacherId = Integer.parseInt(teacherID);
            int workingHours = Integer.parseInt(workingHoursStr);
            int yearOfExperience = Integer.parseInt(yearOfExperienceStr);

            // Check for negative values in input fields
            if (isNegative(teacherId, workingHours, yearOfExperience)) {
                JOptionPane.showMessageDialog(null,
                        "Invalid input!! \nInput Field cannot have negative values.\n");
                return null;
            }
            // Check for numeric values in non-numeric fields

if((containsNumbers(department))||containsNumbers(employmentStatus)||containsNu
mbers(workingType)
                    ||containsNumbers(address)||containsNumbers(teacherName)){
                JOptionPane.showMessageDialog(null,
                        "Invalid input! \nNumeric values are not allowed in name, address,
working type,"+
                            "\nand employment status fields. \nPlease correct your
input.\n\n"+
                            "Note : See instructions for help." );
                return null;
            }

            if(yearOfExperience>100)
            {
                JOptionPane.showMessageDialog(null,"Year Of Experience must be
valid.");
                return null;
            }
```

```
            if(workingHours>168)
            {
                JOptionPane.showMessageDialog(null,"Working Hours must in range
from 1 to 168.");
                return null;
            }
            // Creating new Lecturer object
            Lecturer lecturer = new Lecturer(teacherId, teacherName, address,
workingType, employmentStatus,
                    workingHours, department, yearOfExperience);
             // Displaying success message, clear text fields, and return newly created
Lecturer object
            JOptionPane.showMessageDialog(null, "Lecturer added successfully.");
            clearTextField();
            return lecturer;

        } else {
            // Display error message if teacher with the same ID already exists
            JOptionPane.showMessageDialog(null,
                    "Teacher with ID : " + Integer.parseInt(teacherID) + " already exists.");
        }

      return null;
   } catch (NumberFormatException e) {
         // Display error message for invalid numerical input
      JOptionPane.showMessageDialog(null,
            "Oops! It seems you've entered invalid input. \nPlease provide valid
numerical values for \n"+
            "1) Teacher ID \n"
            +"2) Working Hours \n"
            +"3) Year of Experience");
```

```
        return null;
    } catch (Exception y) {
            // Displaying a standard error message for other exceptions
        JOptionPane.showMessageDialog(null,
                "Invalid input! Tutor was not added .");
        return null;
    }
    }



/**
 * Creates a new Tutor object with the input provided by the user.
 * Validates the input fields and ensures they are not empty or contain invalid data.
 * Displays appropriate error messages if input is invalid or if a teacher with the same
ID already exists.
 * Returns the newly created Tutor object if successfully added, otherwise returns null.
 */


 private Tutor addTutor() {

        // Retrieve input values from text fields
        String teacherID = teacherIDTextField.getText().trim();
        String teacherName = teacherNameTextField.getText().trim();
        String address = addressTextField.getText().trim();
        String workingType = workingTypeTextField.getText();
        String employmentStatus = employmentStatusTextField.getText().trim();
        String workingHoursStr = workingHoursTextField.getText().trim();
        String salaryStr = salaryTextField.getText().trim();
        String performanceIndexStr = performanceIndexTextField.getText().trim();
```

```
        String academyQualification = academyQualificationTextField.getText().trim();
        String specialization = specializationTextField.getText().trim();



            // Check if all input fields are valid i.e thay are not empty string
        if (!isValidInput(teacherID, teacherName, address, workingType,
employmentStatus, workingHoursStr, salaryStr,
                performanceIndexStr, academyQualification, specialization)) {
            // Displaying error message for invalid or empty fields
            JOptionPane.showMessageDialog(null,
                    "Oops! It looks like some fields are empty or contain invalid data. "
                        +"\nPlease fill in all required fields with accurate information.\n\n"+
                        "Note : See instructions for help." );
            return null;
        }
        try {
            // Checking if teacher with the same ID already exists
            if (findTeacher(Integer.parseInt(teacherID)) == null) {

                // Parsing String input values to integaer data types
                int teacherId = Integer.parseInt(teacherID);
                int workingHours = Integer.parseInt(workingHoursStr);
                int salary = Integer.parseInt(salaryStr);
                int performanceIndex = Integer.parseInt(performanceIndexStr);

                // Checking if any numeric field is negative
                if (isNegative(teacherId, workingHours, salary, performanceIndex)) {
                    JOptionPane.showMessageDialog(null,
                            "Invalid input! \nInput Field cannot have Negative Values");
                    return null;
                }
```

```
            // Checking if numeric values are present in non-numeric fields

if((containsNumbers(specialization))||containsNumbers(academyQualification)
                ||containsNumbers(employmentStatus)||containsNumbers(workingType)
                ||containsNumbers(address)||containsNumbers(teacherName)){

                JOptionPane.showMessageDialog(null,
                   "Invalid input! \nNumeric values are not allowed in name, address,
working type,"+
                   "\nemployment status, specialization and academic qualificationfields.
"
                   +"\nPlease correct your input.\n\n"+
                   "Note : See instructions for help." );
                return null;
            }

            if(performanceIndex>10)
            {
                JOptionPane.showMessageDialog(null,"Performance Index value ranges
from 1 to 10.");
                return null;
            }

            if(workingHours>168)
            {
                JOptionPane.showMessageDialog(null,"Working Hours value ranges
from 1 to 168.");
                return null;
            }
```

```
            // Creating new Tutor object
            Tutor tutor = new Tutor(teacherId, teacherName, address, workingType,
employmentStatus, workingHours,
                    salary, specialization, academyQualification, performanceIndex);



            // Displaing success message
            JOptionPane.showMessageDialog(null, "Tutor added successfully!");
            clearTextField(); // clearing TextFields after adding tutors
            return tutor;

        } else {
            // Displaying error message if teacher with the same ID already exists
            JOptionPane.showMessageDialog(null,
                    "Teacher with ID : " + Integer.parseInt(teacherID) + " already exists.");
        }
            // Displaying error message for invalid numerical input
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null,
            "Oops! It seems you've entered invalid input. \nPlease provide valid
numerical values for :\n"+
            "1) Teacher ID \n"
            +"2) Working Hours \n"
            +"3) Salary \n"
            +"4) Perforamnce Index");
        return null;
             // Displaying a standard error message for other exceptions
    } catch (Exception y) {
        JOptionPane.showMessageDialog(null,
                "Invalid input! Tutor was not added .");
        return null;
```

```
        }
        return null;
    }



    public void gradeAssignment()
    {
            // Retrieve input values from text fields and trim leading/trailing whitespace
            String teacherID = teacherIDTextField.getText().trim();
            String gradeStr = gradedScoreTextField.getText().trim();
            String department = departmentTextField.getText().trim();
            String yearStr = yearOfExperienceTextField.getText().trim();

            try {
                // Input validation
                if (!isValidInput(teacherID, gradeStr, yearStr, department)) {
                    JOptionPane.showMessageDialog(null,
                            "\nInvalid input!! \nPlease check your entries for Teacher ID,
GradedScore, \nDepartment and Year of Experience.\n\n"+
                            "Note : See instructions for help." );
                    return;
                }

                int teacherId = Integer.parseInt(teacherID);
                int gradedScore = Integer.parseInt(gradeStr);
                int yearsOfExperience = Integer.parseInt(yearStr);

                // Checking for negative values
                if (isNegative(teacherId, gradedScore, yearsOfExperience)) {
                    JOptionPane.showMessageDialog(null,
```

```
                    "Invalid input!! \nInput Field cannot have negative values.\n");
                return;
            }
            // Check for numeric values in the department field
            if(containsNumbers(department)){
                JOptionPane.showMessageDialog(null,
                    "Invalid input! \nNumeric values are not allowed in the department
field. "+
                        "\nPlease correct your input.");
                return ;
            }
            if(gradedScore>100){
                JOptionPane.showMessageDialog(null,
                    "Invalid input!! \nGraded Score must be in range from 0 to
100.\n");
                return;
            }

            // cheaking identity of lecturer to know if they exists already or not
            Lecturer lecturer = findLecturer(teacherId);
            if (lecturer != null) { // if lecturer exists then
                // Calling grade assignment method from Lecturer class
                String result = lecturer.gradeAssignment(gradedScore, department,
yearsOfExperience);
                if (!result.equals("Teacher Not Eligible For Grade Assignment")) {
                    JOptionPane.showMessageDialog(null,
                        "( " + result + " ) Assigned and Grade assignment successful
From lecturer "
                                + teacherID + "\nTeacher ID : " + teacherId + "\nGraded
Score : "
```

```
                                    + gradedScore + "\nDepartment : " + department +
"\nYears of Experience : "
                                    + yearsOfExperience);
                        clearTextField();   // clearing textfield after gradingAssignment
                    } else {
                        JOptionPane.showMessageDialog(null, result);
                    }

                } else {
                    JOptionPane.showMessageDialog(null, "Lecturer with ID " + teacherID
+ " not found.");
                }
            } catch (NumberFormatException g) {
                // Display error message for invalid numerical input
                JOptionPane.showMessageDialog(null,
                        "Oops! It seems you've entered invalid input. \nPlease provide valid
numerical values for :\n"+
                        "1) Teacher ID \n"
                        +"2) Graded Score \n"
                        +"3) Year Of Experience \n"
                       );
            } catch (Exception z) {
                // Displaying standard error message for other exceptions
                JOptionPane.showMessageDialog(null,
                        "Invalid input! \nGrade cannot be assigned.");
            }
    }


    public void removeTutor()
    {
```

```
        // Retrieve teacher ID from text field and trim leading/trailing whitespace
        String teacherID = teacherIDTextField.getText().trim();

        try {
            // check if teacher ID input is empty string or not
            if (!isValidInput(teacherID)) {
                JOptionPane.showMessageDialog(null, "\nInvalid input!! \n" +
                                        "Please check your entries for Teacher ID\n\n"+
                "Note : See instructions for help." );
                return;
            }
            // Parsing teacher ID to integer
            int teacherId = Integer.parseInt(teacherID);

            // Checking for negative values in teacher ID
            if (isNegative(teacherId)) {
                JOptionPane.showMessageDialog(null,
                        "Invalid input!! \nInput Field cannot have negative values.\n");
                return;
            }

            // Asking for confirmation before removing the tutor
            int option = JOptionPane.showConfirmDialog(null, "Are you sure you want to
proceed?",
            "Confirmation", JOptionPane.YES_NO_OPTION);
       if (option == 0) {
            // Finding if Tutor object with given ID already exists or not
            Tutor tutor = (Tutor) findTutor(teacherId);
            if (tutor != null) {
                // if it yes then do following
```

```
                String removeornot = tutor.removetutor(); // Calling the function to set all
of its instances to default
                if(!removeornot.equals("Tutor is cerified & cannot be removed")){
                teachers.remove(tutor); // Removing the tutor object from the arrayList
                JOptionPane.showMessageDialog(null,
                        "Tutor with ID " + teacherID + " removed successfully.");
                clearTextField();// Clearing text fields after removing tutor
            }else {
            JOptionPane.showMessageDialog(null,removeornot);
            return;
          }
          }
          else {
            JOptionPane.showMessageDialog(null, "Tutor with ID " + teacherID + " not
found.");
          }
      }
      } catch (NumberFormatException h) {
          // Display error message for invalid numerical input
          JOptionPane.showMessageDialog(null,
                "Oops! It seems you've entered invalid input. "+
                  "\nPlease provide valid numerical values for : \n1) Teacher ID\n\n"+
                  "Note : See instructions for help.");
      } catch (Exception y) {
          // Displaying standard error message for other exceptions
          JOptionPane.showMessageDialog(null,
                "Invalid input! Tutor was not removed." + y);
      }

   }
```

```
    public void setSalary()
    {
            // Retrieve input values from text fields and trim leading/trailing whitespace
            String teacherID = teacherIDTextField.getText().trim();
            String salaryStr = salaryTextField.getText().trim();
            String performanceStr = performanceIndexTextField.getText().trim();

            try {
                // check if teacher ID, salary or performanceindex input is empty string or
not
                if (!isValidInput(teacherID, salaryStr, performanceStr)) {
                    // if the input field is empty string showing suitable message in
information dialog
                    JOptionPane.showMessageDialog(null,
                            "\nInvalid input!! \nPlease check your entries for Teacher ID,\nSalary
and Performance Index.\n\n"+
                            "Note : See instructions for help." );
                    return;
                }

                // Parsing input strings to integers
                int teacherId = Integer.parseInt(teacherID);
                int salary = Integer.parseInt(salaryStr);
                int performanceIndex = Integer.parseInt(performanceStr);



                // Checking for negative values
                if (isNegative(teacherId, salary, performanceIndex)) {
                     // if the input field is a negative value showing suitable message in
information dialdog
```

```
                JOptionPane.showMessageDialog(null,
                        "Invalid input!! \nInput Field cannot have negative values.\n");
                return;
            }

            // Finding tutor to know if they exists already or not
            Tutor tutor = findTutor(teacherId);
            if (tutor != null) {
                // Calling set salary method and storing the returned boolean value onto
a variable
                boolean isIncreased = tutor.setsalary(salary, performanceIndex);



                // if salary was set then show new salary and performance index in a
dialog box
                if (isIncreased == true) {
                    int appraisal = (int)tutor.getsalary() - salary;
                    JOptionPane.showMessageDialog(null,
                            "Salary and performance index updated for Tutor ID : " +
teacherID
                                    + "\nNew Salary : "
                                    + (int)tutor.getsalary() + "\nUpdated Performance Index : " +
performanceIndex+
                                    "\nBonus Amount : "+appraisal);
                    clearTextField();       // clearing textfields after setting salary
                } else {
                    // if salary was not set then displaying suitable information dialog
                    JOptionPane.showMessageDialog(null, "Requirements not met for
salary increment.");


                }
```

```
            } else {
                JOptionPane.showMessageDialog(null, "Tutor with ID " + teacherID + "
not found.");
            }
        } catch (NumberFormatException f) {
            // Display error message for invalid numerical input
            JOptionPane.showMessageDialog(null,
                    "Oops! It seems you've entered invalid input. \nPlease provide valid
numerical values for  \n"+
                     "1) Teacher ID \n"
                    +"2) Salary \n"
                    +"3) Perforamnce Index");
        } catch (Exception y) {
            // Displaying standard error message for other exceptions
            JOptionPane.showMessageDialog(null,
                    "Invalid input! Salary was not set.");
        }
    }









/**
 * Displays details of all lecturers in a new frame.
 * If there are no lecturers to display, shows a message dialog.
 */
public void displayLecturer() {
    int count = 0;
```

```
    // Cheak if there exists any Lecturer type of object in ArrayList
    for (Teacher teacher : teachers) {
        if (teacher instanceof Lecturer) {


            // if any instance of Lecturer found increase the count value and break the
loop
            count++;
            break;
        }
    }
    if (count <= 0) {
        // If no lecturers found, display a message dialog
        JOptionPane.showMessageDialog(null, "There are currently no Lecturers to
display.");
        return;
    } else {

        // Creating a new frame to display lecturer details
        JFrame displayLecturerFrame = new JFrame("Lecturer Details");

displayLecturerFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        displayLecturerFrame.setSize(940, 280);
        displayLecturerFrame.setLocationRelativeTo(null); // Center the frame on the
screen

        JPanel panel = new JPanel(new BorderLayout());
        displayLecturerFrame.add(panel);
```

```
        String[] columnNames = { "ID", "Lecturer Name", "Address", "Working Type",
"Employment Status",
                "Working Hour", "Department", "Year Of Experience" };
        String[][] rowData = new String[teachers.size()][columnNames.length];

        int rowIndex = 0;
        // Iterate over the teachers list to populate the table data
        for (Teacher teacher : teachers) {
            if (teacher instanceof Lecturer) {
                Lecturer lecturer = (Lecturer) teacher;
                // Populate rowData with lecturer details
                rowData[rowIndex][0] = String.valueOf(lecturer.getteacher_id());
                rowData[rowIndex][1] = lecturer.getname();
                rowData[rowIndex][2] = lecturer.getaddress();
                rowData[rowIndex][3] = lecturer.getworking_type();
                rowData[rowIndex][4] = lecturer.getemployment_status();
                rowData[rowIndex][5] = String.valueOf(lecturer.getworking_hours());
                rowData[rowIndex][6] = String.valueOf(lecturer.getdepartment());
                rowData[rowIndex][7] = String.valueOf(lecturer.getyearOfExperience());
                rowIndex++;
            }
        }
        // Creating a JTable with the rowData and columnNames
        JTable table = new JTable(rowData, columnNames);
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF); // Disable auto-resizing
of columns

        // Set preferred column widths for each column
        table.getColumnModel().getColumn(0).setPreferredWidth(50); // ID
        table.getColumnModel().getColumn(1).setPreferredWidth(125); // Tutor Name
        table.getColumnModel().getColumn(2).setPreferredWidth(160); // Address
```

```
        table.getColumnModel().getColumn(3).setPreferredWidth(110); // Working Type
        table.getColumnModel().getColumn(4).setPreferredWidth(130); // Employment
Status
        table.getColumnModel().getColumn(5).setPreferredWidth(100); // Working Hour
        table.getColumnModel().getColumn(6).setPreferredWidth(140); // Department
        table.getColumnModel().getColumn(7).setPreferredWidth(115); // Year Of
Experience

         // Create a JScrollPane to contain the table
        JScrollPane scrollPane = new JScrollPane(table);
        panel.add(scrollPane, BorderLayout.CENTER);

        // setting the visible property to true if display Lecturer button is pressed
        displayLecturerFrame.setVisible(true);
    }
}









/**
* Displays details of all tutors in a new frame.
* If there are no tutors to display, shows a message dialog.
*/

public void displayTutor() {
```

```
    // will count the number of Tutor in the ArrayList
    int count = 0;  // starting from zero

    // iterate over all the ArrayList element & check if there exists any Tutor type of
object in ArrayList
    for (Teacher teacher : teachers) {
        if (teacher instanceof Tutor) {

            // if any instance of Tutor found increase the count value and break the loop
            count++;
            break;
        }
    }
    if (count <= 0) {   // check if there exists atleast one object of tutor in the list or not
        // if there are not even 1 object of tutor in the list display appropriate message
and return
        JOptionPane.showMessageDialog(null, "There are currently no Tutors to
display.");
        return;
    } else {            // if there are objects of tutor in the list do the following things

        // Creating a new frame to display Tutor details
        JFrame displayTutorFrame = new JFrame("Tutor Details");
        displayTutorFrame.setTitle("Tutor Details");
        displayTutorFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        displayTutorFrame.setSize(1135, 280);
        displayTutorFrame.setLocationRelativeTo(null);

        // Creating a panel for the frame and adding it onto the frame
        JPanel panel = new JPanel(new BorderLayout());
```

```
        displayTutorFrame.add(panel);

        // Creating an array for holding column names
        String[] columnNames = { "ID", "Tutor Name", "Address", "Working Type",
"Employment Status",
                "Working Hour", "Salary", "Specialization", "Academic Qualification",
"Performance Index" };

        // Creating a two-dimensional array to hold the data for each teacher in the table
        String[][] rowData = new String[teachers.size()][columnNames.length];

        int rowIndex = 0;
        for (Teacher teacher : teachers) {
            if (teacher instanceof Tutor) {
                Tutor tutor = (Tutor) teacher;
                // Extracting data from each Tutor object and populating the table row by
row
                rowData[rowIndex][0] = String.valueOf(tutor.getteacher_id());
                rowData[rowIndex][1] = tutor.getname();
                rowData[rowIndex][2] = tutor.getaddress();
                rowData[rowIndex][3] = tutor.getworking_type();
                rowData[rowIndex][4] = tutor.getemployment_status();
                rowData[rowIndex][5] = String.valueOf(tutor.getworking_hours());
                rowData[rowIndex][6] = String.valueOf((int) tutor.getsalary());
                rowData[rowIndex][7] = tutor.getspecialization();
                rowData[rowIndex][8] = tutor.getacademic_qualifications();
                rowData[rowIndex][9] = String.valueOf(tutor.getperformance_index());
                rowIndex++;
            }
        }
```

```
        // Initialized the JTable variable which will contain all the values of the Tutor
        JTable table = new JTable(rowData, columnNames);
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF); // Disabled auto-
resizing of columns

        // Set preferred column widths for each column
        table.getColumnModel().getColumn(0).setPreferredWidth(45); // ID
        table.getColumnModel().getColumn(1).setPreferredWidth(120); // Tutor Name
        table.getColumnModel().getColumn(2).setPreferredWidth(160); // Address
        table.getColumnModel().getColumn(3).setPreferredWidth(110); // Working Type
        table.getColumnModel().getColumn(4).setPreferredWidth(120); // Employment
Status
        table.getColumnModel().getColumn(5).setPreferredWidth(100); // Working Hour
        table.getColumnModel().getColumn(6).setPreferredWidth(80); // Salary
        table.getColumnModel().getColumn(7).setPreferredWidth(130); // Specialization
        table.getColumnModel().getColumn(8).setPreferredWidth(140); // Academic
Qualification
        table.getColumnModel().getColumn(9).setPreferredWidth(120); // Performance
Index

        JScrollPane scrollPane = new JScrollPane(table);    // initialized a JScrollpane to
contain the table
        panel.add(scrollPane, BorderLayout.CENTER);         // adding scrollPane to the
panel of new Popup display Tutor frame

        // if display tutor button is pressed set the visible property of the frame to true
and display the frame
        displayTutorFrame.setVisible(true);
    }

}
```

```
/**
 * Clears all text fields in the GUI.
 */
public void clearTextField() {



    // Clear each text field in frame
    teacherIDTextField.setText("");
    teacherNameTextField.setText("");
    addressTextField.setText("");
    workingTypeTextField.setText("");
    employmentStatusTextField.setText("");
    workingHoursTextField.setText("");
    departmentTextField.setText("");
    gradedScoreTextField.setText("");
    yearOfExperienceTextField.setText("");
    salaryTextField.setText("");
    performanceIndexTextField.setText("");
    academyQualificationTextField.setText("");
    specializationTextField.setText("");

}
```

```
/*

            <<< ---------  -----------   -----------   ----------    Simple Validation Method &
Functions Section  ---------  -----------   -----------  ----------    >>>

 */









/**
 * Checks if all given strings are not empty using Method OverLoading.
 *  returns true if all strings are not empty, otherwise false.
 */

   private boolean isValidInput(String a) {
       if (a.equals("")) {
```

```
            return false;
        }
        return true;
    }

    private boolean isValidInput(String a, String b, String c) {
        if ((a.equals("")) || (b.equals("")) || (c.equals(""))) {
            return false;
        }
        return true;
    }

    private boolean isValidInput(String a, String b, String c, String d) {
        if ((a.equals("")) || (b.equals("")) || (c.equals("")) || (d.equals(""))) {
            return false;
        }
        return true;
    }

    private boolean isValidInput(String a, String b, String c, String d, String e, String f,
String g, String h) {
        if ((a.equals("")) || (b.equals("")) || (c.equals("")) || (d.equals("")) || (e.equals("")) ||
(f.equals(""))
                || (g.equals("")) || (h.equals(""))) {
            return false;
        }
        return true;
    }

    private boolean isValidInput(String a, String b, String c, String d, String e, String f,
String g, String h,
```

```
            String i, String j) {
        if ((a.equals("")) || (b.equals("")) || (c.equals("")) || (d.equals("")) || (e.equals("")) ||
(f.equals(""))
                || (g.equals("")) || (h.equals("")) || (i.equals("")) || (j.equals(""))) {
            return false;
        }
        return true;
    }









    /**
     * Cheak if the input is less than 0
     * if input < 0 return true otherwise, return false
     */

    private boolean isNegative(int a) {
        if (a < 0) {
            return true;
```

```
    }
    return false;
}

private boolean isNegative(int a, int b, int c) {
    if ((a < 0) || (b < 0) || (c < 0)) {
        return true;
    }
    return false;
}

private boolean isNegative(int a, int b, int c, int d) {
    if ((a < 0) || (b < 0) || (c < 0) || (d < 0)) {
        return true;
    }
    return false;
}
```

```
 /**
 * Checks if a teacher with the same ID already exists in the ArrayList.
 * Iterates through each teacher's ID and compares it with the given ID.
 * Returns true if a teacher with the same ID is found, otherwise false.
 */

   // for Teacher as a whole
   private Teacher findTeacher(int teacherId) {
       for (Teacher teacher : teachers) {  // iterate over each elements present in
teacherlist

           // if its a instance of Teacher class and also the id matches return the object
           if ((teacher instanceof Teacher) && (teacher.getteacher_id() == teacherId)) {
               return teacher;
           }
       }
       return null;
   }



   // for Lecturer
   private Lecturer findLecturer(int teacherId) {
       for (Teacher teacher : teachers) {  // iterate over each elements present in
teacherlist

           // if its a instance of Teacher class and also the id matches return the object of
Teacher type
           if ((teacher instanceof Lecturer) && (teacher.getteacher_id() == teacherId)) {
               return (Lecturer) teacher; // Casting to Lecturer class
           }
```

```
        }
        return null;
    }



    // for Tutor
    private Tutor findTutor(int teacherId) {

        for (Teacher teacher : teachers) {  // iterate over each elements present in
teacherlist

            // if its a instance of Teacher class and also the id matches return the object of
Teacher type
            if ((teacher instanceof Tutor) && (teacher.getteacher_id() == teacherId)) {
                return (Tutor) teacher; // Casting to Tutor class
            }
        }
        return null;
    }









/**
 * Checks if the given string contains any numeric digits.
 * takes input and cheak each character using charat and compare it with number
using isDigit() method
```

```
 * returns true if if contains number, otherwise false
 */
public static boolean containsNumbers(String value) {

    // Iterating over each character in value
    for (int i = 0; i < value.length(); i++) {
        char character = value.charAt(i);
        // Checking if the character is the digit
        if (Character.isDigit(character)) {
            return true;        // Return true if a digit is found
        }
    }
    // Return false if no digits are found
    return false;
}


}









/*

            <<< --------- ----------- ----------- ---------- Main Method & Object Creation
Section --------- ----------- ----------- ---------- >>>

 */
```

```
public class TeacherGUI extends Teacher {

    public static void main(String[] args) {

        // Creating an object of the frame which will cause the constructor to display the
frame
        MyFrame frame = new MyFrame();

    }
}
```

Table 16 : Code for TeacherGUI Class