CS4001NI Programming

30% Individual Coursework

2023-24 Autumn

Student Name: Evani Raut

London Met ID: 23047473

College ID: NP01NT4A230151

Group: N6

Assignment Due Date: Friday, January 26, 2024

Assignment Submission Date: Friday, January 26, 2024

Table of Contents

Table of Figures

Table of Tables

1.Introduction

- **1.1 About the coursework**

This project is centered around the development of three Java classes – Teacher, Lecturer, Tutor using Object-Oriented Programming principles. Each class represents a specific role with unique attributes and functions, demonstrating how these principles apply in a real-world context. This report covers the entire design process, including class diagrams, pseudocode, detailed method explanations, testing approaches, error handling strategies, and concludes with reflective insights. By thoroughly examining the roles and interactions of each class, the projects showcases proficiency in Java programming and the successful application of Object-Oriented Design principles.

Additionally, it demonstrates practical scenarios like grading assignments and determining salaries based on performance, providing a comprehensive foundation for managing personnel in an educational setting. The coursework showcases effective software design and implementation for handling diverse roles within an educational institution.

- 1.2 Tools used

BlueJ

In my project, I selected BlueJ as the primary development environment for coding. It’s simplicity enhances code readability and fosters a clear understanding of class structures, making it an ideal choice for building and organizing my project’s codebase. BlueJ’s interactive and visual approach to Java development is particularly beneficial for educational purposes and projects centered around Object-Oriented programming.

Microsoft World

For documentation task, I choose Microsoft word as a reliable tool. MS word provides a robust platform for creating and formatting project documentation. Its rich set of features supports the creation of detailed documents, allowing me to articulate project requirements, design specifications, and user manuals with ease.

Draw.io

To visually represent the architecture and relationships between classes in my project, I turned to draw.io. As a web-based diagramming tool, it offers an intuitive interface for creating various diagrams, including class diagrams. It’s flexibility and ease make it valuable asset in conveying the design aspects of my project.

2.Class Diagram

2.1 Class Diagram for class Teacher:

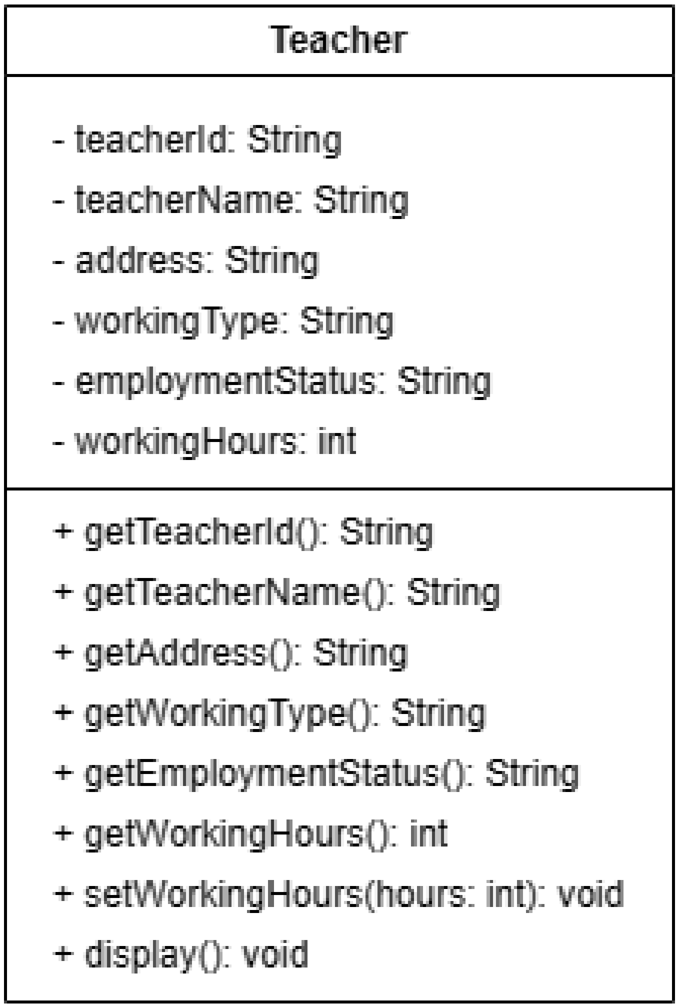


Figure 1: Class Diagram of Teacher class

2.2 Class Diagram for class Lecturer:

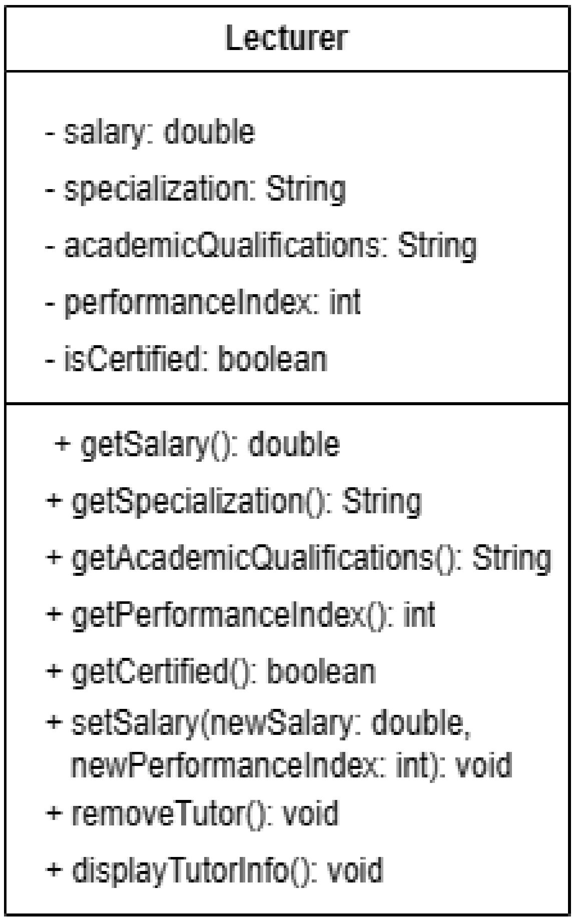

Lecturer
- salary: double
- specialization: String
- academicQualifications: String
- performanceIndex: int
- isCertified: boolean
+ getSalary(): double
+ getSpecialization(): String
+ getAcademicQualifications(): String
+ getPerformanceIndex(): int
+ getCertified(): boolean
+ setSalary(newSalary: double, newPerformanceIndex: int): void
+ removeTutor(): void
+ displayTutorInfo(): void

Figure 2 : Class Diagram of Lecturer class

2.3 Class Diagram for class Tutor:

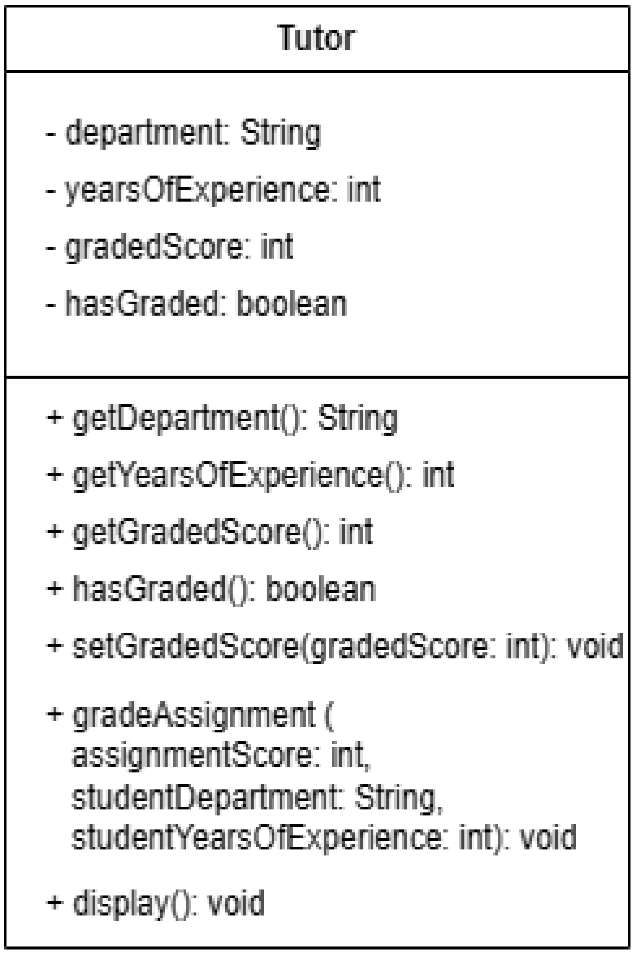

Tutor
- department: String
- yearsOfExperience: int
- gradedScore: int
- hasGraded: boolean
+ getDepartment(): String
+ getYearsOfExperience(): int
+ getGradedScore(): int
+ hasGraded(): boolean
+ setGradedScore(gradedScore: int): void
+ gradeAssignment (assignmentScore: int, studentDepartment: String, studentYearsOfExperience: int): void
+ display(): void

Figure 3: Class Diagram of Tutor class

2. 4 Inheritance Class Diagram:

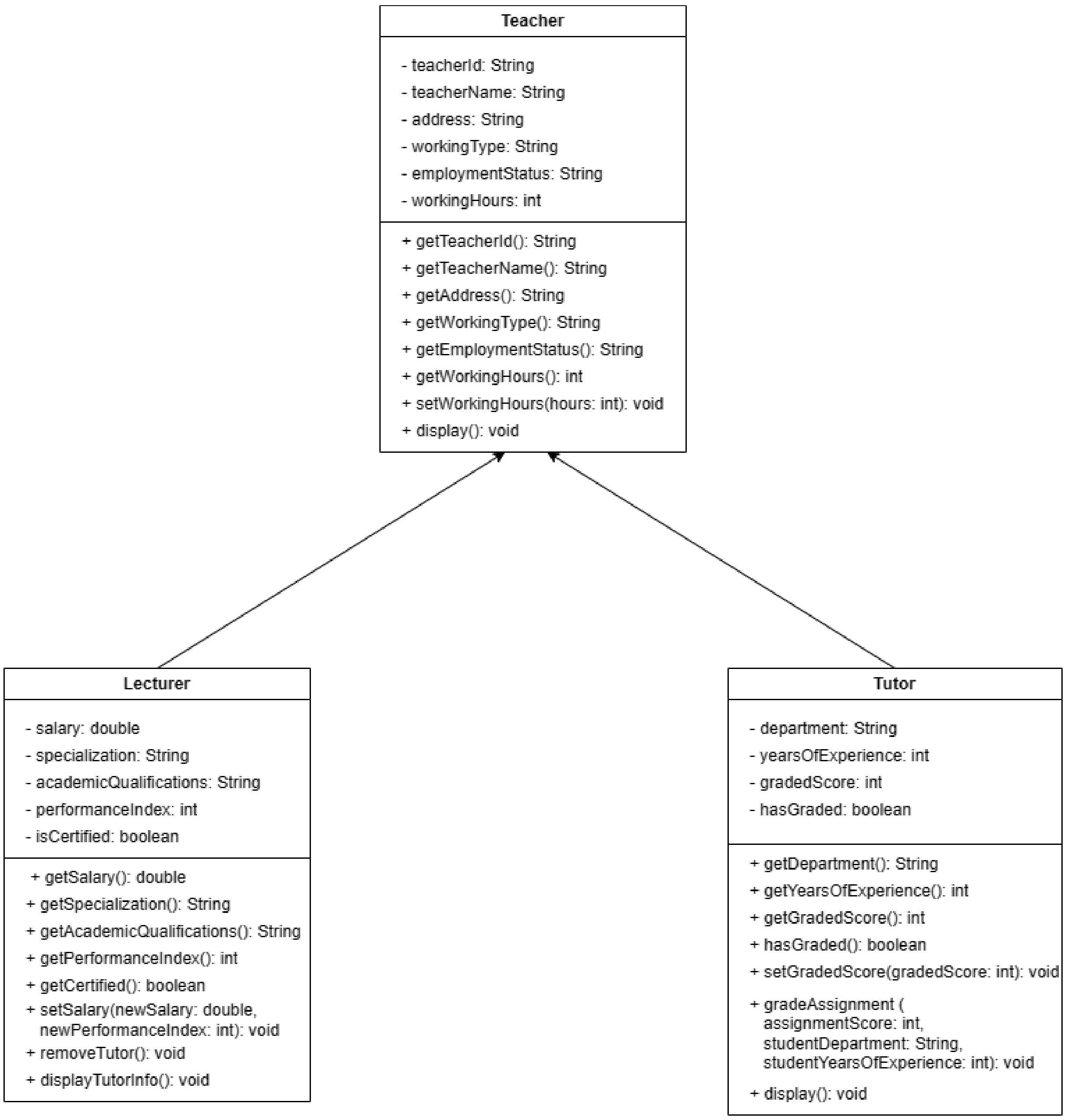

Teacher
- teacherId: String
- teacherName: String
- address: String
- workingType: String
- employmentStatus: String
- workingHours: int
+ getTeacherId(): String
+ getTeacherName(): String
+ getAddress(): String
+ getWorkingType(): String
+ getEmploymentStatus(): String
+ getWorkingHours(): int
+ setWorkingHours(hours: int): void
+ display(): void
Lecturer
- salary: double
- specialization: String
- academicQualifications: String
- performanceIndex: int
- isCertified: boolean
+ getSalary(): double
+ getSpecialization(): String
+ getAcademicQualifications(): String
+ getPerformanceIndex(): int
+ getCertified(): boolean
+ setSalary(newSalary: double, newPerformanceIndex: int): void
+ removeTutor(): void
+ displayTutorInfo(): void
Tutor
- department: String
- yearsOfExperience: int
- gradedScore: int
- hasGraded: boolean
+ getDepartment(): String
+ getYearsOfExperience(): int
+ getGradedScore(): int
+ hasGraded(): boolean
+ setGradedScore(gradedScore: int): void
+ gradeAssignment (assignmentScore: int, studentDepartment: String, studentYearsOfExperience: int): void
+ display(): void

Figure : Inheritance Class Diagram

3.Pseudocode

3.1 Teacher

```
CREATE a class TEACHER:
DO
DECLARE instance variable teacherId as String datatype
DECLARE instance variable teacherName as String datatype
DECLARE instance variable address as String datatype
DECLARE instance variable workingType as String datatype
DECLARE instance variable employmentStatus as String datatype
DECLARE instance variable workingHours as int datatype

CALL Teacher with parameter
DO
ASSIGN teacherId
ASSIGN teacherName
ASSIGN address
ASSIGN workingType
ASSIGN employmentStatus
END DO

CREATE an accessor method getTeacherId with return type String
```

```
DO
RETURN TeacherId
END DO

CREATE an accessor method getTeacherName with return type String
DO
RETURN TeacherName
END DO
CREATE an accessor method getAddress with return type String
DO
RETURN Address
END DO

CREATE an accessor method getWorkingType with return type String
DO
RETURN WorkingType
END DO

CREATE an accessor method getEmploymentStatus with return type String
DO
RETURN EmploymentStatus
```

```
END DO

CREATE an accessor method getWorkingHours with return type
int
DO
RETURN WorkingHours
END DO

 CREATE an method to set working hours
DO
    METHOD setWorkingHours(hours: int): void
        workingHours <- hours
END DO

  CALL DISPLAY
DO
    METHOD display(): void
        Print("Teacher Id:" + teacherId)
        Print("Teacher Name:" + teacherName)
        Print("Address:" + address)
        Print("Working Type:" + workingType)
        Print("Employment Status:" + employmentStatus)
        IF workingHours > 0 THEN
            Print("Working Hours:" + workingHours)
END IF
END DO
DO
        ELSE
            Print("Working Hours:Not assigned")
END ELSE
END DO
```

3.2 LECTURER

```
CREATE a class Lecturer extendsTEACHER
DO
DECLARE instance variable department as String datatype
DECLARE instance variable yearsOfExperience as int datatype
DECLARE instance variable gradedScore as int datatype
DECLARE instance variable hasGrade as boolean datatype



CALL Lecturer with parameter
DO
```

```
ASSIGN  super with parameters
ASSIGN super with department
ASSIGN super with yearsOfExperience
ASSIGN super with gradedScore
ASSIGN super with hasGraded
END DO

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

CREATE an accessor method getYearsOfExperience() with
return type int
DO
RETURN yearsOfExperience
END DO


CREATE an accessor method getGradedScore() with return type
int
DO
RETURN getGradedScore
```

```
END DO


CREATE an accessor method hasGraded( )with return type
boolean

DO

RETURN hasGraded

END DO

    CREATE a  mutator method for a graded score

     setGradedScore(gradedScore: int): void

DO

     SET this.gradedScore <- gradedScore

END DO


CREATE method to grade assignmments

gradeAssignment(assignmentScore: int, studentDepartment:
String, studentYearsOfExperience: int): void

DO

        IF NOT hasGraded AND yearsOfExperience >= 5 AND
department EQUALS studentDepartment THEN

            IF assignmentScore >= 70 THEN

                gradedScore <- assignmentScore

            ELSE IF assignmentScore >= 60 THEN

                gradedScore <- assignmentScore

            ELSE IF assignmentScore >= 50 THEN
                gradedScore <- assignmentScore
            ELSE IF assignmentScore >= 40 THEN
                gradedScore <- assignmentScore
            ELSE
                gradedScore <- assignmentScore
            ENDIF
END DO
DO
        hasGraded <- true
        ELSE
            Print "The lecturer has already graded or does not meet
the criteria for grading."
        ENDIF
END DO


    CREATE a method to display details of the Lecturer
   display(): void
    CALL super.display()
   DO
        Print "Department: " + department
        Print "Years of Experience: " + yearsOfExperience

        IF hasGraded THEN
```

```
            Print "Graded Score: " + gradedScore
END IF
END DO
DO
        ELSE
            Print "Graded Score: Not graded yet"
END ELSE
        END DO
```

3.3 TUTOR

```
CREATE a class Tutor extends TEACHER:
DO
DECLARE instance variable  salary as a double datatype
DECLARE instance variable specialization as a String datatype
DECLARE instance variable academicQualifications as a String datatype
DECLARE instance variable performanceIndex as a int datatype
DECLARE instance variable isCertified as a Boolean datatype

CALL Teacher with parameter
DO
ASSIGN salary
ASSIGN specialization
ASSIGN academicQualifications
ASSIGN performanceIndex
ASSIGN isCertified
END DO

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


END DO

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

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

```
CREATE an accessor method getPerformanceIndex() with return
type int

DO

RETURN performanceIndex

END DO


CREATE an accessor method isCertified() with return type
boolean

DO

RETURN isCertified

END DO


   CREATE a method to set salary

       setSalary(salary: double, performanceIndex: int): void

DO

        IF performanceIndex > 5 AND getWorkingHours() > 20
THEN

            double appraisal <- 0

            IF performanceIndex >= 5 AND performanceIndex <= 7
THEN

                appraisal <- 0.05

END IF

END DO

DO
```

```
            ELSE IF performanceIndex >= 8 AND performanceIndex
<= 9 THEN
                appraisal <- 0.1
            ELSE IF performanceIndex EQUALS 10 THEN
                appraisal <- 0.2
            ELSEIF
            double newSalary <- salary + (appraisal * salary)
            this.salary <- newSalary
            isCertified <- true
END ELSEIF
END DO
DO
        ELSE
            Print "Salary cannot be approved yet."
END ELSE
END DO


    CREATE  a method to remove tutor
      removeTutor(): void
    DO
        IF NOT isCertified THEN
            salary <- 0
            specialization <- ""
            academicQualifications <- ""
            performanceIndex <- 0
            isCertified <- false
        ENDIF
END DO

    CREATE display method
DO
       display(): void
END DO

  CALL DISPLAY
DO
        IF isCertified THEN
            Print "Salary: " + salary
            Print "Specialization: " + specialization
            Print "Academic Qualifications: " + academicQualifications
            Print "Performance Index: " + performanceIndex
        ENDIF
END DO
```

4. Description of Methods

4.1 The description of methods of Teacher class:

1. Constructor

- Method Name: ‘Teacher’

- Description: A constructor method for the ‘Teacher’ class takes parameters to initialize the attributes of a teacher object and sets the values of ‘teacherId’, ‘teacherName’, ‘address’, ‘workingType’, and ‘employmentStatus’ based on the provided arguments.

2. Accessor Methods

- Method Name: ‘getTeacherId’, ‘getTeacherName’, ‘getAddress’, ‘getWorkingType’, ‘getEmploymentStatus’, ‘getWorkingHours’

- Description: These accessor methods used to retrieve the values of the respective attributes of a ‘Teacher’ object provide read-only access to the private attributes.

3. Setter Method

- Method Name: ‘setWorkingHours’

- Description: This method allows setting the value of the ‘workingHours’ attribute as it takes an integer parameter (‘hours’) and assigns it to the ‘workingHours’ attribute.

4. Display Method

- Method Name: ‘display’

- Description: This method prints out the details of a ‘Teacher’ object to the console and it includes the teacher’s ID, name, address, working type, employment status, and working hours if assigned. If working hours are not assigned which means it is less than or equal to 0, it displays “Not assigned.”

5. Example Usage Method

- Method Name: ‘main’

- Description: The ‘main’ method serves as an entry point for the program as it creates an instance of the ‘Teacher’ class and sets the working hours using the ‘setWorkingHours’ method and then displays the teacher’s information using the ‘display’ method.

4.2 The description of methods of Lecturer class:

1.Constructor:

- Method Name: ’Lecturer’

- Description: A constructor for the ‘Lecturer’ class takes parameters to initialize attributes for both a teacher and additional attributes specific to a lecturer and it calls the superclass constructor(‘Teacher’) using the ‘super’ keyword and initializes the ‘department’, ‘yearsOfExperience’, ‘gradedScore’, and ‘hasGraded’ attributes.

2. Accessor Methods for Additional Attributes:

- Method Name: ‘getDepartment’, ‘getYearsOfExperience’, ‘getGradedScore’,   ‘hasGraded’

- Description: These accessor methods used to retrieve the values of the additional attributes specific to a lecturer: ‘department’, ‘yearsOfExperience’, ‘gradedScore’, and ‘hasGraded’.

3. Mutator Method for ‘gradedScore’:

- Method Name: ‘setGradedScore’

- Description: This method sets the value of the ‘gradedScore’ attribute based on the provided parameter (‘gradedScore’).

4. Method to Grade Assignments:

- Method Name: ‘gradeAssignment’

- Description: This method grades assignments based on specified criteria as it takes parameters for ‘assignmentScore’, ‘studentDepartment’, and ‘studentYearsOfExperience’. It checks if the lecturer meets the criteria for grading, and if so it assigns a score to ‘gradedScore’ based on certain conditions and it also sets ‘hasGraded’ to true once grading is done.

5. Display Method:

- Method Nmae: ‘ display’

- Description: This method overrides the ‘display’ method in the superclass (‘Teacher’). First it calls the ‘display’ method of the superclass using ‘super.display()’, and then it adds additional information specific to lecturer, such as ‘department’, ‘yearsOfExperience’, and ‘gradedScore’. If grading has not been done then it displays “Not graded yet.”

6. Example Usage Method:

- Method Name: ‘main’

- Description: The ‘main’ method serves as an entry point for the program and it creates an instance of the ‘Teacher’ class (‘t1’) and sets working hours using the ‘setWorkingHours’ method. It also creates an instance of the ‘Lecturer’ class (‘Lecturer’) and demonstrates the usage of methods like ‘gradeAssignment’ and ‘display’ specific to the ‘Lecturer’ class.

4.3The description of methods of Tutor class:

1.Constructor:

- Method Name: ’Tutor’

- Description: A constructor for the ‘Tuto’ class takes parameters to initialize attributes for both a teacher and additional attributes specific to a tutor and it calls the superclass constructor(‘Teacher’) using the ‘super’ keyword and initializes the ‘salary’, ‘specialization’, ‘academicQualifications’, ‘performanceIndex’, and ‘isCertified’ attributes.

2. Accessor Methods:

- Method Name: ‘getSalary’, ‘getSpecialization’, ‘getAcademicQualifications’, ‘getPerformanceIndex’, ‘isCertified’

- Description: These accessor methods used to retrieve the values of the additional attributes specific to a tutor ‘salary’, ‘specialization’, ‘academicQualifications’, ‘performanceIndex’ and ‘isCertified’.

3. Setter Method for salary:

- Method Name: ‘setSalary’

- Description: This method sets the value of the ‘salary’ attribute based on specified conditions. If the ‘performanceIndex’ is greater than 5 and the working hours are more than 20, it calculates a new salary based on an appraisal percentage. The tutor is marked as certified (‘isCertified’) after the salary is approved.

4.Method to Remove Tutor:

- Method Name: ‘removeTutor’

- Description: This method is used to remove a tutor by setting all attributes to default values if the tutor is not certified. If the tutor is certified then no removal takes place.

5. Display Method:

- Method Name: ‘display’

- Description: This method overrides the ‘display’ method in the superclass (‘Teacher’).It first calls the ‘display’ method of the superclass using ‘super.display()’, and then it adds additional information specific to a certified tutor, such as ‘salary’, ‘specialization’, ‘academicQualifications’, and ‘performanceIndex’, If the tutor is not certified, no additional information is displayed.

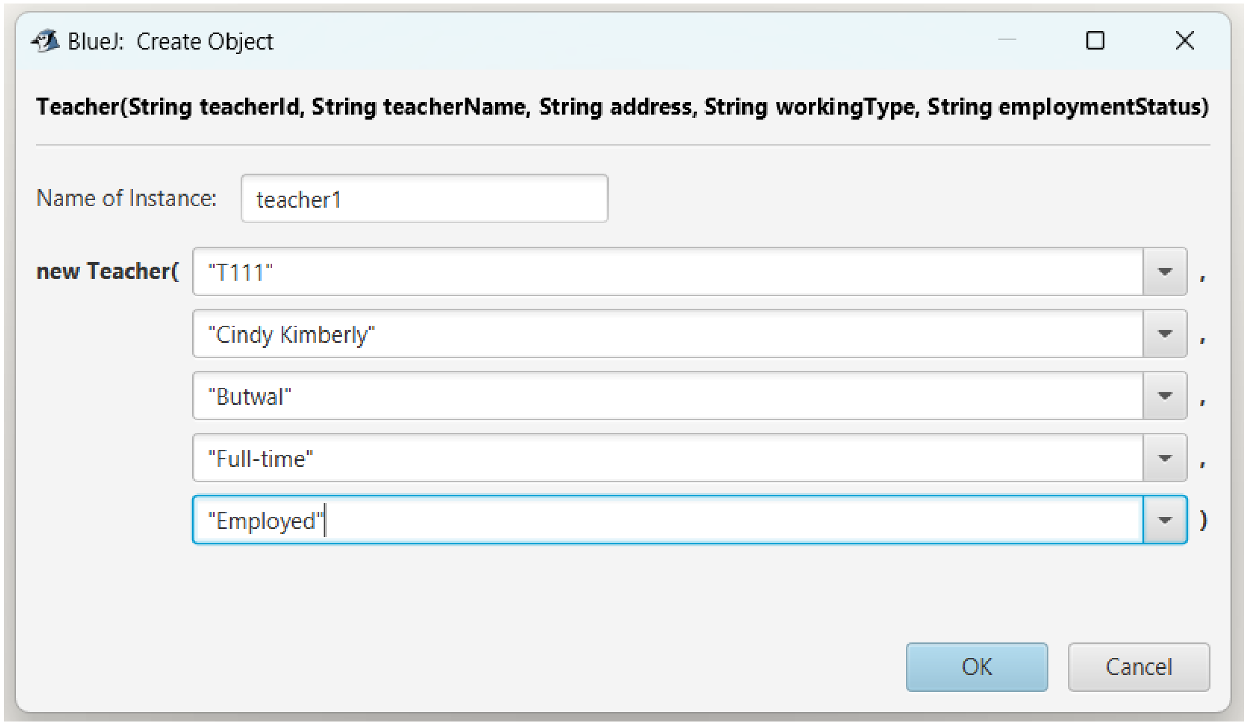
5.Testing

-5.1 Testing 1

Objectives:	To inspect the ‘Teacher’ class, set working hours, and re-inspect the class.
Action:	1. Create an instance of the ‘Teacher’ class with the following arguments: - teacherId = “T111” - teacherName = “Cindy Kimberly” - address = “Butwal” - workingType = “Full-time” - employmentStatus = “Employed” 2. Inspect the details of the ‘Teacher’ class. 3. Set working hours for the teacher: - Call ‘setWorkingHours’ method with ‘hours = 40’. 4. Re-inspect the details of the ‘Teacher’ class.
Expected Result:	The working hours of the teacher are updated to 40, and the class details reflect the changes.
Actual Result:	The working hours of the teacher are updated to 40, and the class details reflect the changes.
Conclusion:	Test is successful.

Table 1: Testing 1

- **Output Result**


BlueJ: Create Object

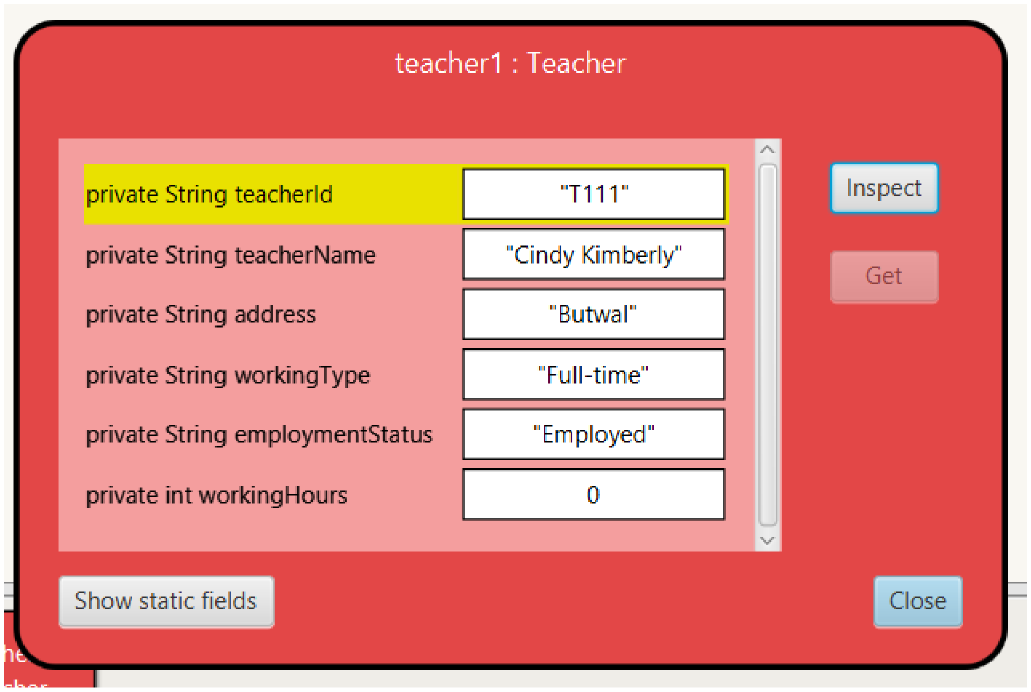
Teacher(String teacherId, String teacherName, String address, String workingType, String employmentStatus)

Name of Instance: teacher1

new Teacher("T111" ,
"Cindy Kimberly" ,
"Butwal" ,
"Full-time" ,
"Employed")

OK Cancel

Figure 5: Screenshot of assigning the value in Teacher class


teacher1 : Teacher

private String teacherId	"T111"
private String teacherName	"Cindy Kimberly"
private String address	"Butwal"
private String workingType	"Full-time"
private String employmentStatus	"Employed"
private int workingHours	0

Inspect
Get
Show static fields
Close

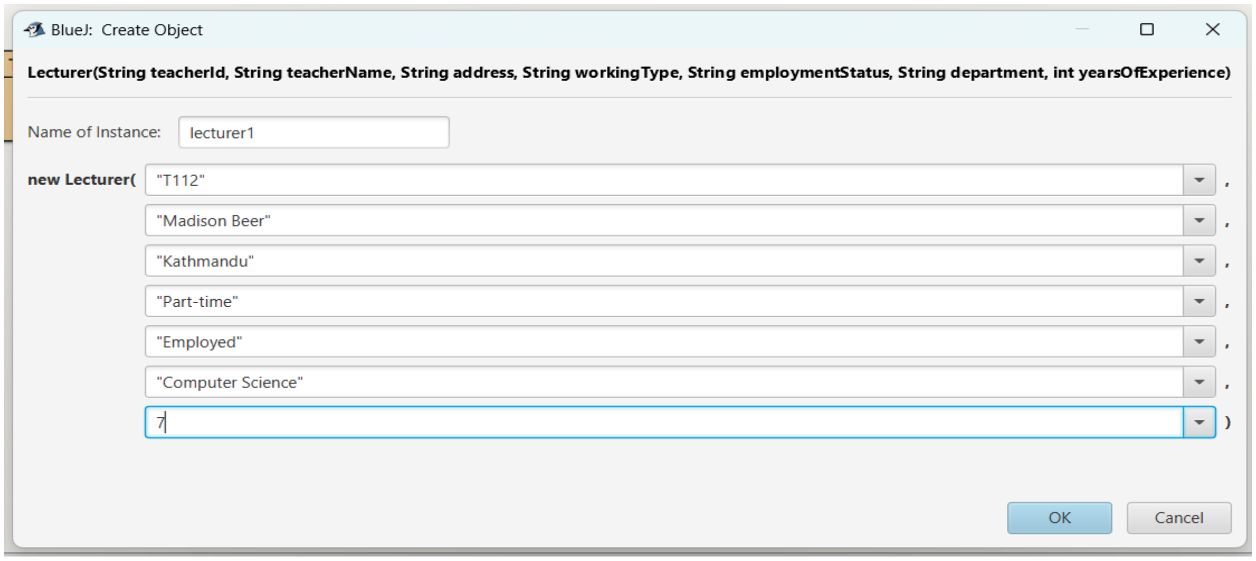
Figure 6: Screenshot for the inspection of Teacher class

- 5.2 Testing 2

Objectives:	To inspect the ‘Lecturer’ class, grade an assignment, and display Lecturer details.
Action:	1. Create an instance of the ‘Lecturer’ with the following arguments: - teacherId = “T112” - teacherName = “Madison Beer” - address = “Kathmandu” - workingType = “Part-time” - employmentStatus = “Employed” - department = ”Computer Science” - yearsOfExperience = 7 2. Inspect the details of the ‘Lecturer’ class. 3. Grade an assignment: - Call ‘greadeAssignment’ method with ‘assignmentScore = 75’, ‘studentDepartment = “Computer Science” ‘, and ‘studentYearsOfExperience = 2’. 4. Display the details of the ‘Lecturer’ class.
Expected Result:	The assignment is graded successfully, and the lecturer details are displayed with the graded score.
Actual Result:	The assignment is graded successfully, and the lecturer details are displayed with the graded score.
Conclusion:	Test is successful.

Table 2: Testing 2

- Output Result

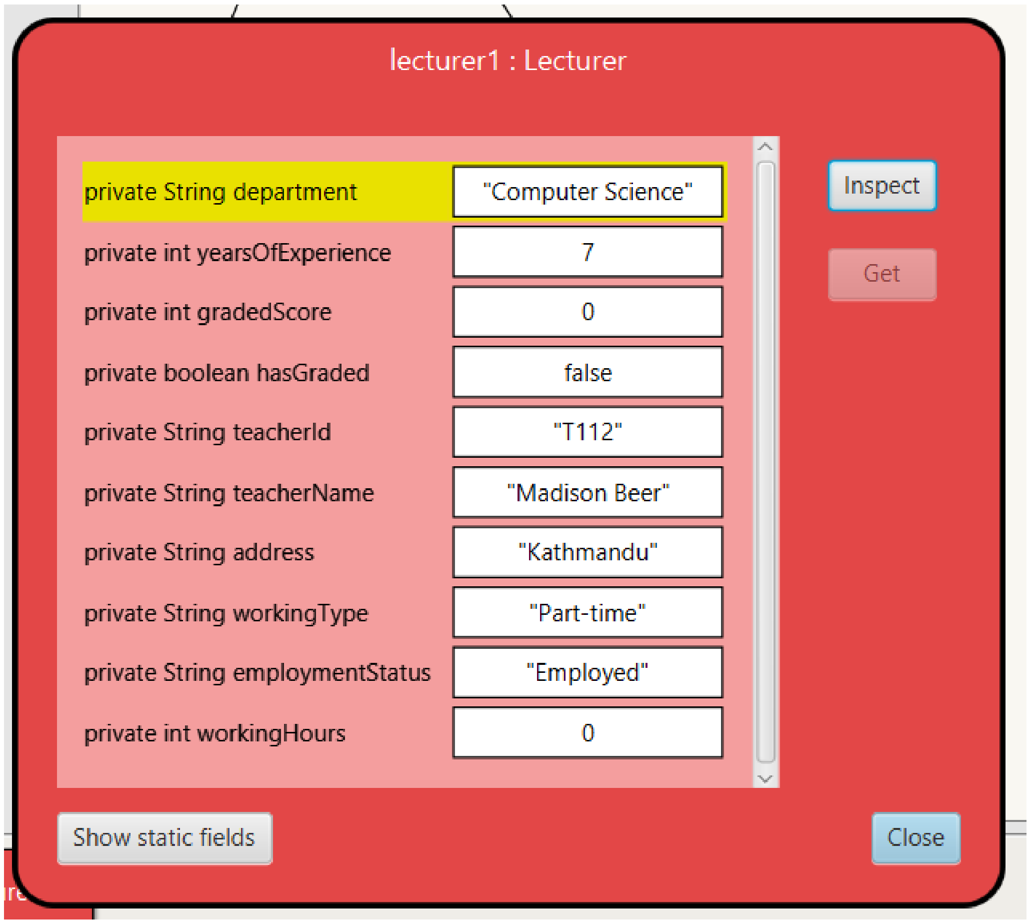

BlueJ: Create Object

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

Name of Instance: lecturer1

new Lecturer("T112" ,
"Madison Beer" ,
"Kathmandu" ,
"Part-time" ,
"Employed" ,
"Computer Science" ,
7)

Figure 7: Screenshot of assigning the value in Lecturer class


lecturer1 : Lecturer

private String department	"Computer Science"
private int yearsOfExperience	7
private int gradedScore	0
private boolean hasGraded	false
private String teacherId	"T112"
private String teacherName	"Madison Beer"
private String address	"Kathmandu"
private String workingType	"Part-time"
private String employmentStatus	"Employed"
private int workingHours	0

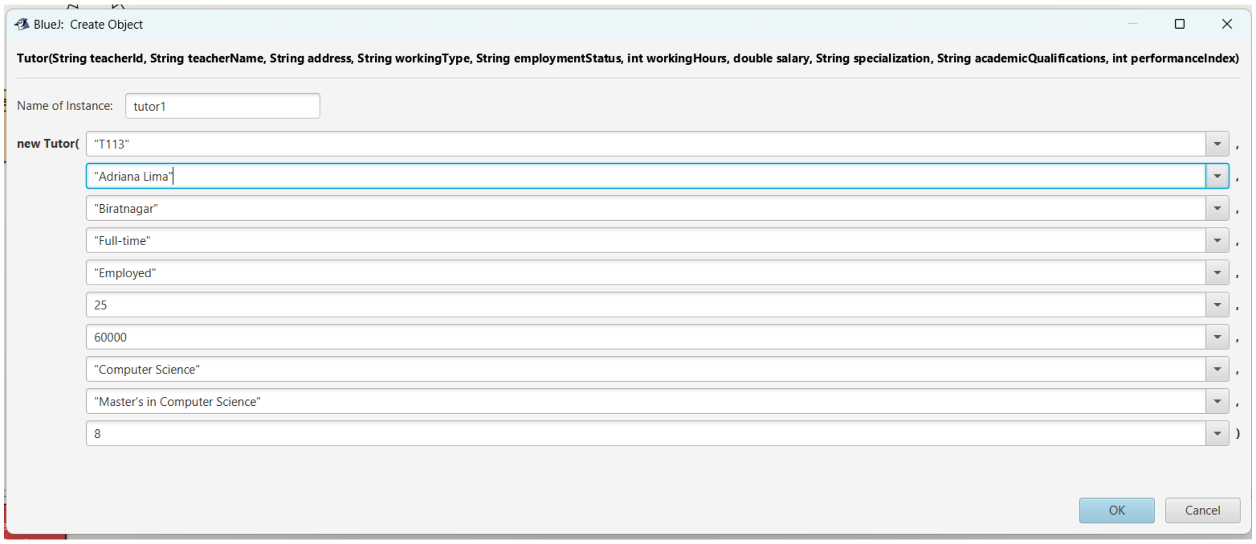
Figure 8: Screenshot for the inspection of Lecturer class

-5.3 Testing 3

Objectives:	To inspect the 'Tutor' class, update the salary, and remove the tutor.
Action:	1. Create an instance of the 'Tutor' claa with the following arguments: - teacherId = "T113" - teacherName = "Adriana Lima" - address = "Biratnagar" - workingType = "Full-time" - employmentStatus = "Employed" - workingHours = 25 - salary = 60000 - specialization = "Computer Science" - academicQualifications = "Master's in Computer Science" - performanceIndex = 8 2. Inspect the details of the 'Tutor' class. 3. Update the salary of the tutor: - Call 'setSalary' method with 'newSalary = 60000' and 'newPerformanceIndex = 8'. 4. Display the details of the 'Tutor' class.
Expected Result:	The tutor's salary is updated, and the removal sets attributes to default values. The class reflects the changes.
Actual Result:	The tutor's salary is updated, and the removal sets attributes to default values. The class reflects the changes.
Conclusion:	Test is successful.

Table 3: Testing 3

- Output Result



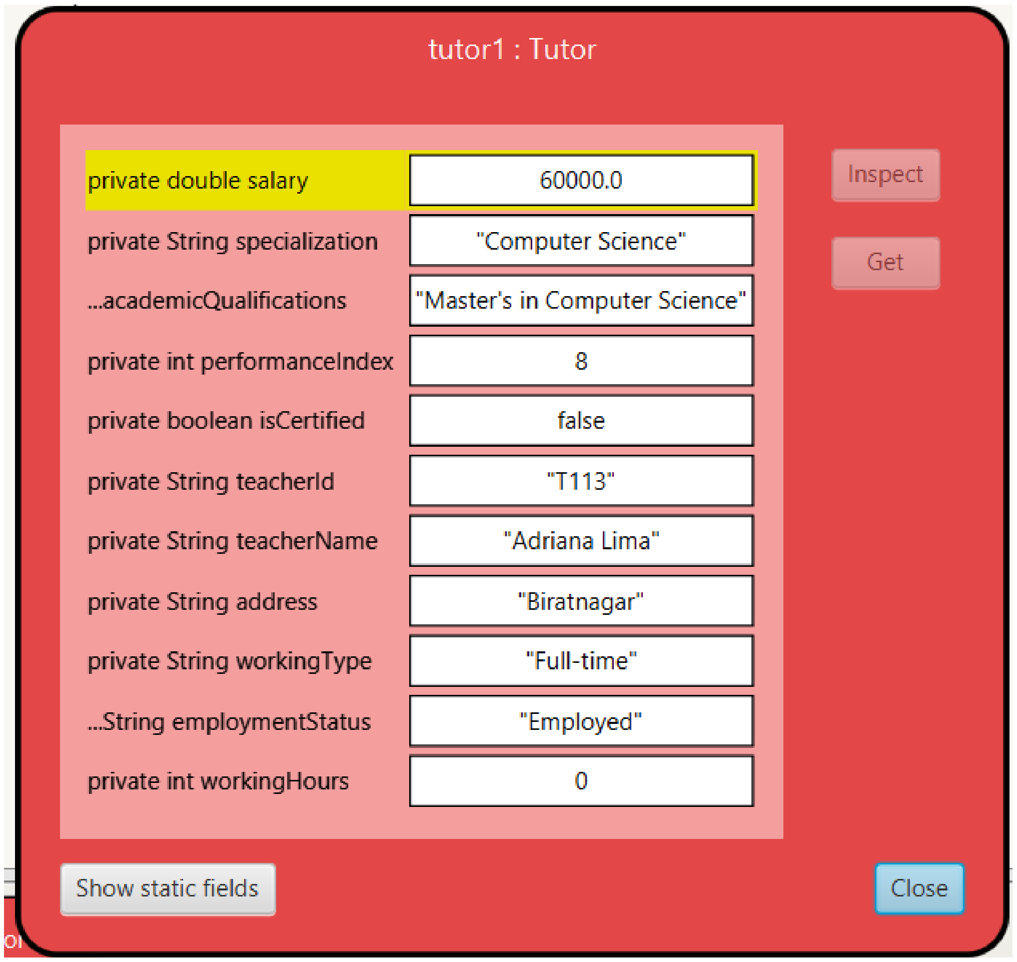
BlueJ: Create Object

Tutor(String teacherId, String teacherName, String address, String workingType, String employmentStatus, int workingHours, double salary, String specialization, String academicQualifications, int performanceIndex)

Name of Instance: tutor1

```
new Tutor( "T113" ,
           "Adriana Lima" ,
           "Biratnagar" ,
           "Full-time" ,
           "Employed" ,
           25 ,
           60000 ,
           "Computer Science" ,
           "Master's in Computer Science" ,
           8 )
```

Figure 9: Screenshot of assigning the value in Tutor class



tutor1 : Tutor

private double salary	60000.0
private String specialization	"Computer Science"
...academicQualifications	"Master's in Computer Science"
private int performanceIndex	8
private boolean isCertified	false
private String teacherId	"T113"
private String teacherName	"Adriana Lima"
private String address	"Biratnagar"
private String workingType	"Full-time"
...String employmentStatus	"Employed"
private int workingHours	0

Figure 10: Screenshot for the inspection of Tutor class

6.Error Detection

Types of error:

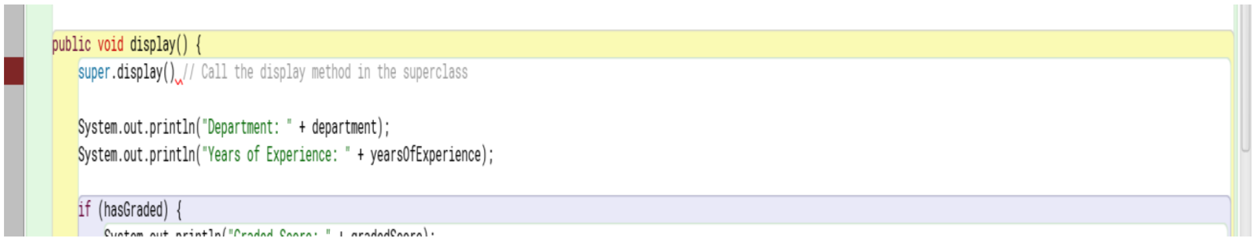
- Syntax Error

- Semantics Error

- Logical Error

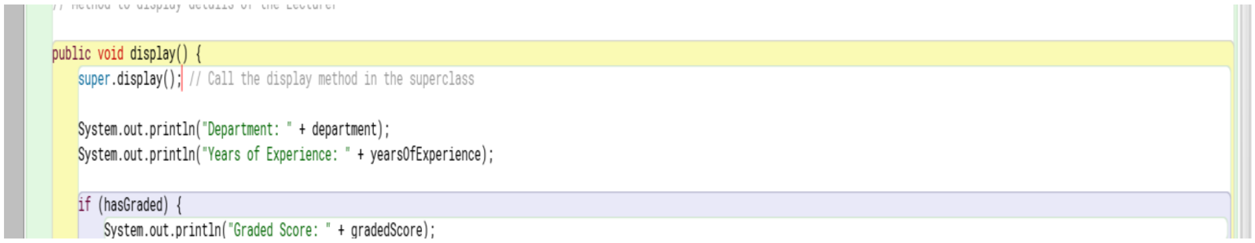
6.1 Syntax Error

A mistake in a program that violates the rules or structure of the programming language which prevents the code from being properly understood or execute is syntax error.



```
public void display() {
    super.display() // Call the display method in the superclass

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

    if (hasGraded) {
        System.out.println("Graded Score: " + gradedScore);
```

Figure 11: Syntax Error

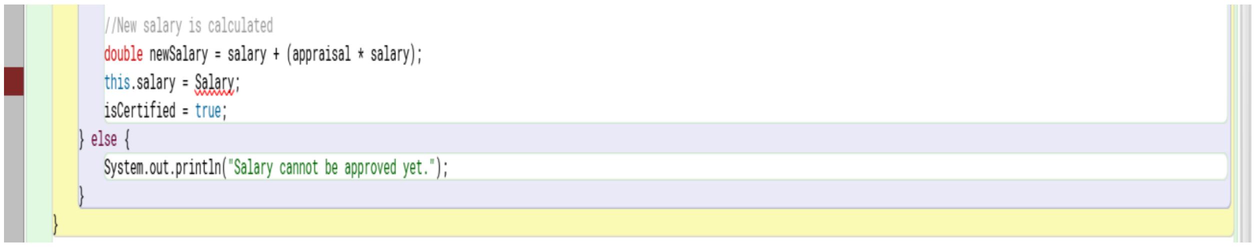


```
// method to display details of the Lecturer

public void display() {
    super.display(); // Call the display method in the superclass

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

    if (hasGraded) {
        System.out.println("Graded Score: " + gradedScore);
```

Figure 12: Correction of Syntax Error

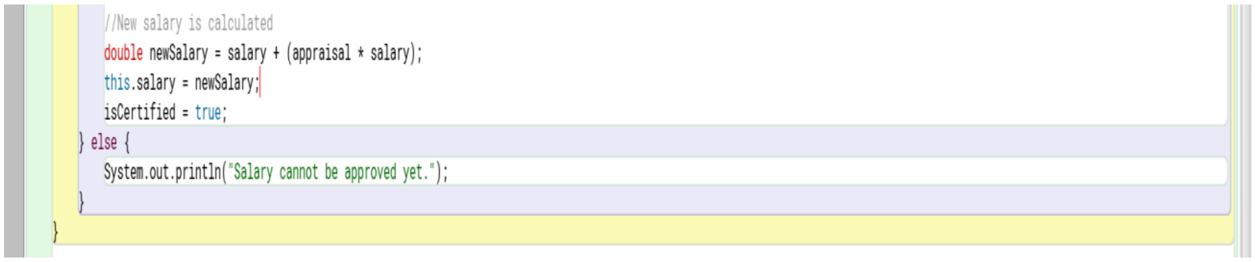
6.2 Semantics Error

Semantics error in a program occurs when code is grammatically correct but conveys a meaning that leads to unintended and incorrect behavior, it doesn’t cause the program to crash but may result in unexpected outputs or logical flaws.



```
        //New salary is calculated
        double newSalary = salary + (appraisal * salary);
        this.salary = Salary;
        isCertified = true;
    } else {
        System.out.println("Salary cannot be approved yet.");
    }
}
```

Figure 13: Semantics Error

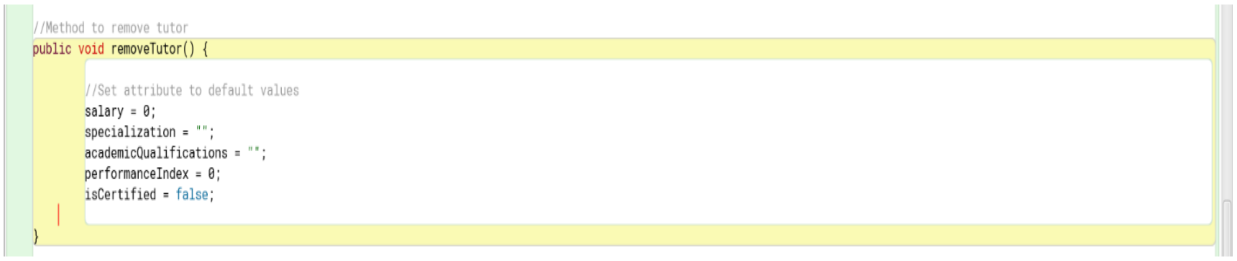


```
        //New salary is calculated
        double newSalary = salary + (appraisal * salary);
        this.salary = newSalary;
        isCertified = true;
    } else {
        System.out.println("Salary cannot be approved yet.");
    }
}
```

Figure 14: Correction of Semantics Error

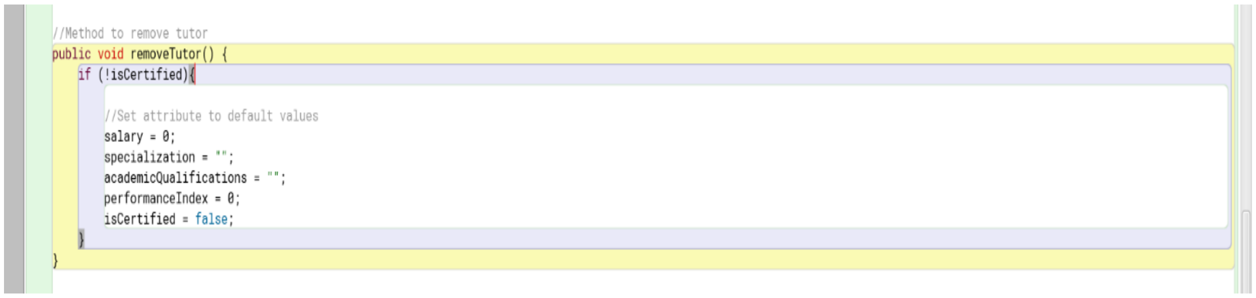
6.3 Logical Error

A logical error in programming is a error that occurs when the code is syntactically correct but the algorithm logic or instructions is flawed which leads to incorrect and unintended results in the output.



```
//Method to remove tutor
public void removeTutor() {

        //Set attribute to default values
        salary = 0;
        specialization = "";
        academicQualifications = "";
        performanceIndex = 0;
        isCertified = false;

}
```

Figure 15: Logical Error



```
//Method to remove tutor
public void removeTutor() {
    if (!isCertified){

        //Set attribute to default values
        salary = 0;
        specialization = "";
        academicQualifications = "";
        performanceIndex = 0;
        isCertified = false;
    }
}
```

Figure 16: Correction of Logical Error

7.Conclusion

This coursework focuses on the development of a program applicable to real-world scenarios in the educational field. It encompasses various tests, error detection, and error correction procedures, providing a comprehensive understanding of the source code.

The code showcases a hierarchy of teacher-related classes with specific attributes and behaviours for different roles like Teacher, Lecturer, Tutor. It demonstrates concepts of inheritance, method overriding, and encapsulation. The ‘main’ methods in each class provide examples of creating instances and using the defined functionality. The code structure supports the modelling of various types of educators with distinct characteristics.

The coursework has proven instrumental in enhancing our knowledge of Java programming, equipping us with practical insights into the implementation of essential programming concepts. It has been a valuable learning experience, covering aspects like the logic behind program outputs, the creation of constructors, and the utilization of various methods.

8.Appendix

8.1 For Teacher class

```
public class Teacher {
    /**Attributes*/
    private String teacherId;
    private String teacherName;
    private String address;
    private String workingType;
    private String employmentStatus;
    private int workingHours;

    /**Constructor*/
    public Teacher(String teacherId, String teacherName, String address,
String workingType, String employmentStatus) {
        //Initializing attributes with provided values
        this.teacherId = teacherId;
        this.teacherName =teacherName;
        this.address = address;
        this.workingType = workingType;
        this.employmentStatus = employmentStatus;
    }

    /**Accessor methods*/
    public String getTeacherId() {
        return teacherId;
```

```
}
 public String getTeacherName() {
    return teacherName;
}
 public String getAddress() {
    return address;
}
 public String getWorkingType() {
    return workingType;
}
 public String getEmploymentStatus() {
    return  employmentStatus;
}
 public int getWorkingHours() {
    return workingHours;
}

/**Method to set working hours*/
public void setWorkingHours(int hours)
{
    workingHours = hours;
}

/**Display method*/
public void display() {
    System.out.println("Teacher Id:" + teacherId);
```

```
        System.out.println("Teacher Name:" + teacherName);
        System.out.println("Address:" + address);
        System.out.println("Working Type:" + workingType);
        System.out.println("Employment Status:" + employmentStatus);

         if(workingHours > 0){
             System.out.println("Working Hours:" + workingHours);
            } else{
        System.out.println("Working Hours:Not assigned");
        }
      }
    /**Example usage*/
    public static void main(String[] args) {
        Teacher t1 = new Teacher("T111", "Cindy Kimberly", "Butwal", "Full-
time", "Employeed");
        t1.setWorkingHours(40);
        t1.display();
    }
}
```

8.2 For Lecturer class

```
public class Lecturer extends Teacher {
    /** Additional attributes for Lecturer*/
    private String department;
    private int yearsOfExperience;
    private int gradedScore;
    private boolean hasGraded;

    /**Constructor*/
    public Lecturer(String teacherId, String teacherName, String address,
String workingType,
                    String employmentStatus, String department, int
yearsOfExperience) {
        /** Call superclass constructor with five parameters*/
        super(teacherId, teacherName, address, workingType,
employmentStatus);

        // Assign additional attributes
        this.department = department;
        this.yearsOfExperience = yearsOfExperience;
        this.gradedScore = 0;
        this.hasGraded = false;
    }

    /**Accessor methods for additional attributes*/
    public String getDepartment() {
        return department;
    }

    public int getYearsOfExperience() {
        return yearsOfExperience;
    }

    public int getGradedScore() {
        return gradedScore;
    }

    public boolean hasGraded() {
        return hasGraded;
    }

    /** Mutator method for gradedScore*/
    public void setGradedScore(int gradedScore) {
        this.gradedScore = gradedScore;
    }

    /** Method to grade assignments*
    public void gradeAssignment(int assignmentScore, String
studentDepartment, int studentYearsOfExperience) {
        if (!hasGraded && yearsOfExperience >= 5 &&
department.equals(studentDepartment)) {
            if (assignmentScore >= 70) {
                gradedScore = assignmentScore;
            } else if (assignmentScore >= 60) {
                gradedScore = assignmentScore;
            } else if (assignmentScore >= 50) {
                gradedScore = assignmentScore;
            } else if (assignmentScore >= 40) {
                gradedScore = assignmentScore;
            } else {
                gradedScore = assignmentScore;
            }

            hasGraded = true;
        } else {
            System.out.println("The lecturer has already graded or does not
meet the criteria for grading.");
        }
    }

    /** Method to display details of the Lecturer*/

    public void display() {
        super.display(); // Call the display method in the superclass

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

        if (hasGraded) {
            System.out.println("Graded Score: " + gradedScore);
        } else {
            System.out.println("Graded Score: Not graded yet");
        }
    }

    /**Example usage*/
    public static void main(String[] args) {
        Teacher t1 = new Teacher("T111", "Cindy Kimberly", "Butwal", "Full-time", "Employeed");
        t1.setWorkingHours(40);
        t1.display();
        Lecturer lecturer = new Lecturer("T112", "Madison Beer", "Kathmandu", "Part-time", "Employeed", "Computer Science", 7);
        lecturer.gradeAssignment(75, "Computer Science", 2);
        lecturer.display();
    }
}
```

8.3 For Tutor class

```
public class Tutor extends Teacher {
    /**Additional attributes*
    private double salary;
    private String specialization;
    private String academicQualifications;
    private int performanceIndex;
    private boolean isCertified;
    /**Constructor*/
    public Tutor(String teacherId, String teacherName, String address, String
workingType, String employmentStatus, int workingHours, double salary,
String specialization,
    String academicQualifications, int performanceIndex) {
        /**Superclass constructor is called*/
        super(teacherId, teacherName, address, workingType,
employmentStatus);

        /**Additional Attributes are initialized*/
        this.salary = salary;
        this.specialization = specialization;
        this.academicQualifications = academicQualifications;
        this.performanceIndex = performanceIndex;
        this.isCertified = false;
    }

    /**Accessor methods*
    public double getSalary() {
        return salary;
    }
    public String getSpecialization() {
        return specialization;
    }
    public String getAcademicQualifications() {
        return academicQualifications;
    }
    public int getPerformanceIndex() {
```

```
    return performanceIndex;
}
public boolean isCertified() {
    return isCertified;
}

/**Method to set salary*/
public void setSalary(double salary, int performanceIndex) {
    if (performanceIndex > 5 && getWorkingHours() > 20) {
        double appraisal = 0;
        if (performanceIndex >= 5 && performanceIndex <= 7) {
            appraisal = 0.05;
        } else if (performanceIndex >= 8 && performanceIndex <= 9) {
            appraisal = 0.1;
        } else if (performanceIndex == 10) {
            appraisal = 0.2;
        }

        /**New salary is calculated*/
        double newSalary = salary + (appraisal * salary);
        this.salary = newSalary;
        isCertified = true;
    } else {
        System.out.println("Salary cannot be approved yet.");
    }
}
```

```
    /**Method to remove tutor*/
    public void removeTutor() {
        if (!isCertified) {
            /**Set attribute to default values*/
            salary = 0;
            specialization = "";
            academicQualifications = "";
            performanceIndex = 0;
            isCertified = false;
        }
    }

    /**Display method*/
    public void display() {
        super.display();
        if (isCertified) {
            System.out.println("Salary: " + salary);
            System.out.println("Specialization: " + specialization);
            System.out.println("Academic Qualifications: " +
academicQualifications);
            System.out.println("Performance Index: " + performanceIndex);
        }
    }
}
```

9.References

1.GreeksforGreeks:

- Website: https://www.geeksforgeeks.org/

- Accessed: 23 January 2024

2.W3Schools:

- Website: https://www.w3schools.com/

- Accessed: 25 January 2024

3.Javatpoint:

- Website: https://www.javatpoint.com/

- Accessed: 25 January 2024

4.Codecademy:

- Website: https://www.codecademy.com/

- Accessed: 26 January 2024