LONDON METROPOLITAN UNIVERSITY

Informatics College Pokhara

informatics college • pokhara

<u>Programming</u>

<u>CS4001NP</u>

Coursework 1

Submitted By:		Submitted To:
Student Name:	Janak Devkota	Mr. Sushil Paudel
London Met ID:	23048806	Module Leader
Group:	CC4057	Programming
Date:	23-Jan-2024	

Table of Contents

Table of Figures

Table of Tables

1. INTRODUCTION

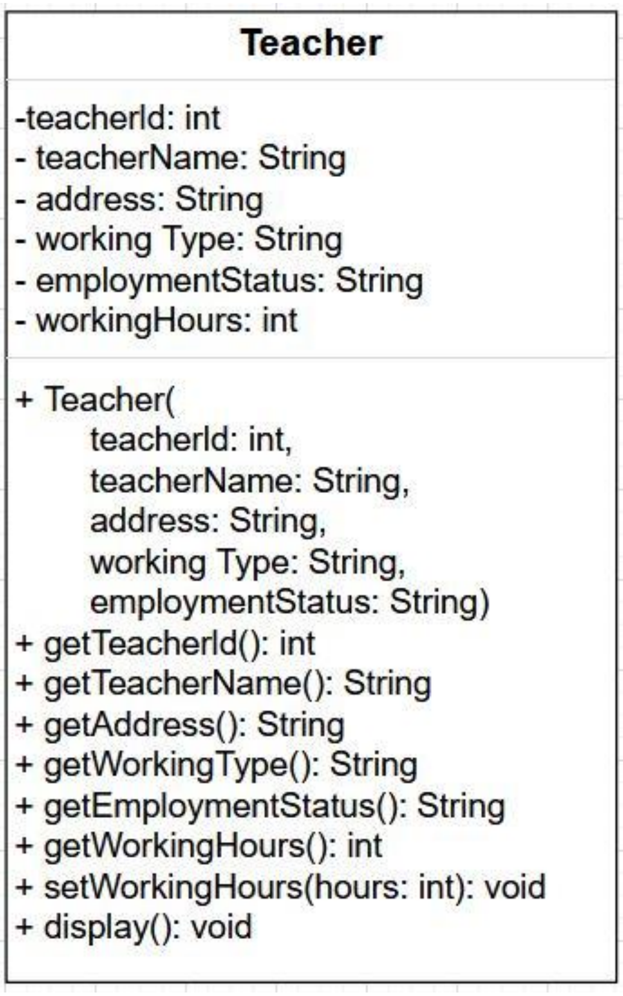
This report is about my first assignment in the Programming class, and it's a really important part of my overall grade—30% important! I decided to use Java for this task because it's known for being friendly to people who are just starting to learn programming. (Kölling, 2004) The assignment was a significant challenge given by the module leader, and it was a big deal for my overall grade. I dove into it, opting for Java as my programming language. The actual coding part was made easier with the help of a tool called BlueJ. It is friendly playground for code. It gave me a nice space to write my code and see how it works. This made the whole process much less scary and more like a fun adventure.

As I started working on the assignment, I began to understand Java better. It's like learning a new language, but instead of words, you use code. Writing and running the code wasn't just about finishing the assignment; it was like solving little puzzles and fixing mistakes. Each mistake taught me something new, like how to make my code better and avoid problems in the future. Java turned out to be full of twists and turns, like an exciting journey. It has its own set of rules and tricks, and I had to figure them out. Despite the challenges, the whole experience was surprisingly enjoyable. It was a mix of difficulty and fun, showing me that learning to code can be both tough and rewarding.

In the end, this report is like telling the tale of my first coding adventure in the Programming class. Choosing Java and using BlueJ made it less intimidating, and the challenges turned into opportunities to learn and grow. (Kölling, 2004) This journey showed me that coding is not just about getting things right; it's about enjoying the process and becoming better with every step.

2. Class Diagram

2.1) Teacher class:



```
Teacher
-----------------------------------------
-teacherId: int
- teacherName: String
- address: String
- working Type: String
- employmentStatus: String
- workingHours: int
-----------------------------------------
+ Teacher(
      teacherId: int,
      teacherName: String,
      address: String,
      working Type: String,
      employmentStatus: String)
+ getTeacherId(): int
+ getTeacherName(): String
+ getAddress(): String
+ getWorkingType(): String
+ getEmploymentStatus(): String
+ getWorkingHours(): int
+ setWorkingHours(hours: int): void
+ display(): void
```

Figure 1: Teacher class

The class diagram for the "Teacher" class defines attributes like teacherId, teacherName, address, workingType, employmentStatus, and workingHours. It includes methods (getters and setters) for accessing and modifying these attributes, facilitating object-oriented management and manipulation of teacher data in a program.

2.2) Lecturer class:

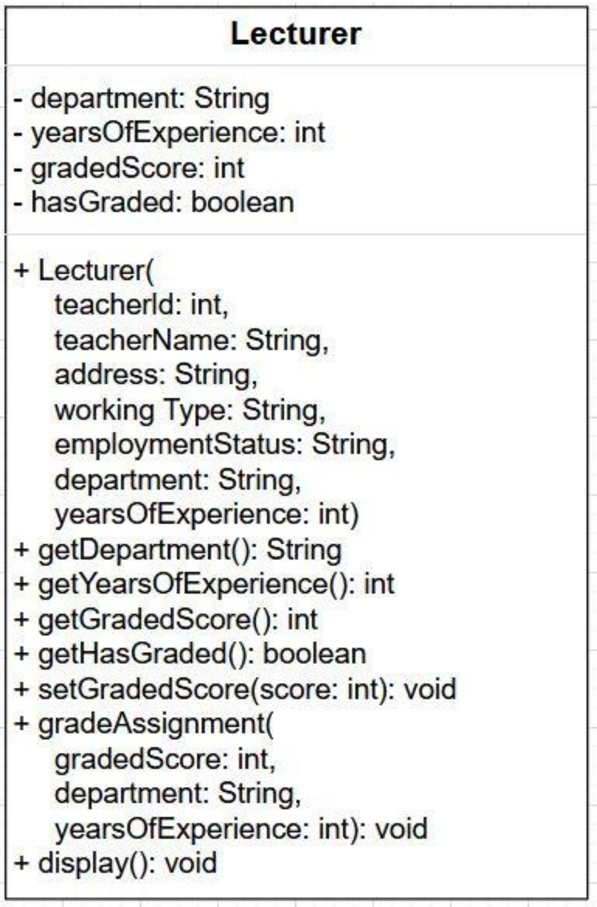
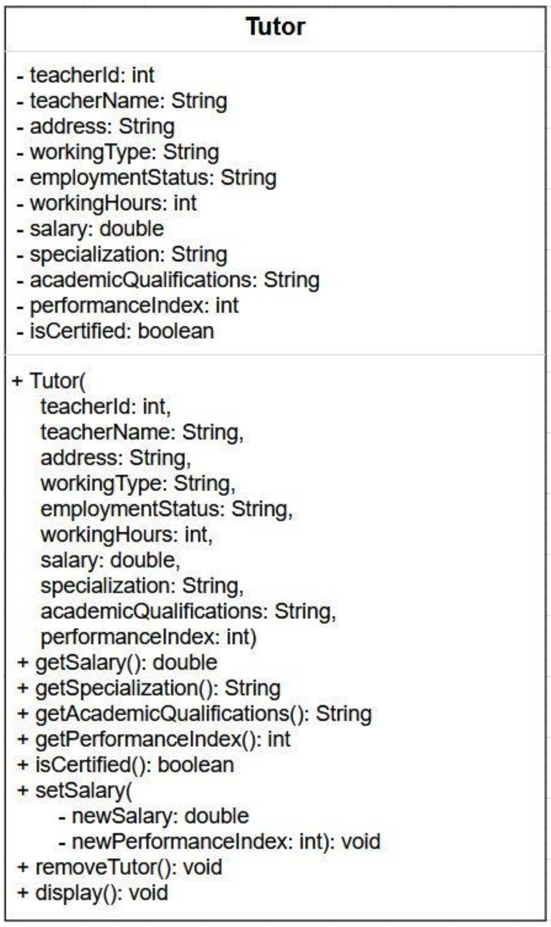


Figure 2: Lecturer Class

The extended "Lecturer" class introduces additional attributes: department, yearsOfExperience, gradedScore, and hasGraded. It includes methods like getDepartment, getYearsOfExperience, getGradedScore, getHasGraded, setGradedScore, gradeAssignment, and display, tailored for lecturers' unique responsibilities. This extension enhances the object-oriented framework, addressing specific lecturer requirements and functionalities.

2.3) Tutor class:



```
Tutor
- teacherId: int
- teacherName: String
- address: String
- workingType: String
- employmentStatus: String
- workingHours: int
- salary: double
- specialization: String
- academicQualifications: String
- performanceIndex: int
- isCertified: boolean

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

Figure 3: Tutor class

The "Tutor" class expands upon "Teacher" and "Lecturer," featuring attributes such as salary, specialization, academic qualifications, performance index, and isCertified. Accompanying methods include getSalary, getSpecialization, getAcademicQualifications, getPerformanceIndex, isCertified, setSalary, removeTutor, and display. This design caters to the intricate demands of managing tutoring personnel within an object-oriented paradigm, addressing real-world complexities while adhering to modularity and reusability principles. The setSalary method facilitates dynamic adjustments based on performance, hours worked, and certification status. The removeTutor method provides flexibility, allowing the removal of tutors when needed. This comprehensive template ensures effective representation and interaction with tutor instances in software systems. Feel free to inquire about specific implementation details or further clarification.

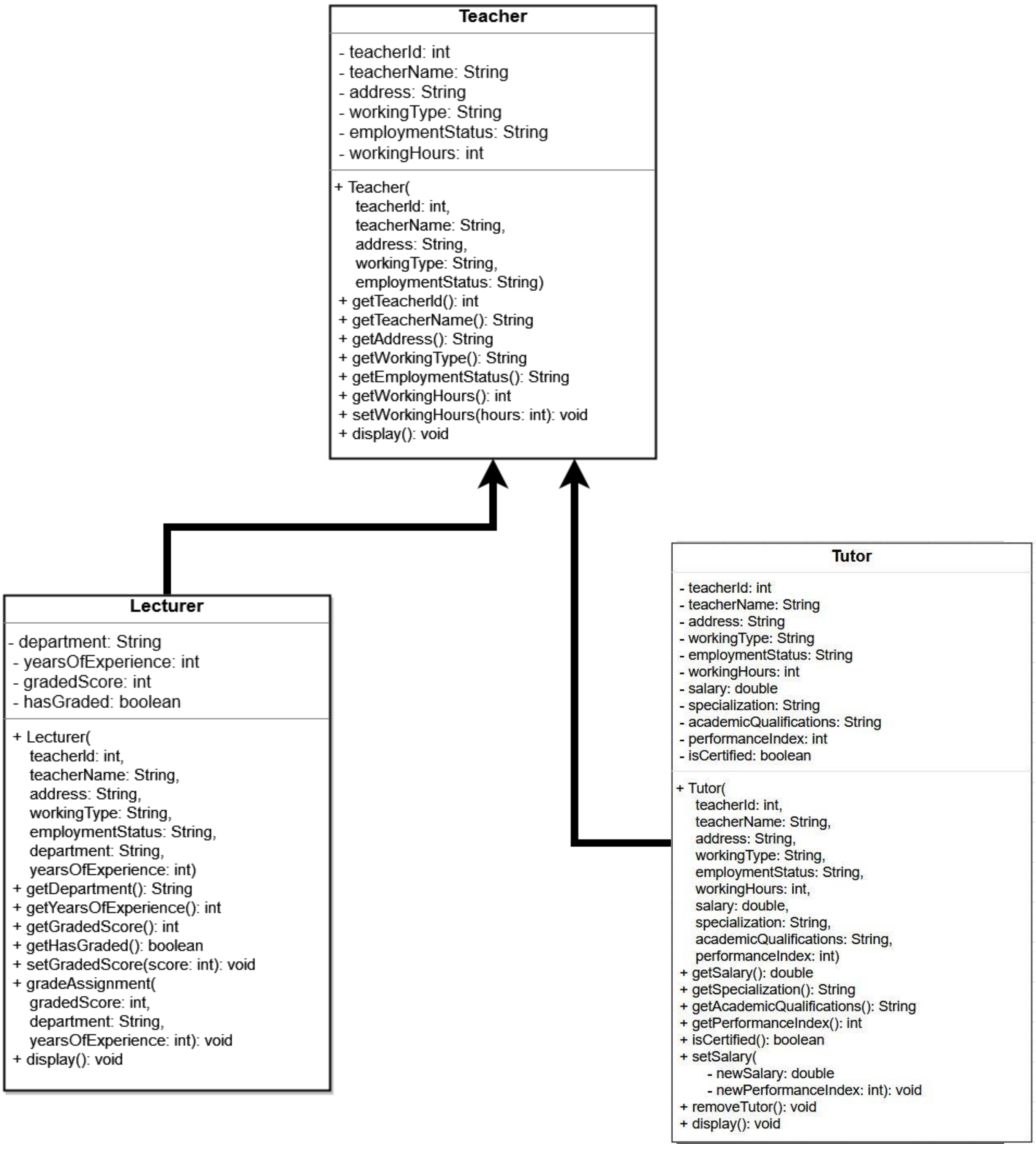


Figure 4: Class Diagram

This program features a hierarchical class structure comprising three classes: Teacher, Lecturer, and Tutor. In this design, Teacher acts as the superclass, while Lecturer and Tutor serve as its subclasses. This arrangement exemplifies hierarchical inheritance, showcasing a logical relationship where both Lecturer and Tutor inherit attributes and behaviour's from the more generalized Teacher class. The class diagram visually represents the relationships and elements within these classes. It illustrates the methods and variables associated with each class, providing a clear overview of their functionalities. Emphasis is placed on the utilization of getter and setter methods. These methods play a pivotal role in accessing and modifying private instance variables, ensuring controlled and secure manipulation of class-specific data. (Singh, 2012)

In summary, the class diagram encapsulates the hierarchical inheritance structure, showcasing the relationships and attributes of Teacher, Lecturer, and Tutor classes. The strategic use of getter and setter methods underlines the program's commitment to encapsulation, promoting a well-organized and efficient approach to handling private instance variables.

3. Pseudocode

Pseudocode is a simple plan for a computer program. It uses basic English phrases instead of complex code words. It's like a list of steps with arrows to show repeating actions. Indentation helps organize the flow. It saves time in coding and helps people understand the program. Some projects use only pseudocode, others use both flow charts and pseudocode. (Toulson, 2017)

3.1) Pseudocode for Teacher class:

```
START PROGRAM Teacher AS PARENT
    DECLARE PRIVATE VARIABLE int teacherId
    DECLARE PRIVATE VARIABLE string teacherName
    DECLARE PRIVATE VARIABLE string address
    DECLARE PRIVATE VARIABLE string workingType
    DECLARE PRIVATE VARIABLE string employmentStatus
    DECLARE PRIVATE VARIABLE int workingHours

    DECLARE CONSTRUCTOR Teacher with teacherId, teacherName, address, workingType, employmentStatus AS PARAMETERS
        ASSIGN teacherId VALUE IN INSTANCE VARIABLE teacherId
        ASSIGN teacherName VALUE IN INSTANCE VARIABLE teacherName
        ASSIGN address VALUE IN INSTANCE VARIABLE address
        ASSIGN workingType VALUE IN INSTANCE VARIABLE workingType
        ASSIGN employmentStatus VALUE IN INSTANCE VARIABLE employmentStatus
    END CONSTRUCTOR Teacher
```

```
DECLARE METHOD getTeacherId WITH RETURN TYPE int

    RETURN teacherId

END METHOD getTeacherId


DECLARE METHOD getTeacherName WITH RETURN TYPE STRING

    RETURN teacherName

END METHOD getTeacherName


DECLARE METHOD getAddress WITH RETURN TYPE STRING

    RETURN address

END METHOD getAddress


DECLARE METHOD getWorkingType WITH RETURN TYPE STRING

    RETURN workingType

END METHOD getWorkingType


DECLARE METHOD getEmploymentStatus WITH RETURN TYPE STRING

    RETURN employmentStatus

END METHOD getEmploymentStatus


DECLARE METHOD getWorkingHours WITH RETURN TYPE INT

    RETURN workingHours

END METHOD getWorkingHours
```

```
    DECLARE VOID METHOD setWorkingHours WITH hours AS PARAMETER
        ASSIGN hours VALUE IN INSTANCE VARIABLE workingHours
    END METHOD setWorkingHours


    DECLARE VOID METHOD DISPLAY
        PRINT "Teacher ID: " + teacherId
        PRINT "Teacher Name: " + teacherName
        PRINT "Address: " + address
        PRINT "Working Type: " + workingType
        PRINT "Employment Status: " + employmentStatus


        START IF STATEMENT
        IF workingHours > 0 THEN
            PRINT "Working Hours: " + workingHours
        ELSE
            PRINT "Working Hours have not been assigned yet."
        END IF
        END IF
END PROGRAM Teacher
```

3.2) Pseudocode for Lecturer class:

```
START PROGRAM Lecturer AS CHILD OF Teacher
    DECLARE PRIVATE VARIABLE string department
    DECLARE PRIVATE VARIABLE int yearsOfExperience
    DECLARE PRIVATE VARIABLE int gradedScore
    DECLARE PRIVATE VARIABLE boolean hasGraded

    DECLARE CONSTRUCTOR Lecturer with teacherId, teacherName, address,
        workingType, employmentStatus, department, yearsOfExperience AS PARAMETERS
        CALL SUPERCLASS CONSTRUCTOR Teacher with teacherId, teacherName, address,
            workingType, employmentStatus
        ASSIGN department VALUE IN INSTANCE VARIABLE department
        ASSIGN yearsOfExperience VALUE IN INSTANCE VARIABLE yearsOfExperience
        ASSIGN gradedScore VALUE 0 IN INSTANCE VARIABLE gradedScore
        ASSIGN hasGraded VALUE false IN INSTANCE VARIABLE hasGraded
        CALL SUPERCLASS METHOD setWorkingHours WITH 0 AS PARAMETER
    END CONSTRUCTOR Lecturer

    DECLARE METHOD getDepartment WITH RETURN TYPE STRING
        RETURN department
    END METHOD getDepartment

    DECLARE METHOD getYearsOfExperience WITH RETURN TYPE INT
        RETURN yearsOfExperience
    END METHOD getYearsOfExperience

    DECLARE METHOD getGradedScore WITH RETURN TYPE INT
        RETURN gradedScore
    END METHOD getGradedScore

    DECLARE METHOD getHasGraded WITH RETURN TYPE BOOLEAN
```

```
        RETURN hasGraded
    END METHOD getHasGraded

    DECLARE VOID METHOD setGradedScore WITH score AS PARAMETER
        ASSIGN score VALUE IN INSTANCE VARIABLE gradedScore
    END METHOD setGradedScore

    START METHOD gradeAssignment WITH PARAMETERS gradedScore, department, yearsOfExperience
        START IF STATEMENT
            IF yearsOfExperience >= 5 AND this.department EQUALS department THEN
                START IF STATEMENT
                    IF gradedScore LESS THAN OR EQUAL TO 0 OR gradedScore GREATER THAN OR EQUAL TO 100 THEN
                        PRINT "Out of range grading score. Must be between 1 and 100."
                    ELSE
                        // Grading logic
                        START IF-ELSE CHAIN
                            IF gradedScore >= 70 THEN
                                PRINT "Grade: A"
                            ELSE IF gradedScore >= 60 THEN
                                PRINT "Grade: B"
                            ELSE IF gradedScore >= 50 THEN
                                PRINT "Grade: C"
                            ELSE IF gradedScore >= 40 THEN
                                PRINT "Grade: D"
                            ELSE
                                PRINT "Grade: E"
                        END IF-ELSE CHAIN
```

```
                  // Update gradedScore and hasGraded
                  this.gradedScore = gradedScore
                  this.hasGraded = true
               END IF
            END IF STATEMENT
         ELSE
            // Display a suitable message when assignments have not been graded.
            PRINT "Assignments have not been graded."
         END IF
      END IF STATEMENT
   END METHOD gradeAssignment

   OVERRIDE METHOD display
      CALL SUPERCLASS METHOD display
      PRINT "Department: " + department
      PRINT "Years of Experience: " + yearsOfExperience
      START IF STATEMENT
         IF hasGraded THEN
            PRINT "Graded Score: " + gradedScore
         ELSE
            PRINT "This Lecturer has not graded any assignment yet."
         END IF
      END IF
   END METHOD display
END PROGRAM Lecturer
```

3.3) Pseudocode for Tutor class:

```
START PROGRAM Tutor AS CHILD OF Teacher
    START PROGRAM Tutor AS CHILD OF Teacher
    DECLARE PRIVATE VARIABLE double salary
    DECLARE PRIVATE VARIABLE string specialization
    DECLARE PRIVATE VARIABLE string academicQualifications
    DECLARE PRIVATE VARIABLE int performanceIndex
    DECLARE PRIVATE VARIABLE boolean isCertified


    DECLARE CONSTRUCTOR Tutor WITH teacherId, teacherName, address, workingType, employmentStatus,
                    workingHours, salary, specialization, academicQualifications, performanceIndex AS PARAMETERS
        CALL SUPERCONSTRUCTOR Teacher WITH teacherId, teacherName, address, workingType, employmentStatus AS ARGUMENTS
        CALL setWorkingHours METHOD WITH workingHours AS ARGUMENT
        ASSIGN salary VALUE IN INSTANCE VARIABLE salary
        ASSIGN specialization VALUE IN INSTANCE VARIABLE specialization
        ASSIGN academicQualifications VALUE IN INSTANCE VARIABLE academicQualifications
        ASSIGN performanceIndex VALUE IN INSTANCE VARIABLE performanceIndex
        ASSIGN isCertified VALUE IN INSTANCE VARIABLE false
    END CONSTRUCTOR Tutor


    DECLARE METHOD getSalary WITH RETURN TYPE double
        RETURN salary
    END METHOD getSalary


    DECLARE METHOD getSpecialization WITH RETURN TYPE STRING
        RETURN specialization
```

```
END METHOD getSpecialization

DECLARE METHOD getAcademicQualifications WITH RETURN TYPE STRING
    RETURN academicQualifications
END METHOD getAcademicQualifications

DECLARE METHOD getPerformanceIndex WITH RETURN TYPE INT
    RETURN performanceIndex
END METHOD getPerformanceIndex

DECLARE METHOD isCertified WITH RETURN TYPE BOOLEAN
    RETURN isCertified
END METHOD isCertified

DECLARE VOID METHOD setSalary WITH newSalary, newPerformanceIndex AS PARAMETERS
    START IF STATEMENT
    IF performanceIndex >= 5 AND getWorkingHours() > 20 THEN
        DECLARE LOCAL VARIABLE double appraisal
        ASSIGN appraisal VALUE 0.05
        START IF STATEMENT
        IF performanceIndex >= 8 THEN
            ASSIGN appraisal VALUE 0.1
        ELSE IF performanceIndex EQUALS 10 THEN
            ASSIGN appraisal VALUE 0.2
        END IF
        ASSIGN salary VALUE newSalary + (newSalary * appraisal)
        ASSIGN isCertified VALUE true
        PRINT "Salary has been approved, and the appraisal has been applied!"
        ELSE
            PRINT "Tutor cannot be certified yet. Salary cannot be approved."
        END IF
    END METHOD setSalary

    DECLARE VOID METHOD removeTutor
        START IF STATEMENT
        IF NOT isCertified THEN
            ASSIGN salary VALUE 0.0
            ASSIGN specialization VALUE ""
            ASSIGN academicQualifications VALUE ""
            ASSIGN performanceIndex VALUE 0
            ASSIGN isCertified VALUE false
            PRINT "Tutor is removed successfully."
        ELSE
            PRINT "The tutor is certified. Cannot remove certified tutor."
        END IF
    END METHOD removeTutor

    DECLARE VOID METHOD display
        CALL SUPERMETHOD display // Calling display method of the parent
class (Teacher) to display teacher details
        PRINT "Salary: " + salary
        PRINT "Specialization: " + specialization
        PRINT "Academic Qualifications: " + academicQualifications
        PRINT "Performance Index: " + performanceIndex
        PRINT "Certified: " + isCertified
    END METHOD display
END PROGRAM Tutor
```

4. Method Description

Accessor Method

An accessor method, also referred to as a getter method, is designed to retrieve the value of a private variable in Java. These methods have a return type corresponding to the data type of the accessed variable. Accessor methods enable the retrieval of specific attribute values, providing a means to obtain information without directly exposing the underlying implementation. (Plynko, 2022)

Examples of accessor methods within this program include:

- getTeacherId(): Accesses and returns the teacherId attribute.
- getAddress(): Accesses and returns the address attribute.
- getWorkingType(): Retrieves and returns the workingType attribute.
- getEmploymentStatus(): Accesses and returns the employmentStatus attribute.
- getWorkingHours(): Retrieves and returns the value of the workingHours attribute.

Mutator Method

A mutator method, commonly known as a setter method, is employed to modify the value of a private field in Java. These methods have a void return type and accept parameters of the same data type as the corresponding instance variable. Mutator methods facilitate the modification of object state, allowing controlled updates to private attributes. (Plynko, 2022)

Examples of mutator methods within this program include:

- setWorkingHours(int hours): Modifies the workingHours attribute with the specified value.
- setGradedScore(int score): Sets the gradedScore attribute with the given score.

- setSalary(double salary, int performanceIndex, int workingHours, boolean isCertified): Adjusts the salary attribute based on specific conditions.

These accessor and mutator methods play a crucial role in encapsulating the internal state of objects, promoting data integrity and controlled access to class attributes. The following sections detail the specific methods present in each class.

4.1) Methods in Class Teacher

- Teacher(int teacherId, String teacherName, String address, String workingType, String employmentStatus)
    - Initializes instance variables by passing values through parameters.
- getTeacherId()
    - Accessor method used to return the value of teacherId.
- setClientName(String newClientName)
    - Mutator method used to set the value of the attribute newClientName through the parameter.
- getClientName()
    - Accessor method used to return the value of clientName.
- getAddress()
    - Accessor method used to return the value of address.
- getWorkingType()
    - Accessor method used to return the value of workingType.
- getEmploymentStatus()
    - Accessor method used to return the value of employmentStatus.
- setWorkingHours(int hours)
    - Mutator method used to set the value of workingHours through the parameter.
- getWorkingHours()
    - Accessor method used to return the value of workingHours.
- void display()
    - Method used to display all the assigned values of the attributes.

4.2) Methods in Class Lecturer

- Lecturer(int teacherId, String teacherName, String address, String workingType, String employmentStatus, String department, int yearsOfExperience)
  - Initializes instance variables by passing values through parameters.
- getDepartment()
  - Accessor method used to return the value of department.
- getYearsOfExperience()
  - Accessor method used to return the value of yearsOfExperience.
- getGradedScore()
  - Accessor method used to return the value of gradedScore.
- getHasGraded()
  - Accessor method used to return the value of hasGraded.
- setGradedScore(int score)
  - Mutator method used to set the value of gradedScore through the parameter.
- gradeAssignment(int gradedScore, String department, int yearsOfExperience)
  - Method to grade assignments based on certain conditions, updating gradedScore and hasGraded.
- void display()
  - Override of the display method in the superclass, displaying all assigned values of the attributes, including those specific to the Lecturer class.

4.3) Methods in Class Tutor

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

    - Initializes instance variables by passing values through parameters.
- getSalary()
    - Accessor method used to return the value of salary.
- getSpecialization()
    - Accessor method used to return the value of specialization.
- getAcademicQualifications()
    - Accessor method used to return the value of academicQualifications.
- getPerformanceIndex()
    - Accessor method used to return the value of performanceIndex.
- isCertified()
    - Accessor method used to return the value of isCertified.
- setSalary(double salary, int performanceIndex, int workingHours, boolean isCertified)
    - Mutator method used to set the value of salary based on specific conditions.
- removeTutor()
    - Method to remove tutor, resetting attributes if the tutor is not certified.
- void display()
    - Override of the display method in the superclass, displaying all assigned values of the attributes, including those specific to the Tutor class.

5. Testing

5.1) Test 1:

• Inspecting the Lecturer Class

Table 1: Table of lecturer for test 1

Objective	Inspect the Lecturer class object, grade an assignment, and verify the graded score and department.

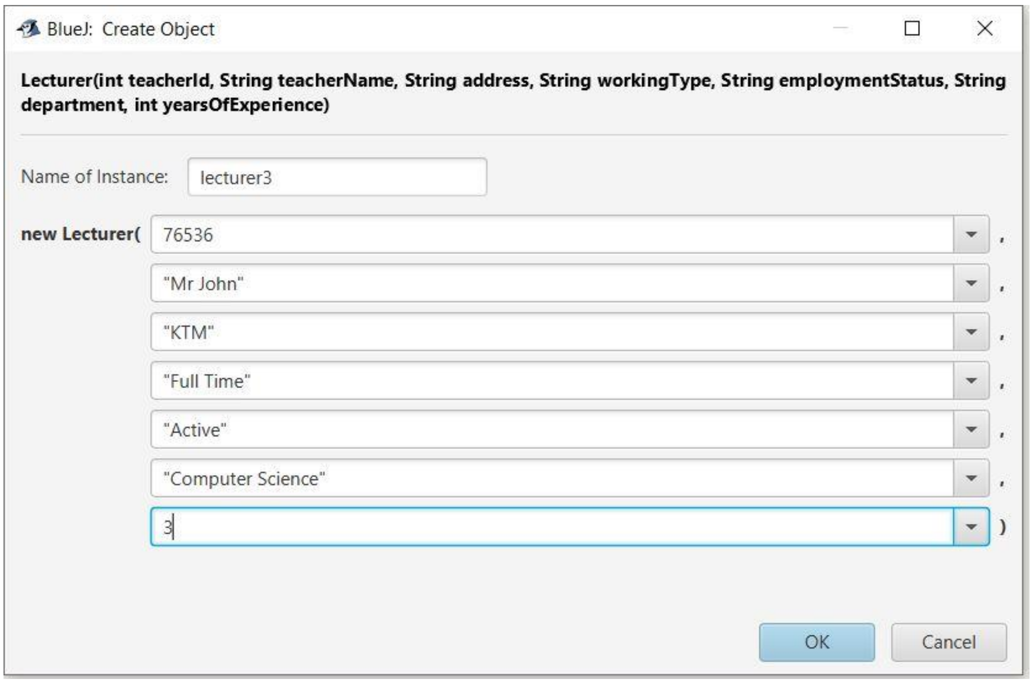
Actions	The Lecturer Class is assigned the following attributes :
	teacherId = 76536
	teacherName = “Mr John”
	address = “KTM”
	workingType = “Full Time”
	employmentStatus = “Active”
	department = “Computer Science”
	yearsOfExperience = 6
	
	The Class is then inspected.
	Afterwards, the void gradeAssignment is declared with the following attributes:
	gradedScore : 70
	department : “Computer Science”
	yearsOfExperience : 6
	
	The Class is then inspected again.

Expected Result	The graded score should be updated to 70. - The department should reflect "Computer Science". - The hasGraded attribute should be updated to true. - Updated attributes of the Lecturer object.
Actual Result	The assignment was graded and Boolean hasGraded became true.
Conclusion	The test has become successful.

At first, we need to create an object and insert the required values

• Creating Object and Inserting Values

Now, we need to create an object and insert the required values.

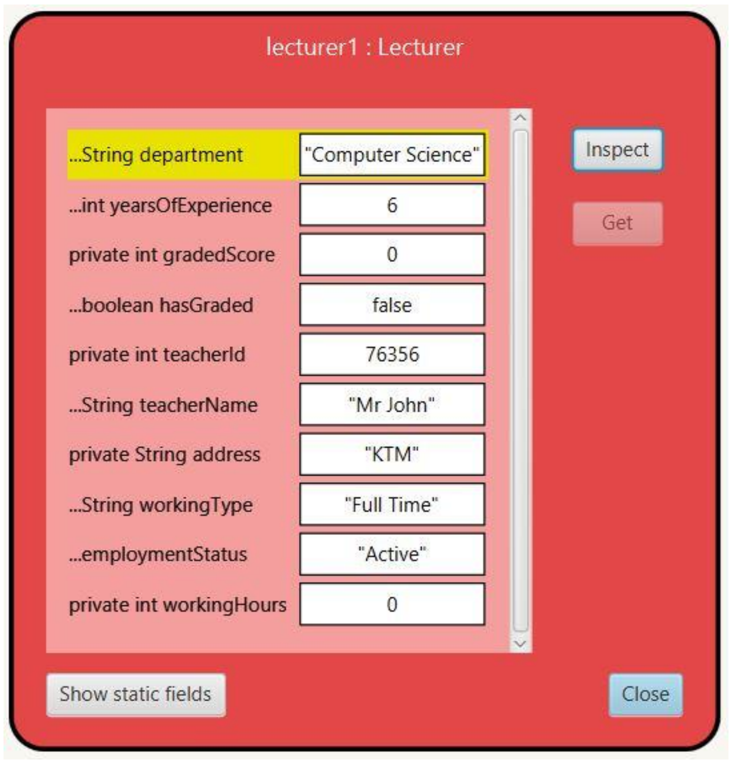

BlueJ: Create Object

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

Name of Instance: lecturer3

new Lecturer(76536 ,
"Mr John" ,
"KTM" ,
"Full Time" ,
"Active" ,
"Computer Science" ,
3)

Figure 5: Inserting values to Lecturer object

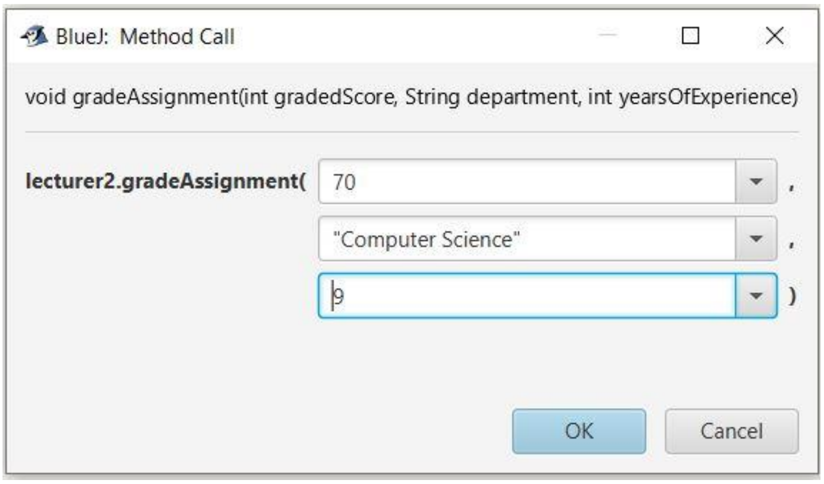

lecturer1 : Lecturer

...String department	"Computer Science"
...int yearsOfExperience	6
private int gradedScore	0
...boolean hasGraded	false
private int teacherId	76356
...String teacherName	"Mr John"
private String address	"KTM"
...String workingType	"Full Time"
...employmentStatus	"Active"
private int workingHours	0

Figure 6: Inspecting the Lecturer Class before setting the GradedScore

• Setting the Graded Score

In order to set the Graded, we need to insert the required values. After inserting the values, we need to re-inspect the Lecturer class.



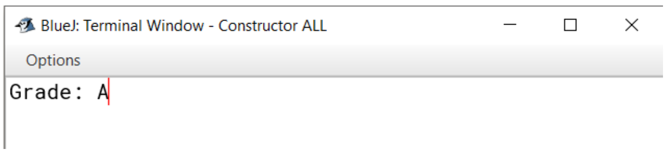
BlueJ: Method Call

void gradeAssignment(int gradedScore, String department, int yearsOfExperience)

lecturer2.gradeAssignment(70 , "Computer Science" , 9)

Figure 7: Setting the Graded Score

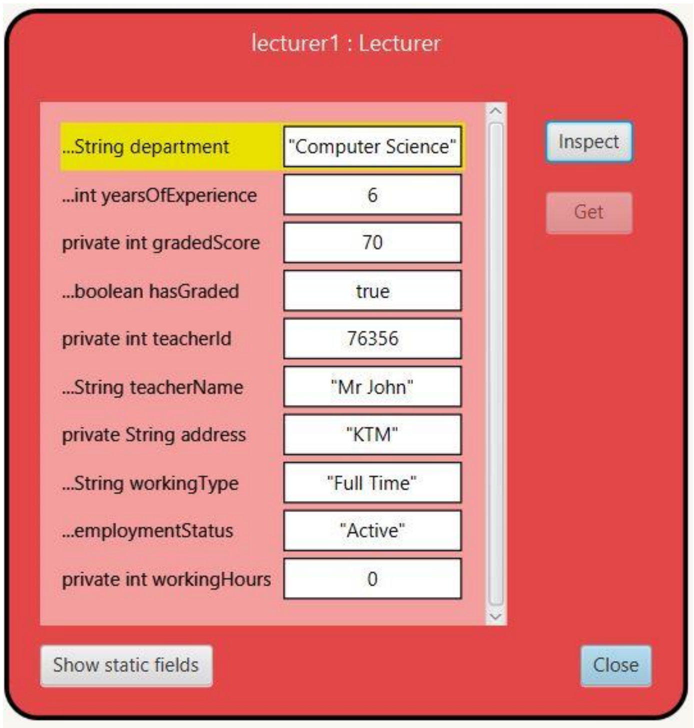
• Output after setting the Graded Score



BlueJ: Terminal Window - Constructor ALL

```
Grade: A
```

Figure 8: Output after setting the Graded Score



lecturer1 : Lecturer

...String department	"Computer Science"
...int yearsOfExperience	6
private int gradedScore	70
...boolean hasGraded	true
private int teacherId	76356
...String teacherName	"Mr John"
private String address	"KTM"
...String workingType	"Full Time"
...employmentStatus	"Active"
private int workingHours	0

Figure 9: Inspecting the Lecturer Class After setting the GradedScore

5.2) Test 2:

• Inspecting the Tutor Class

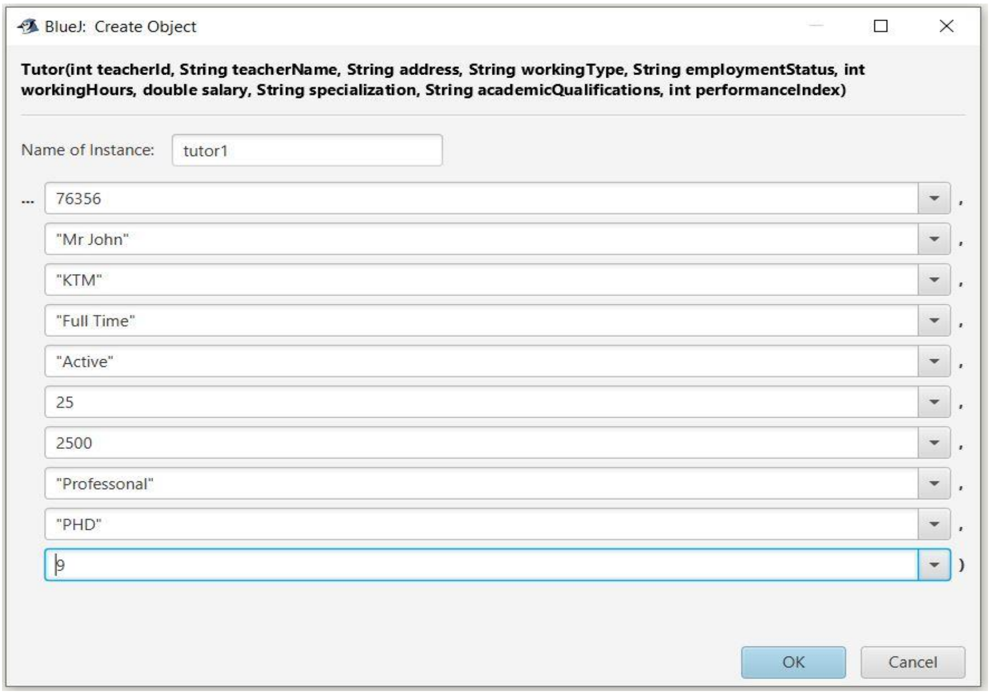
Table 2: Table of Tutor for test 2

Objective	Inspect the Tutor class object, update salary, and verify the updated salary.

Actions	The Tutor Class is assigned the following attributes :
	teacherID = 76356
	teacherName = “Mr John”
	address = “KTM”
	workingType = “Full Time”
	employmentStatus = “Active”
	workingHours = 25
	salary = 2500
	specialization = “Professonal”
	academicQualifications = “PHD”
	performanceIndex = 9
	
	The Class is then inspected.
	Afterwards, the void setSalary is declared with the following attributes:
	newSalary = 50000
	newPerformanceIndex = 9
	
	The Class is then inspected again.

Expected Result	A new salary should appear despite the default salary value entered through the parameter, by adding appraisal to it. Also, the boolean isCertified should also be true.
Actual Result	New salary with appraisal appeared and the boolean isCertified was also set to be true
Conclusion	Test become successful

• Creating the object of Tutor Class

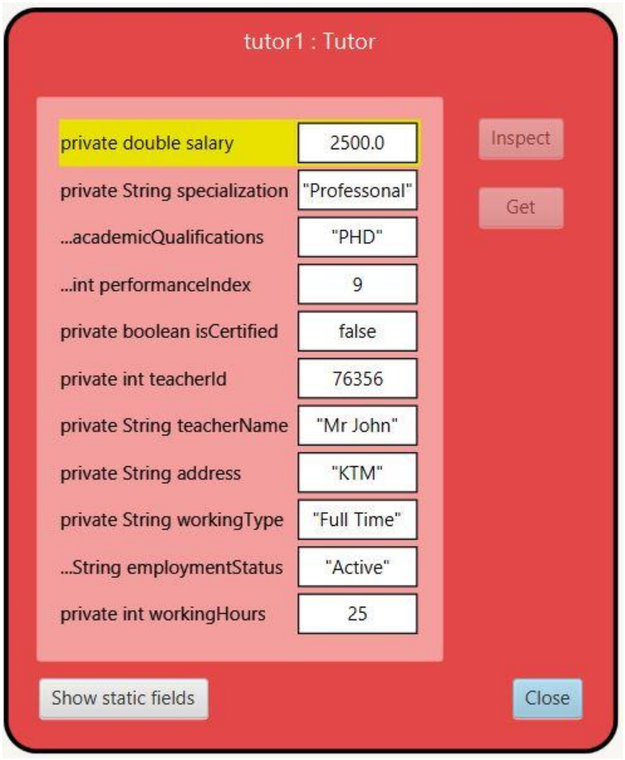

BlueJ: Create Object

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

Name of Instance:	tutor1
...	76356 ,
	"Mr John" ,
	"KTM" ,
	"Full Time" ,
	"Active" ,
	25 ,
	2500 ,
	"Professonal" ,
	"PHD" ,
	9)

Figure 10: Inserting values to Tutor object

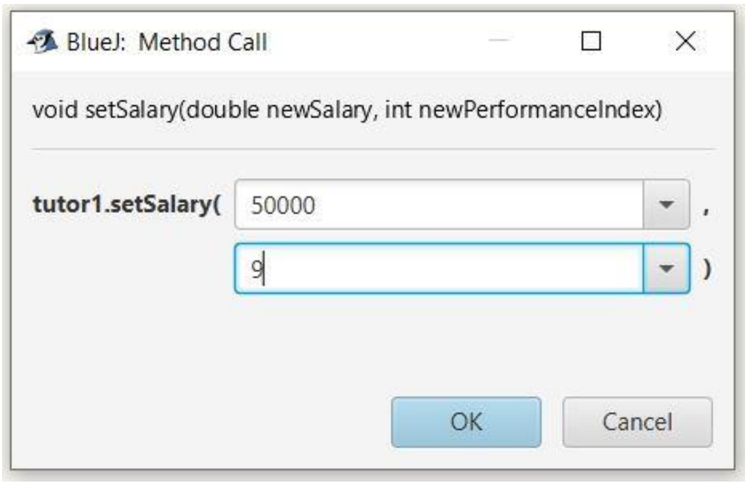
• inspecting the Tutor Class before setting salary:-


tutor1 : Tutor

private double salary	2500.0
private String specialization	"Professonal"
...academicQualifications	"PHD"
...int performanceIndex	9
private boolean isCertified	false
private int teacherId	76356
private String teacherName	"Mr John"
private String address	"KTM"
private String workingType	"Full Time"
...String employmentStatus	"Active"
private int workingHours	25

Figure 11: Inspecting the Tutor class before setting the Salary

• Assigning the Tutor Class for salary:-

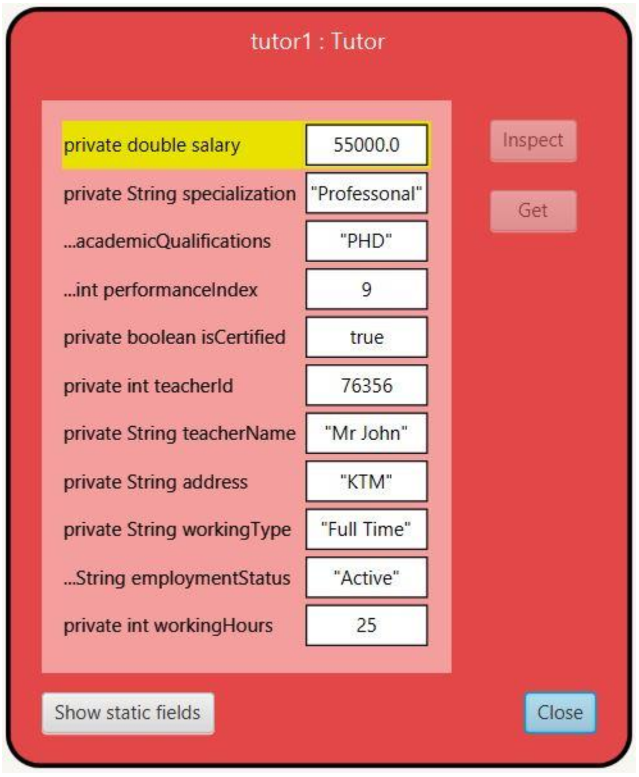

BlueJ: Method Call

void setSalary(double newSalary, int newPerformanceIndex)

tutor1.setSalary(50000 ,
9)

Figure 12: Assigning the Tutor Class for salary

• inspecting the Tutor Class After setting salary:-


tutor1 : Tutor

private double salary	55000.0
private String specialization	"Professonal"
...academicQualifications	"PHD"
...int performanceIndex	9
private boolean isCertified	true
private int teacherId	76356
private String teacherName	"Mr John"
private String address	"KTM"
private String workingType	"Full Time"
...String employmentStatus	"Active"
private int workingHours	25

Figure 13: inspecting the Tutor Class After setting salary

5.3) Test 3:

• Inspecting the Tutor class for removing Tutor:-

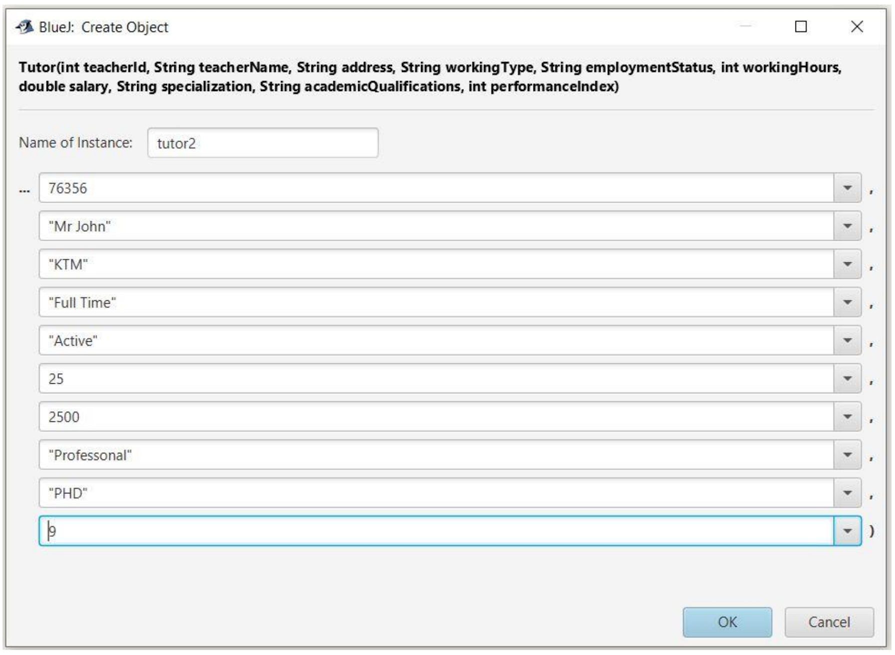
Table 3: Table of Tutor for test 3

Objective	Inspect the Tutor class object, update salary, and verifying the removeTutor.

Actions	The Tutor Class is assigned the following attributes :
	teacherID = 76356
	teacherName = “Mr John”
	address = “KTM”
	workingType = “Full Time”
	employmentStatus = “Active”
	workingHours = 25
	salary = 2500
	specialization = “Professonal”
	academicQualifications = “PHD”
	performanceIndex = 9
	
	the void removeTutor is declared
	
	The Class is then inspected.

Expected Result	A display message displaying that the tutor is removed is expected. Also, upon inspection, some details of the tutor including salary, specialization, academicQualifications and performance index are all set to null values. Also, the certification of the tutor is set to false.
Actual Result	Display message received. The details of the tutor are set to null values. Also, the certification of the tutor is set to false.
Conclusion	Test become successful

• Creating the object of Tutor Class



BlueJ: Create Object

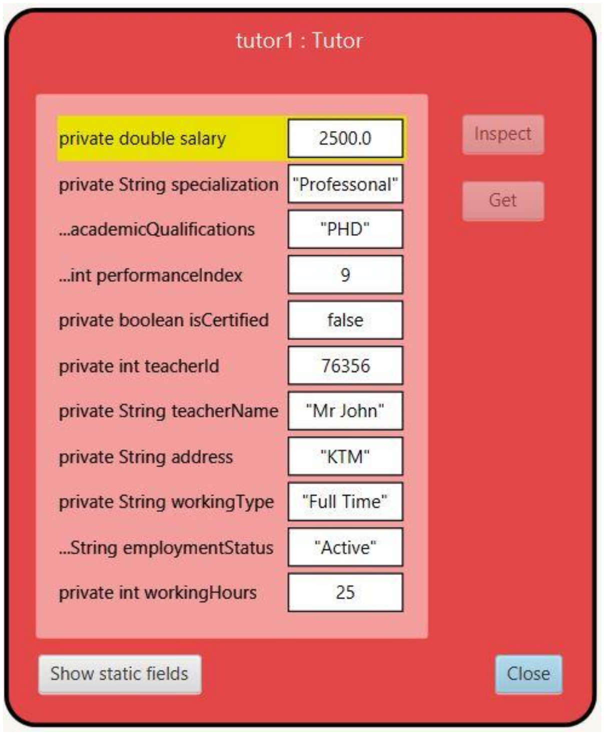
Tutor(int teacherId, String teacherName, String address, String workingType, String employmentStatus, int workingHours, double salary, String specialization, String academicQualifications, int performanceIndex)

Name of Instance: tutor2

76356,
"Mr John",
"KTM",
"Full Time",
"Active",
25,
2500,
"Professonal",
"PHD",
9)

Figure 14: Creating the object of Tutor Class

• inspecting the Tutor Class before Removing Tutor:-

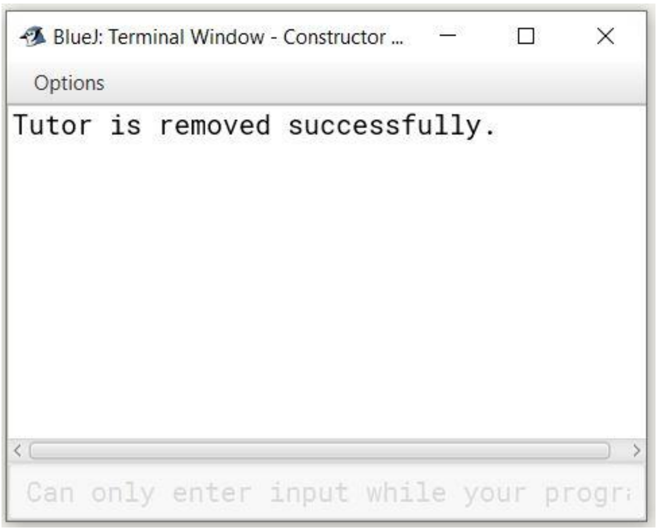


tutor1 : Tutor

private double salary	2500.0
private String specialization	"Professonal"
...academicQualifications	"PHD"
...int performanceIndex	9
private boolean isCertified	false
private int teacherId	76356
private String teacherName	"Mr John"
private String address	"KTM"
private String workingType	"Full Time"
...String employmentStatus	"Active"
private int workingHours	25

Figure 15: inspecting the Tutor Class before Removing Tutor

• Message got after running the RemoveTutor method:-

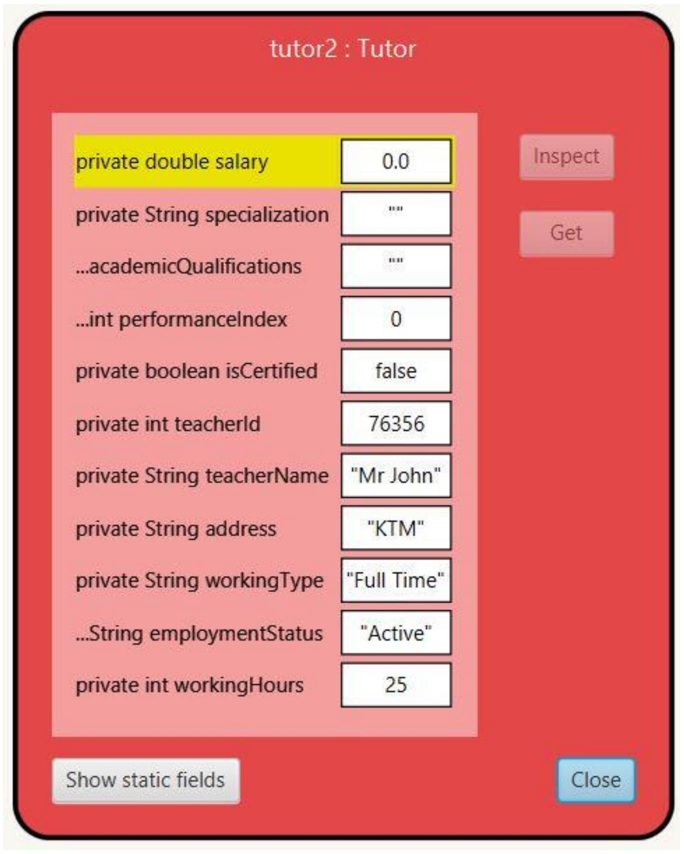

BlueJ: Terminal Window - Constructor ...
Options

```
Tutor is removed successfully.
```

Can only enter input while your progr

Figure 16: message After running the RemoveTutor method

• Reinspecting the Tutor Class After Removing Tutor:-



private double salary	0.0
private String specialization	""
...academicQualifications	""
...int performanceIndex	0
private boolean isCertified	false
private int teacherId	76356
private String teacherName	"Mr John"
private String address	"KTM"
private String workingType	"Full Time"
...String employmentStatus	"Active"
private int workingHours	25

Figure 17: Reinspecting the Tutor Class After Removing Tutor

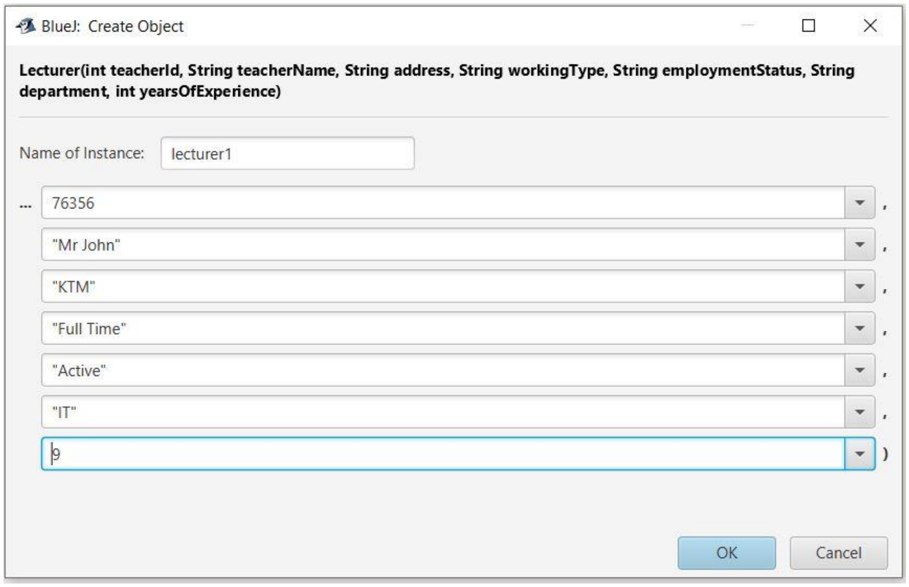
Test 4:

4.1 Displaying the lecturer Class

Table 4: Table of Lecturer for test 4.1

Objective	To display the details of Lecturer Class
Actions	The Lecturer Class is assigned the following attributes : teacherID = 76356 teacherName = “Mr John” address = “KTM” workingType = “Full Time” employmentStatus = “Active” department = “Computer Science” yearsOfExperience = 7 Then, the void gradeAssignment() method is run to enter the gradeScore, department and years of Experience as follows : gradedScore : 70 department : “Computer Science” yearsOfExperience : 7 After the values are inserted to the attributes, the void display() method is run which displays the details of the class Lecturer.
Expected Result	After the display() method is run, all the details of the Lecturer along with the graded score should be visible in the display.
Actual Result	All the details of the Lecturer is seen in the display.
Conclusion	Test become successful

• First of all, we assign values to the parameters of the constructor inside the Lecturer class.


BlueJ: Create Object

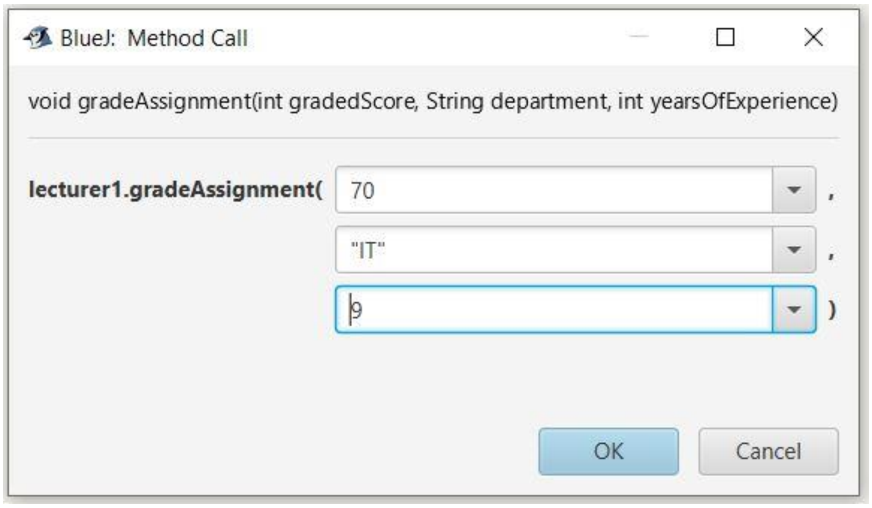
Lecturer(int teacherId, String teacherName, String address, String workingType, String employmentStatus, String department, int yearsOfExperience)

Name of Instance: lecturer1

76356 ,
"Mr John" ,
"KTM" ,
"Full Time" ,
"Active" ,
"IT" ,
9)

Figure 18: creating object for lecturer

• Then, we run the void gradeAssignment() method to provide grade score along with verifying the department and years of experience of the lecturer.


BlueJ: Method Call

void gradeAssignment(int gradedScore, String department, int yearsOfExperience)

lecturer1.gradeAssignment(70 ,
"IT" ,
9)

Figure 19: inserting values for gradeAssignment

• Running the display() of lecturer

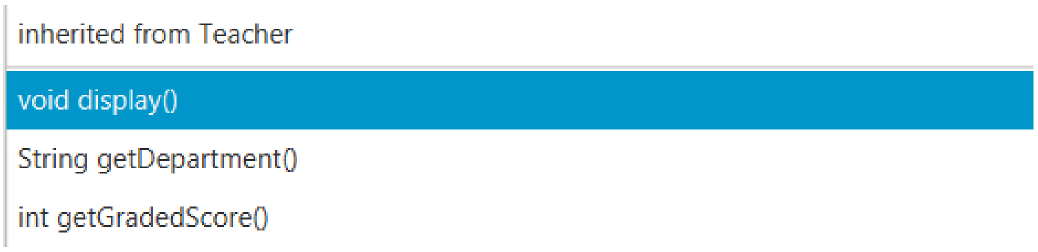
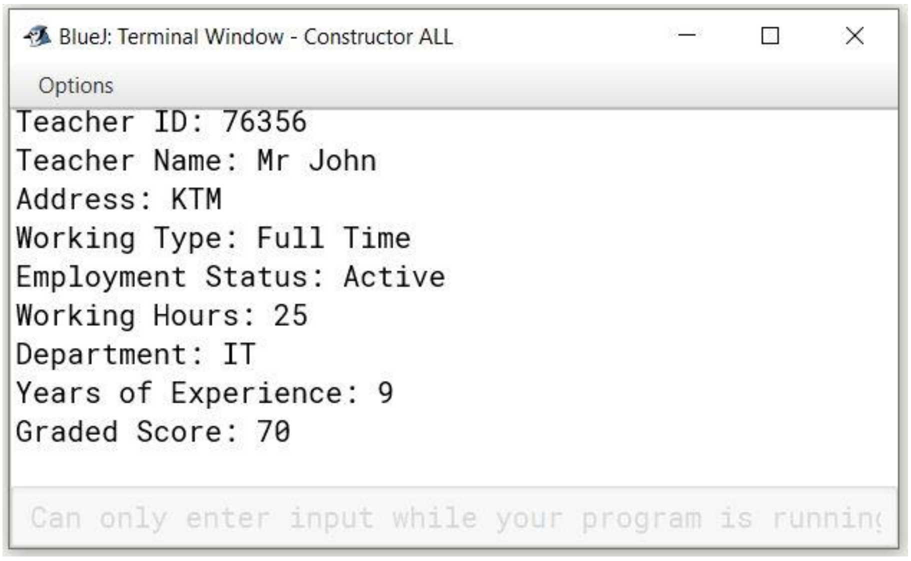


Figure 20: Calling the display() method

• After Running the display() method, following result can be Printed :


BlueJ: Terminal Window - Constructor ALL
Options

```
Teacher ID: 76356
Teacher Name: Mr John
Address: KTM
Working Type: Full Time
Employment Status: Active
Working Hours: 25
Department: IT
Years of Experience: 9
Graded Score: 70
```

Can only enter input while your program is running

Figure 21: Displaying the Lecturer class

4.2 Displaying the Tutor Class

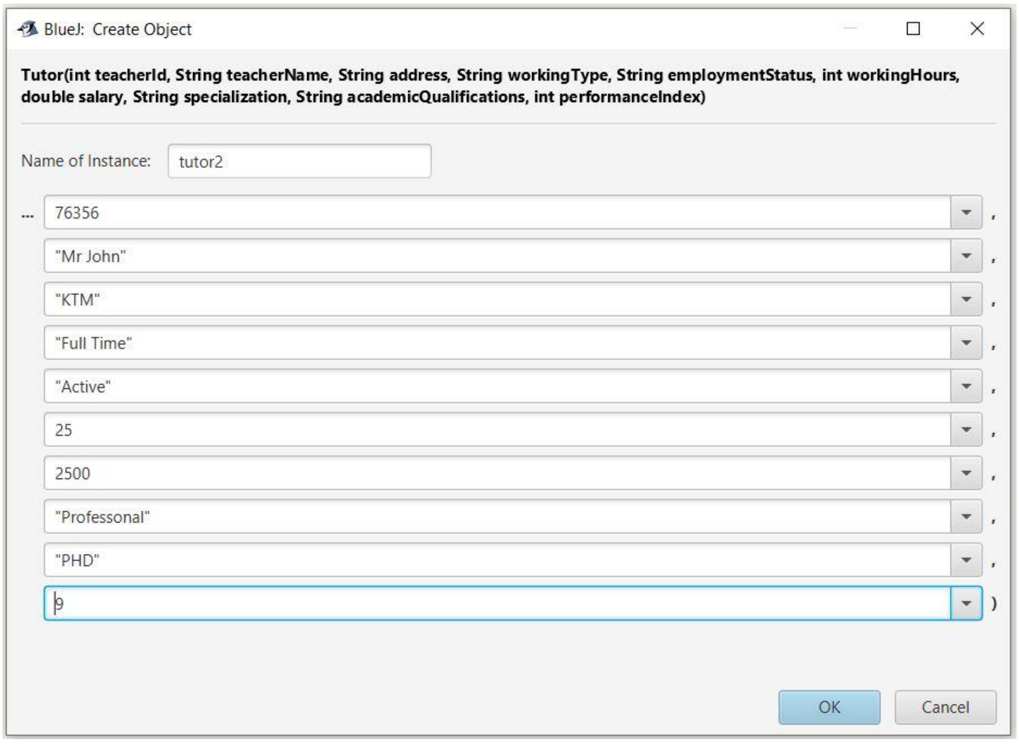
Table 5: Table of Tutor for test 4.2

Objective	To display the details of Tutor Class

Actions	The Tutor Class is assigned the following attributes :
	teacherID = 76356
	teacherName = “Mr John”
	address = “KTM”
	workingType = “Full Time”
	employmentStatus = “Active”
	workingHours = 25
	salary = 2500
	specialization = “Professonal”
	academicQualifications = “PHD”
	performanceIndex = 9
	
	The Class is then inspected.
	Afterwards, the void setSalary is declared with the following attributes:
	newSalary = 50000
	newPerformanceIndex = 9
	
	After the values are inserted to the attributes, the void display() method is run which displays the details of the class Teacher.

Expected Result	After the display() method is run, all the details of the Tutor along with the new salary with appraisal should be visible in the display.
Actual Result	All the details of the Tutor is seen in the display
Conclusion	Test become successful

• First of all, we assign values to the parameters of the constructor inside the Tutor class.


BlueJ: Create Object

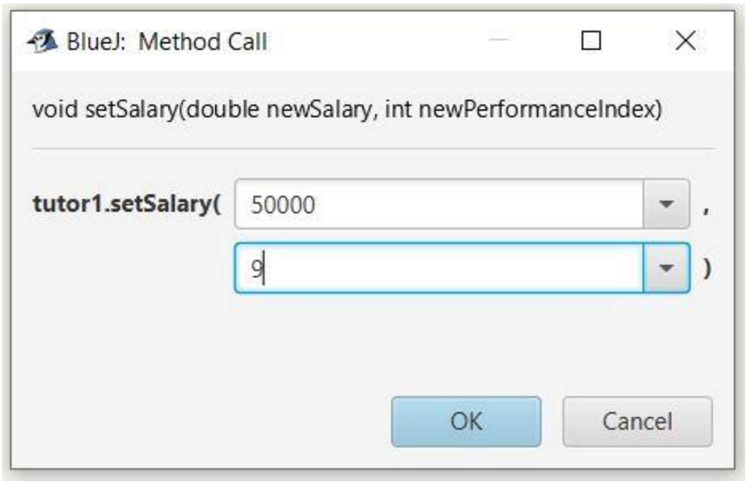
Tutor(int teacherId, String teacherName, String address, String workingType, String employmentStatus, int workingHours, double salary, String specialization, String academicQualifications, int performanceIndex)

Name of Instance: tutor2

```
76356 ,
"Mr John" ,
"KTM" ,
"Full Time" ,
"Active" ,
25 ,
2500 ,
"Professonal" ,
"PHD" ,
9 )
```

Figure 22: Creating an object for Tutor class

• Then, we run the void setSalary() method to provide newSalary along with newPerformanceIndex experience of the Tutor.


BlueJ: Method Call

void setSalary(double newSalary, int newPerformanceIndex)

tutor1.setSalary(50000 , 9)

Figure 23: inserting values for setSalary

• Running the display()

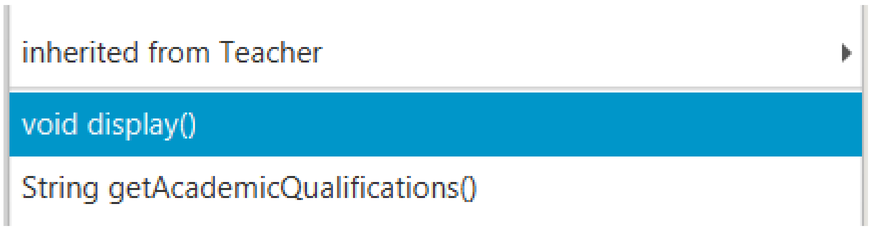
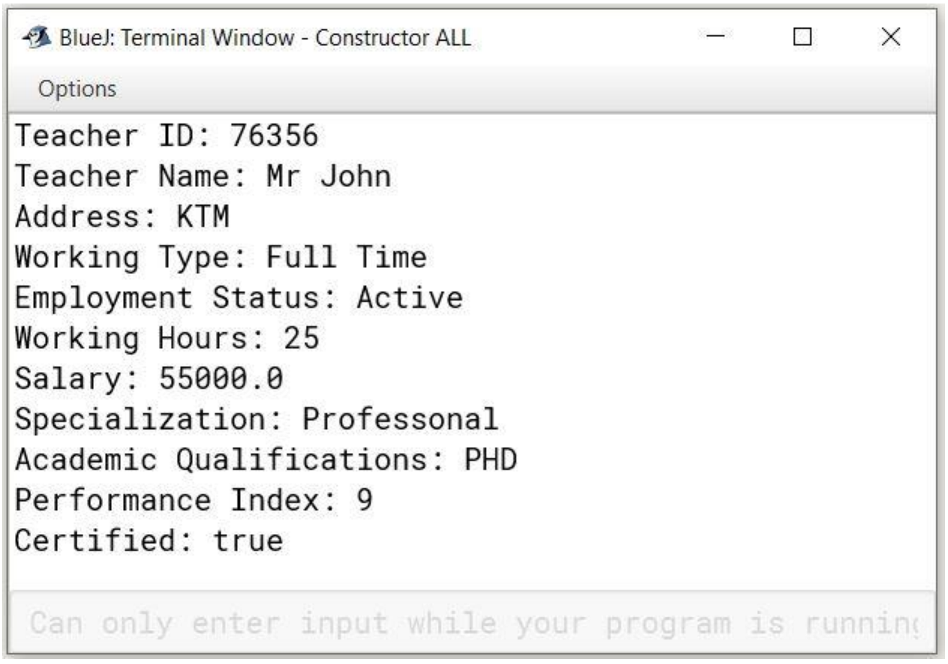


Figure 24: Calling the display() method

• After Running the display() method, following result is Printed:


BlueJ: Terminal Window - Constructor ALL
Options

```
Teacher ID: 76356
Teacher Name: Mr John
Address: KTM
Working Type: Full Time
Employment Status: Active
Working Hours: 25
Salary: 55000.0
Specialization: Professonal
Academic Qualifications: PHD
Performance Index: 9
Certified: true
```

Can only enter input while your program is runnin

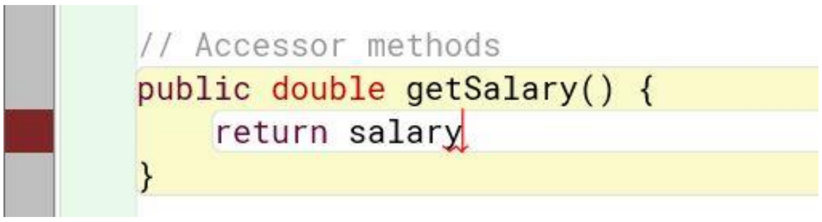
Figure 25: Displaying the Tutor class

6.Error Detection and Error correction

6.1 Syntax Error

A syntax error occurs when the program is not written according to the rules and structure of the Java programming language. (Vyas, 2023) It is the most common error done by coders while writing a program. If there occurs a syntax error, then the code won't work.

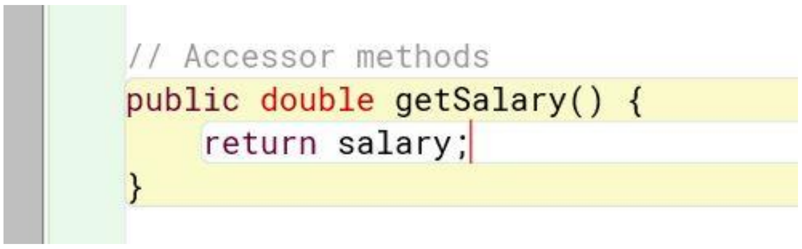
Error:



```
// Accessor methods
public double getSalary() {
    return salary
}
```

Figure 26: Syntax Error

Correction:



```
// Accessor methods
public double getSalary() {
    return salary;
}
```

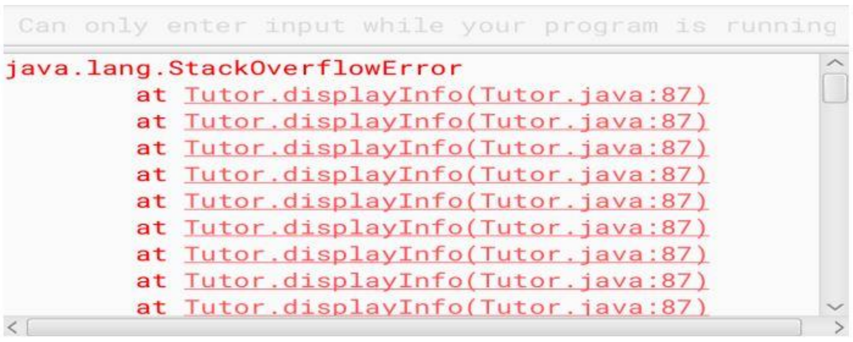
Figure 27: Syntax Error Correction

6.2 Runtime Error

Runtime error refers to the type of error that occurs when a program encounters an unexpected problem during execution, even if it has no syntax or logical errors. It often causes the program to crash or behave abnormally. (Vyas, 2023)

Example: Here's an example of the runtime error I encountered while doing this project:

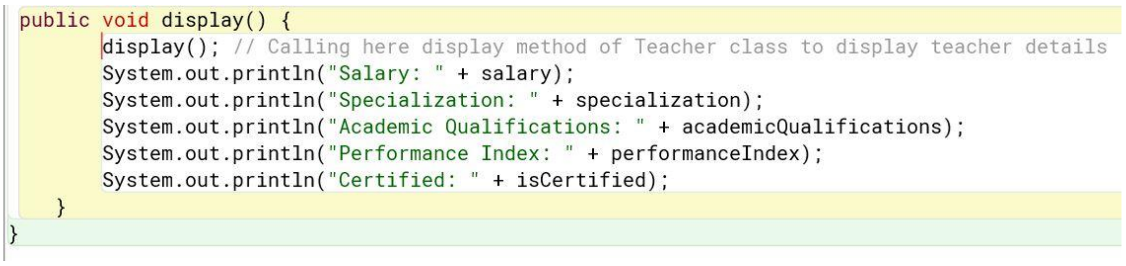
Output of error:



```
Can only enter input while your program is running
java.lang.StackOverflowError
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
        at Tutor.displayInfo(Tutor.java:87)
```

Figure 28: Runtime error detection while execution

Error:

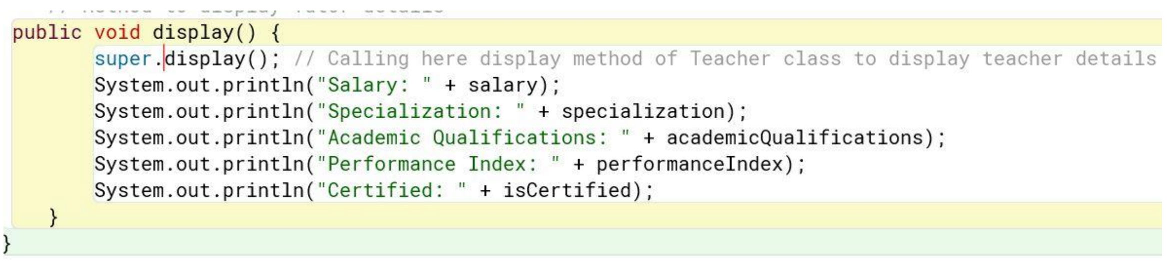


```
public void display() {
        display(); // Calling here display method of Teacher class to display teacher details
        System.out.println("Salary: " + salary);
        System.out.println("Specialization: " + specialization);
        System.out.println("Academic Qualifications: " + academicQualifications);
        System.out.println("Performance Index: " + performanceIndex);
        System.out.println("Certified: " + isCertified);
    }
}
```

Figure 29: : Runtime Error Detection

In the figure above, in the Tutor class inside the display() method, I initially tried to call the superclass Teacher using just display(). This did not result in any kind of error beforehand; logical or syntax error.

Correct:



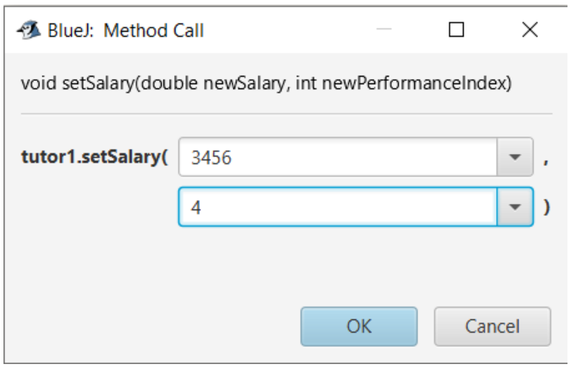
```
public void display() {
        super.display(); // Calling here display method of Teacher class to display teacher details
        System.out.println("Salary: " + salary);
        System.out.println("Specialization: " + specialization);
        System.out.println("Academic Qualifications: " + academicQualifications);
        System.out.println("Performance Index: " + performanceIndex);
        System.out.println("Certified: " + isCertified);
    }
}
```

Figure 30: error correction

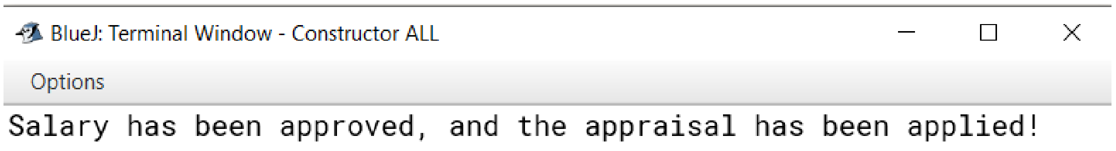
6.3 Logical Error

Logical error is one of the very common errors made by programmers while writing the code. (Vyas, 2023) It can also be very hard to find because it doesn't prevent the program from compiling, but it can generate unexpected results. To avoid such errors, an individual must be careful while writing the code and must recheck once or twice.

Output of error:



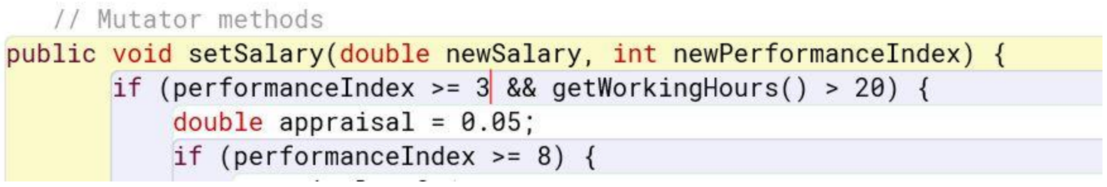
```
void setSalary(double newSalary, int newPerformanceIndex)

tutor1.setSalary( 3456 ,
                  4 )
```



```
Salary has been approved, and the appraisal has been applied!
```

Figure 31: output After the Logical Error

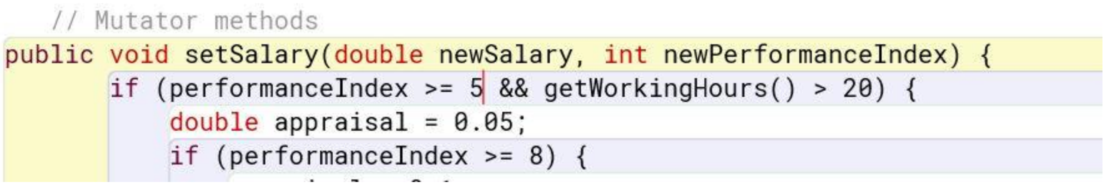
Error:



```
    // Mutator methods
public void setSalary(double newSalary, int newPerformanceIndex) {
        if (performanceIndex >= 3 && getWorkingHours() > 20) {
            double appraisal = 0.05;
            if (performanceIndex >= 8) {
```

Figure 32: Logical Error

Correction:



```
    // Mutator methods
public void setSalary(double newSalary, int newPerformanceIndex) {
        if (performanceIndex >= 5 && getWorkingHours() > 20) {
            double appraisal = 0.05;
            if (performanceIndex >= 8) {
```

Figure 33: Correction of Logical Error

7.Conclusion

The project's completion was smooth but challenging at the same time, as it was the first time for me and everyone else too. We had enough time to finish the coursework, which is why the whole completion was smooth. I believe I did a good job in improving the readability of my code using indentation and comments in the code. Even though it was tough, the journey was worth it. It wasn't just about finishing a task. It was about growing and getting stronger. Dealing with the hard parts, especially the code part, taught me a lot. It wasn't just about grades; I also learned things that go beyond the classroom. Overall, it was like a journey of learning and gaining not just smarts, but also getting tougher.

I learned many new things while doing this coursework. It was very obvious to face some kind of difficulties, as it was a very new thing for the students. However, with the help of the module teachers, friends, and the internet, I was able to complete the coursework on time. I personally had a hard time dealing with the documentation. I am sure everyone hustled day and night for the completion of the coursework, especially the documentation part (for me). But in the end, I got to learn so many things that were new and fruitful for me at the same time.

At last, Completing the project was a big learning experience for me. It was a bit tough because it was something new for everyone. I had enough time to finish the work, and I focused on making my code easy to understand with spaces and comments. Dealing with the hard parts, especially writing about the project, was a challenge, but I learned a lot. With help from teachers, friends, and the internet, I managed to finish on time. Even though it was tough, it was worth it. I not only got better at the project but also became stronger in handling difficulties.

7.References

Kölling, M. (2004, october 4). *bluej.org*. Retrieved from bluej: https://www.bluej.org/doc/tutorial.html

Plynko, P. (2022, June 7). *codegym*. Retrieved from codegym: https://codegym.cc/groups/posts/accessors-and-mutators-in-java

Singh, C. (2012). *beginnersbook*. Retrieved from beginnersbook: https://beginnersbook.com/2013/10/hierarchical-inheritance-java-program/

Toulson, R. (2017). *sciencedirect*. Retrieved from sciencedirect: https://www.sciencedirect.com/topics/engineering/pseudocode

Vyas, H. (2023, September 29). *codingninjas*. Retrieved from codingninjas: https://www.codingninjas.com/studio/library/types-of-error-in-java

8.Appendix

- **Code for Teacher (Super Class)**

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

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

    // Accessor methods
    public int getTeacherId() {
        return teacherId;
    }

    public String getTeacherName() {
        return teacherName;
    }
```

```
public String getAddress() {
    return address;
}

public String getWorkingType() {
    return workingType;
}

public String getEmploymentStatus() {
    return employmentStatus;
}

public int getWorkingHours() {
    return workingHours;
}

// Method to set working hours
public void setWorkingHours(int hours) {
    this.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 have not been assigned yet.");
            }
        }
    }
```

• **Code for Lecturer(Sub class)**

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

    // Constructor for Lecturer which uses the superclass constructor
    public Lecturer(int teacherId, String teacherName, String address,
    String workingType, String employmentStatus,
    String department, int yearsOfExperience) {
        super(teacherId, teacherName, address, workingType, employmentStatus);
        this.department = department;
        this.yearsOfExperience = yearsOfExperience;
        this.gradedScore = 0; // Assign gradedScore as 0
        this.hasGraded = false; // Assign hasGraded as false
    }

    // Accessor methods
    public String getDepartment() {
        return department;
    }

    public int getYearsOfExperience() {
        return yearsOfExperience;
    }

    public int getGradedScore() {
        return gradedScore;
    }
    public boolean getHasGraded() {
        return hasGraded;
    }

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

    // Method to grade assignments
    public void gradeAssignment(int gradedScore, String department, int
yearsOfExperience) {
        if (yearsOfExperience >= 5&&this.department.equals(department))
      {
            if (gradedScore < 0 || gradedScore > 100) {
            System.out.println("Out of range grading score; it must be
between 0 and 100");
        } else {
                // Grading logic
                if (gradedScore >= 70) {
                    System.out.println("Grade: A");
                } else if (gradedScore >= 60) {
                    System.out.println("Grade: B");
                } else if (gradedScore >= 50) {
                    System.out.println("Grade: C");
                } else if (gradedScore >= 40) {
                    System.out.println("Grade: D");
                } else {
                    System.out.println("Grade: E");
                }
                this.gradedScore = gradedScore;
                this.hasGraded = true;
            }
        } else {
            // Display a suitable message when assignments have already
been graded.
            System.out.println("Assignments have not graded.");
        }
    }

    // Override the display method to include Lecturer details
    @Override
```

```
        public void display() {
            super.display();
            System.out.println("Department: " + department);
            System.out.println("Years of Experience: " + yearsOfExperience);
            if (hasGraded) {
                System.out.println("Graded Score: " + gradedScore);
            } else {
                System.out.println("This Lecturer has not graded any assignment yet.");
            }
        }
    }
```

- **Code for Tutor(sub 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(int teacherId, String teacherName, String address, String workingType, String employmentStatus,

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

```
        super(teacherId, teacherName, address, workingType,
employmentStatus);
        setWorkingHours(workingHours);
        this.salary = salary;
        this.specialization = specialization;
        this.academicQualifications = academicQualifications;
        this.performanceIndex = performanceIndex;
        this.isCertified = false;
    }

    // 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;

    }


    // Mutator methods

 public void setSalary(double newSalary, int newPerformanceIndex) {

        if (performanceIndex >= 5 && getWorkingHours() > 20) {

            double appraisal = 0.05;

            if (performanceIndex >= 8) {

                appraisal = 0.1;

            } else if (performanceIndex == 10) {

                appraisal = 0.2;

            }

            this.salary = newSalary + (newSalary * appraisal);

            this.isCertified = true;

            System.out.println("Salary has been approved, and the appraisal has
been applied!");

        } else {

            System.out.println("Tutor cannot be certified yet. Salary cannot be
approved.");

        }

    }
    public void removeTutor() {
        if (!isCertified) {
            salary = 0.0;
            specialization = "";
            academicQualifications = "";
            performanceIndex = 0;
            isCertified = false;
            System.out.println("Tutor is removed successfully.");
        } else {
            System.out.println("The tutor is certified. Cannot remove certified
tutor.");
        }
}
    // Method to display Tutor details
 public void display() {
        super.display(); // Calling here display method of Teacher class to display
teacher details
        System.out.println("Salary: " + salary);
        System.out.println("Specialization: " + specialization);
        System.out.println("Academic Qualifications: " + academicQualifications);
        System.out.println("Performance Index: " + performanceIndex);
        System.out.println("Certified: " + isCertified);
    }
}
```