



**Module Code & Module Title**

**CS4001NT Programming**

**Assessment Weightage & Type:**

**30% Individual Coursework**

**Year and Semester**

**2023 Autumn**

**Student Name: Prince Thapa  
London Met ID: 23049323**

**College ID: np05cp4a230068  
Assignment Due Date: 25th January**

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 mark of zero will be awarded.

Table of Contents

[**1.** **Introduction** 4](#_Toc156978989)

[**1.1** **Components of java:** 5](#_Toc156978990)

[1.2 BlueJ: 6](#_Toc156978991)

[2. Class Diagram: 7](#_Toc156978992)

[2.1 Class Diagram of Teacher: 8](#_Toc156978993)

[2.2 Class diagram of Lecturer: 8](#_Toc156978994)

[2.3 Class diagram of Tutor: 9](#_Toc156978995)

[2.4 Classes in Blue J interface 10](#_Toc156978996)

[3. Pseudo Code 11](#_Toc156978997)

[3.1 Pseudo Code of Teacher Class 12](#_Toc156978998)

[3.2 Pseudo Code of Lecture Class 15](#_Toc156978999)

[3.3 Pseudo Code of Tutor class 19](#_Toc156979000)

[4. Method Description 23](#_Toc156979001)

[4.1 Method description of Teacher class 24](#_Toc156979002)

[4.2 Method description of Lecturer class 25](#_Toc156979003)

[4.3 Method description of Tutor class 26](#_Toc156979004)

[5. Testing 27](#_Toc156979005)

[5.1 Test-1 28](#_Toc156979006)

[5.2 Test 2 32](#_Toc156979007)

[6. Error Detection and correction 38](#_Toc156979008)

Table of table:

[Table 1: Method description of Teacher class 25](#_Toc156978957)

[Table 2: Method description of Lecturer class 26](#_Toc156978958)

[Table 3: Method description of Tutor class 27](#_Toc156978959)

[Table 4: Table of Test-1 28](#_Toc156978960)

[Table 5: Table of Test-2 32](#_Toc156978961)

Table of Figures

[Figure 1: Figure of java 5](#_Toc156978937)

[Figure 2: Figure of BlueJ 7](#_Toc156978938)

[Figure 3: Class diagram of Teacher 8](#_Toc156978939)

[Figure 4: Class diagram of Lecturer 9](#_Toc156978940)

[Figure 5: Class diagram of Tutor 10](#_Toc156978941)

[Figure 6: Classes interface in BlueJ 11](#_Toc156978942)

[Figure 7: Creating Object of Lecturer 29](#_Toc156978943)

[Figure 8: Inspecting object of Lecturer before grading assignment. 29](#_Toc156978944)

[Figure 9: Grading the Assignment 30](#_Toc156978945)

[Figure 10: Inspecting Object after grading assignment. 30](#_Toc156978946)

[Figure 11: Display of Lecturer class 31](#_Toc156978947)

[Figure 12: Creating Object of Tutor class. 32](#_Toc156978948)

[Figure 13: Insepcting of Object of Tutor before setting Salary. 33](#_Toc156978949)

[Figure 14: Setting Salary 33](#_Toc156978950)

[Figure 15: Re-Inspecting object after setting salary. 34](#_Toc156978951)

[Figure 16:Actual display of Tutor Class. 34](#_Toc156978952)

[Figure 17: Displaying syntax error in code. 39](#_Toc156978953)

[Figure 18: Screenshot after correction of syntax error. 40](#_Toc156978954)

[Figure 19: Displaying semantic error in code. 41](#_Toc156978955)

[Figure 20: Screenshot after correction of semantic error. 41](#_Toc156978956)

1. **Introduction**

Java is a programming language that was created in 1995 by James Gosling at Sun Microsystems. Java is widely used for developing applications that run on various platforms, such as Windows, Linux, Mac OS, Android, and iOS. Java is known for its features such as object-oriented design, portability, concurrency, security, and performance. Java follows the principle of "write once, run anywhere", which means that the same code can run on different devices without requiring recompilation. Java is also one of the most popular languages for learning computer science concepts and algorithms.



Figure : Figure of java

* 1. **Components of java:**

Java is a programming language that has three main components. They are Java development Kit (JDK), Java virtual Machine (JVM) and Java Runtime Environment (JRE). All these components are interconnected with each other and are needed to run a java program.

**Java Development Kit (JDK)**

Java Development Kit (JDK) is a software development environment used to develop Java applications. JDK is responsible for converting the java code into byte code and then into machine code.

**Java Virtual Machine (JVM)**

JVM is a specification that defines a runtime environment for executing java bytecode. JVM is platform-dependent, which means it can run on different hardware and software systems. It performs various tasks such as loading, verifying, and executing code, as well as providing memory management and garbage collection. JVM is a part of JRE which also includes libraries and other components to run Java applications.

**Java Runtime Environment (JRE)**

JRE is a software layer that runs on top of a computer’s operating system and provides the class libraries and other resources that a specific Java program needs to run. The JRE is a part of the JDK which is a set of tools for developing java applications.

* 1. BlueJ:

Blue J is an integrated development environment (IDE) for the java programming language. It was created by Michael Kolling and John Rosenberg at the university of Kent in canterbury, England in 1999 and has since become a popular tool for teaching object-oriented programming (OOP) to students of all ages and experience levels.

Blue J is designed to help beginners learns the basic concepts of object-oriented programming and java syntax. Blue J provides a simple graphical user interface (GUI) that allows users to create, edit, compile, run, test, and debug Java programs. However, unlike other IDEs that are geared towards professional developers, Blue J has a simplified interface that focuses on OOP concepts, making it an ideal tool for students and educators. Blue J is free and open source software, and it is widely used in education and research.

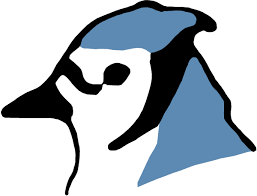


Figure : Figure of BlueJ

1. Class Diagram:

A class diagram is a type of diagram that shows the structure of a system or a program using classes and their relationships. Classes are the building blocks of object-oriented programming, and they represent the concepts, entities, or behaviors of the system. A class diagram can show the attributes and operations of each class, as well as the associations, aggregations, compositions, generalizations, and dependencies between them. A class diagram can help to visualize the design of a system, to identify its components and their responsibilities, and to facilitate communication and collaboration among developers.

We draw class diagram by including instance variables and method. We put sign in front of every instance variable and method to specify access modifier. Like – is for private access modifier and + is for public access modifier and there are different signs for other access modifier.

* 1. Class Diagram of Teacher:

|  |
| --- |
| **Teacher** |
| * teacherId: int * teacherName: String * address: String * workingType: String * employementStatus: String * workingHours: int |
| <<Constructor>> Teacher(teacherId: int, teacherName: String, address: String, workingType: String, employmentStatus: String)   * getTeacherId(): int * getTeacherName(): String * getAddress(): String * getWorkingType(): String * getEmploymentStatus(): String * getWorkingHours(): int * setWorkingHours(workingHours: int): void * display(): void |

Figure : Class diagram of Teacher

* 1. Class diagram of Lecturer:

|  |
| --- |
| **Lecturer** |
| -  department: String       -  yearOfExperience: int       -  gradedScore: int       -  hasGraded: boolean |
| <<Constructor>> Lecturer(teacherId: int, teacherName: String, address: String, workingType: String, employmentStatus: String, department: String, yearOfExperience: int, workingHours: int)   * getDepartment(): String * getYearOfExperience(): int * getGradedScore(): int * getHasGraded(): boolean * setGradedScore(gradedScore: int): void * gradeAssignment(gradedScore: int, department: String, yearOfExperience: int): void * display(): void |

Figure : Class diagram of Lecturer

* 1. Class diagram of Tutor:

|  |
| --- |
| **TUTOR** |
| * salary: double * specialization: String * academicQualifications: String * performanceIndex: int * isCertified: boolean |
| <<Constructor>> Tutor(int teacherId, String teacherName, String address, String workingType, String employmentStatus, int workingHours, double salary, String specialization, String academicQualification, int performanceIndex)   * getSalary(): double * getSpecialization(): String * getAcademicQualifications (): String * getPerformanceIndex(): int * getIsCertified (): boolean * setSalary(double salary, int performanceIndex, int workingHours): void * removeTutor(): void * display(): void |

Figure : Class diagram of Tutor

* 1. Classes in Blue J interface

The Blue J interface allows users to create, edit, compile and run Java classes in a graphical environment. The main window of Blue J shows a class diagram, where each class is represented by a box with its name and attributes. Users can create new classes by clicking on the "New Class" button, or by selecting an existing class and choosing "Duplicate". Users can also drag and drop classes from the project menu or from other sources. To edit a class, users can double-click on its box or right-click and choose "Open Editor". The editor window shows the source code of the class, where users can modify it as needed. To compile a class, users can click on the "Compile" button in the editor window, or right-click on the class box and choose "Compile". If there are any errors or warnings, they will be displayed in the status bar at the bottom of the window. To run a class, users can right-click on its box and choose "Create Object". This will open an object bench, where users can create and interact with objects of the class. Users can also invoke methods on the objects by right-clicking on them and choosing "Call Method".

Overall, understanding classes is essential for programming in Blue J, and it’s easy to create and use them to develop java programs.

Shown below are the classes of Blue J interface of my coursework:

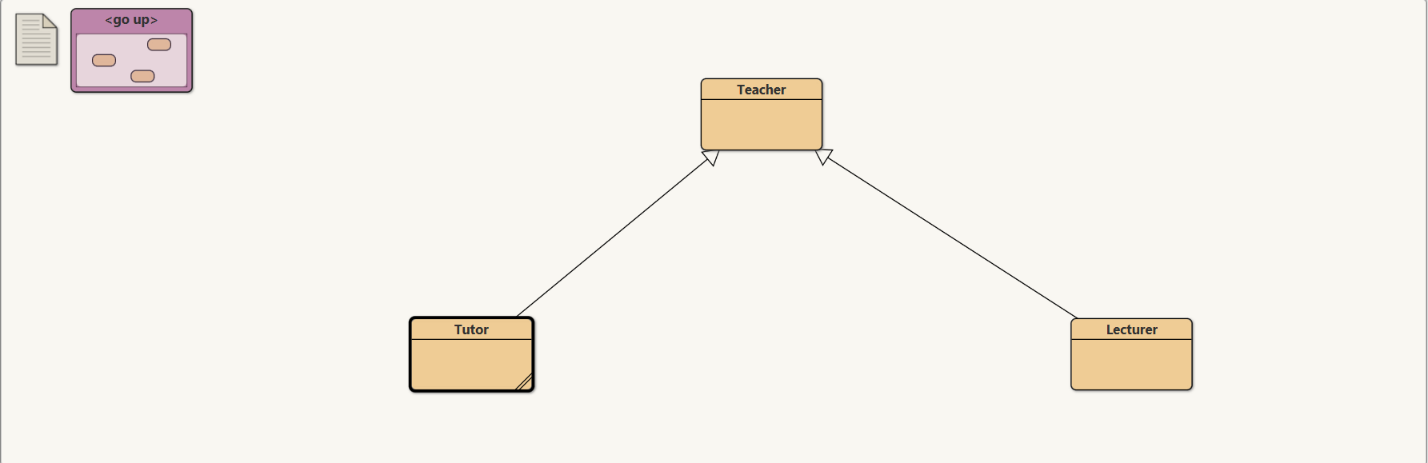


Figure : Classes interface in BlueJ

1. Pseudo Code

Pseudo code is a simple and informal way of representing programming logic that is like a plan English. It’s used to describe the basic steps of an algorithm or program without worrying about specific syntax or programming language rules. It is often used to explain the steps of a solution in a clear way, or to design the structure of a program before coding it. Pseudo code can use natural language, mathematical notation, or a combination of both.

Pseudo code does not have to follow the rules of a programming language, such as indentation, keywords, or data types, but it should be easy to translate into code if needed.

Pseudo code is a useful tool for planning and designing programs, as it allows programmers to focus on the logic of the program without worrying about syntax or language-specific details. It’s a great way to communicate ideas to others and helps to make programs more understandable.

* 1. Pseudo Code of Teacher Class

**CLASS Teacher**

**DO**

**CREATE** teacherId

**CREATE** teacherName

**CREATE** address

**CREATE** workingType

**CREATE** employmentStatus

**CREATE** workingHours

**FUNCTION** Teacher (teacherId, teacherName, address, workingType, employmentStatus)

**DO**

**UPDATE** teacherId

**UPDATE** teacherName

**UPDATE** address

**UPDATE** workingType

**UPDATE** employmentStatus

**UPDATE** workingHours to 0

**END DO**

**FUNCTION** getTeacherId()

**DO**

**RETURN** teacherId

**END DO**

**FUNCTION** getTeacherName()

**DO**

**RETURN** teacherName

**END DO**

**FUNCTION** getAddress()

**DO**

**RETURN** address

**END DO**

**FUNCTION** getWorkingType()

**DO**

**RETURN** workingType

**END DO**

**FUNCTION** getEmploymentStatus()

**DO**

**RETURN** employmentStatus

**END DO**

**FUNCTION** getWorkingHours()

**DO**

**RETURN** workingHours

**END DO**

**FUNCTION** setWorkingHours(workingHours)

**DO**

**UPDATE** workingHours with provided value

**END DO**

**FUNCTION** display()

**DO**

**PRINT** "Teacher ID: " + teacherId

**PRINT** "Teacher Name: " + teacherName

**PRINT** "Address: " + address

**PRINT** "Working Type: " + workingType

**PRINT** "Employment Status: " + employmentStatus

**IF** workingHours > 0

**DO**

**PRINT** "Working Hours: Not assigned"

**END DO**

**ELSE**

**DO**

**PRINT** "Working Hours: " + workingHours

**END DO**

**END FUNCTION**

**END CLASS**

* 1. Pseudo Code of Lecture Class

CLASS Lecturer EXTENDS Teacher

**DO**

**CREATE** department

**CREATE** yearsOfExperience

**CREATE** gradedScore

**CREATE** hasGraded

**FUNCTION** Lecturer (teacherId, teacherName, address, workingType, employmentStatus, department, yearsOfExperience, workingHours)

**DO**

**CALL** super (teacherId, teacherName, address, workingType, employmentStatus)

**CALL** super.setWorkingHours(workingHours)

**UPDATE** department

**UPDATE** yearsOfExperience

**UPDATE** gradedScore to 0

**UPDATE** hasGraded to false

**END DO**

**FUNCTION** getDepartment()

**DO**

**RETURN** department

**END DO**

**FUNCTION** getYearsOfExperience()

**DO**

**RETURN** yearsOfExperience

**END DO**

**FUNCTION** getGradedScore()

**DO**

**RETURN** gradedScore

**END DO**

**FUNCTION** isHasGraded()

**DO**

**RETURN** hasGraded

**END DO**

**FUNCTION** setGradedScore(gradedScore)

**DO**

**UPDATE** gradedScore with provided value

**END DO**

**FUNCTION** gradeAssignment(gradedScore, department, yearsOfExperience)

**DO**

IF yearsOfExperience >= 5 AND this.department EQUALS department

**DO**

IF gradedScore >= 70

**DO**

**CALL** setGradedScore(gradedScore)

**PRINT** "The gradedScore is : A"

**END DO**

ELSE IF gradedScore >= 60

**DO**

**CALL** setGradedScore(gradedScore)

**PRINT** "The gradedScore is : B"

**END DO**

ELSE IF gradedScore >= 50

**DO**

**CALL** setGradedScore(gradedScore)

**PRINT** "The gradedScore is : C"

**END DO**

ELSE IF gradedScore >= 40

**DO**

**CALL** setGradedScore(gradedScore)

**PRINT** "The gradedScore is : D"

**END DO**

**ELSE**

**DO**

**CALL** setGradedScore(gradedScore)

**PRINT** "The gradedScore is : E"

**END DO**

**UPDATE** hasGraded to true

**ELSE**

**PRINT** "It is not graded"

**END IF**

**END**

**FUNCTION** display()

**DO**

**CALL** super.display()

**PRINT** "Department: " + department

**PRINT** "Years Of Experience: " + yearsOfExperience

**IF** hasGraded

**DO**

**PRINT** "Graded Score: " + gradedScore

**END DO**

**ELSE**

**DO**

**PRINT** "Score has not been graded."

**END DO**

**END DO**

**END CLASS**

* 1. Pseudo Code of Tutor class

CLASS Tutor EXTENDS Teacher

**DO**

**CREATE** salary

**CREATE** specialization

**CREATE** academicQualifications

**CREATE** performanceIndex

**CREATE** isCertified

**FUNCTION** Tutor(teacherId, teacherName, address, workingType, employmentStatus, workingHours, salary, specialization, academicQualifications, performanceIndex)

**DO**

**CALL** super(teacherId, teacherName, address, workingType, employmentStatus)

**CALL** super.setWorkingHours(workingHours)

**UPDATE** salary

**UPDATE** specialization

**UPDATE** academicQualifications

**UPDATE** performanceIndex

**UPDATE** isCertified to false

**END** **DO**

**FUNCTION** getSalary()

**DO**

RETURN salary

**END** **DO**

**FUNCTION** getSpecialization()

**DO**

RETURN specialization

**END** **DO**

**FUNCTION** getAcademicQualifications()

**DO**

RETURN academicQualifications

**END DO**

**FUNCTION** getPerformanceIndex()

**DO**

RETURN performanceIndex

**END DO**

**FUNCTION** getIsCertified()

**DO**

RETURN isCertified

**END DO**

**FUNCTION** setSalary(salary, performanceIndex)

**DO**

IF NOT isCertified THEN

**DO**

IF performanceIndex >= 5 AND super.getWorkingHours() > 20 THEN

appraisalPercentage

IF **DECLARE** performanceIndex >= 5 AND performanceIndex <= 7 THEN

**SET** appraisalPercentage to 0.05

ELSE IF performanceIndex >= 8 AND performanceIndex <= 9 THEN

**SET** appraisalPercentage to 0.10

ELSE

**SET** appraisalPercentage to 0.20

**END IF**

**DECLARE** appraisalAmount

**SET** appraisalAmount to salary \* appraisalPercentage

**UPDATE** salary to salary + appraisalAmount

**UPDATE** performanceIndex to performanceIndex

**UPDATE** isCertified to true

**PRINT** "Salary increment"

ELSE

**PRINT** "Uncertified tutor cannot be approved for salary."

ENDIF

ELSE

**PRINT** "Certified tutor cannot be modified for salary."

ENDIF

END

**FUNCTION** removeTutor()

IF NOT isCertified THEN

**UPDATE** salary to 0

**UPDATE** specialization to ""

**UPDATE** academicQualifications to ""

**UPDATE** performanceIndex to 0

**UPDATE** isCertified to false

ELSE

**PRINT** "Certified tutor cannot be removed."

**ENDIF**

**END**

**FUNCTION** display()

**CALL** super.display()

IF NOT isCertified THEN

**PRINT** "Salary: " + salary

**PRINT** "Specialization: " + specialization

**PRINT** "Academic Qualifications: " + academicQualifications

**PRINT** "Performance Index: " + performanceIndex

**ENDIF**

**END**

**END** **CLASS**

1. Method Description

In Java method is a section of code, when it is called, does the operations listed in it. Method is defined once and can be use multiple times. Method description helps in increasing of code readability. Methods description describes about each method in detail.

* 1. Method description of Teacher class

|  |  |
| --- | --- |
| Method Name | Description |
| getTeacherId() | This is an accessor method that retrieves the teacher ID of the current ‘Teacher’ object. |
| getTeacherName() | This is an accessor method that retrieves the name of the current ‘Teacher’ object. |
| getAddress() | This is an accessor method that retrieves the address of the current ‘Teacher’ object. |
| getWorkingType() | This is an accessor method that retrieves the working type (e.g., full-time, part-time) of the current ‘Teacher’ object. |
| getEmploymentStatus() | This is an accessor method that retrieves the employment status of the current ‘Teacher’ object |
| getWorkingHours() | This is an accessor method that retrieves the assigned working hours of the current ‘Teacher’ object |
| setWorkingHours() | This is an mutator method that sets the working hours for the current ‘Teacher’ object. Allows external classes to assign a specific value to the working hours attributes. |
| display() | This is a display method that displays the information about the current ‘Teacher’ object, including teacher ID, name, address, working type, employment status, and working hours (if assigned). If working hours are not assigned (i.e., zero), it indicates that working hours are not assigned. |
| Teacher (int teacherId, String teacherName, String address, String workingType, String employmentStatus) | This is a constructor method for the ‘Teacher’ class. Initializes a new ‘Teacher’ object with the specified parameters, including teacher ID, name, address, working type, and employment status. |

Table : Method description of Teacher class

* 1. Method description of Lecturer class

|  |  |
| --- | --- |
| Method Name | Description |
| Lecturer (int teacherId, String teacherName, String address, String workingType, String employmentStatus, String department, int yearsOfExperience, int workingHours) | This is a Constructor with 8 parameters. |
| getDepartment() | This is an accessor method that retrieves the department of the current ‘Lecturer’ object. |
| getYearsOfExperience() | This is an accessor method that retrieves the years of experience of the current ‘Lecturer’ object. |
| getGradedScore() | This is an accessor method that retrieves the graded score of the current ‘Lecturer’ object. |
| getHasGraded() | This is an accessor method that retrieves a boolean indicating whether the lecturer has graded or not. |
| setGradedScore(int gradedScore) | This is a setter method that sets the graded score for the current ‘Lecturer’ object. |
| gradeAssignment(int gradedScore, String department, int yearsOfExperience) | The gradeAssignment method is responsible for grading an assignment for a lecturer based on specific conditions. It takes three parameters |
| display() | The display method in the Lecturer class is responsible for displaying detailed information about the lecturer. |

Table : Method description of Lecturer class

* 1. Method description of Tutor class

|  |  |
| --- | --- |
| Method Name | Description |
| Tutor(int teacherId, String teacherName, String address, String workingType, String int workingHours, double salary, String specialization, String academicQualifications, int performanceIndex) | This is a Constructor with 9 parameters. |
| getSalary() | This is an accessor method that retrieves the salary of the current ‘Tutor’ object. |
| getSpecialization() | This is an accessor method that retrieves the specialization of the current ‘Tutor’ object. |
| getPerformanceIndex() | This is an accessor method that retrieves the performance index of the current ‘Tutor’ object. |
| getIsCertified() | This is an accessor method that retrieves a Boolean indicating whether the tutor is certified or not. |
| setSalary(double salary, int performanceIndex, int workingHours) | The setSalary method in the Tutor class is responsible for setting the salary for the tutor based on certain conditions. |
| removeTutor() | The removeTutor method in the Tutor class is responsible for removing a tutor's information if the tutor is certified |
| display() | The display method in the Tutor class is responsible for displaying information about the tutor. |

Table : Method description of Tutor class

1. Testing
   1. Test-1

|  |  |
| --- | --- |
| Test | Inspect the Lecturer class, grade the assignment, and re-inspect the Lecturer Class |
| Action | **Creating object Lecturer class:**  TeacherId: 1  TeacherName: Prince Thapa  Address: Itahari  WorkingType: Part-Time  EmploymentStatus: Intern  WorkingHours: 8  Department: IT  YearsOfExperience: 6  workingHours: 8  **Grade the assignment as:**  gradedScore: 95  Department: IT  yearsOfExperience: 8  **Re-inspecting the Lecturer Class after grading the assignment.** |
| Expected Result | Assignment should be graded.  Graded score should be shown. |
| Actual Result | Assignment is graded with its graded marks and score. |
| Conclusion | Test is Successful. |

Table 4: Table of Test-1

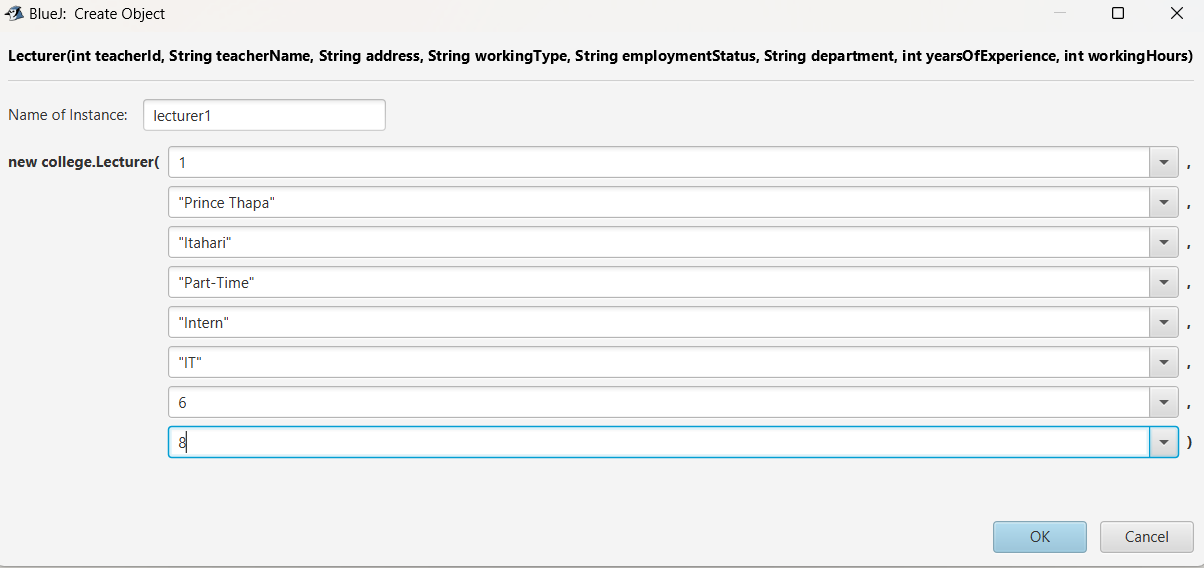


Figure 7: Creating Object of Lecturer

A screenshot of a computer

Description automatically generated

Figure 8: Inspecting object of Lecturer before grading assignment.

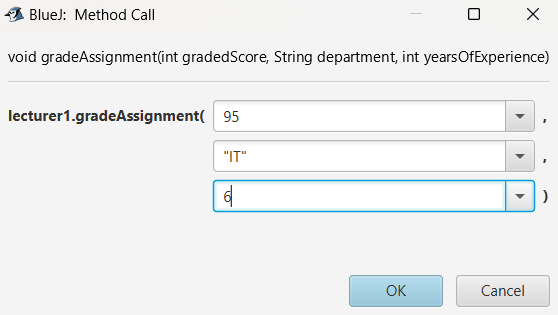


Figure 9: Grading the Assignment

A screenshot of a computer

Description automatically generated

Figure 10: Inspecting Object after grading assignment.

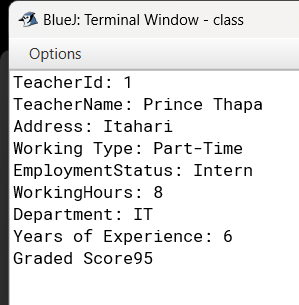


Figure 11: Display of Lecturer class

* 1. Test 2

|  |  |
| --- | --- |
| Test | Inspect Tutor class, set salary and reinspect the Tutor class |
| Action | **Creating Object class Tutor**:  teacherId: 1  teacherName: Prince Thapa  address: Itahari  workingType: Part-Time  employmentStatus: Intern  WorkingHours: 22  salary: 10000  Specialization: Web Developer  academicQualification: Computing  performanceIndex: 24  **Setting Salary as:**  newSalary: 12000  newPerformanceIndex: 25 |
| Expected Result |  |
| Actual Result |  |
| Conclusion | Test is successful. |

Table 5: Table of Test-2

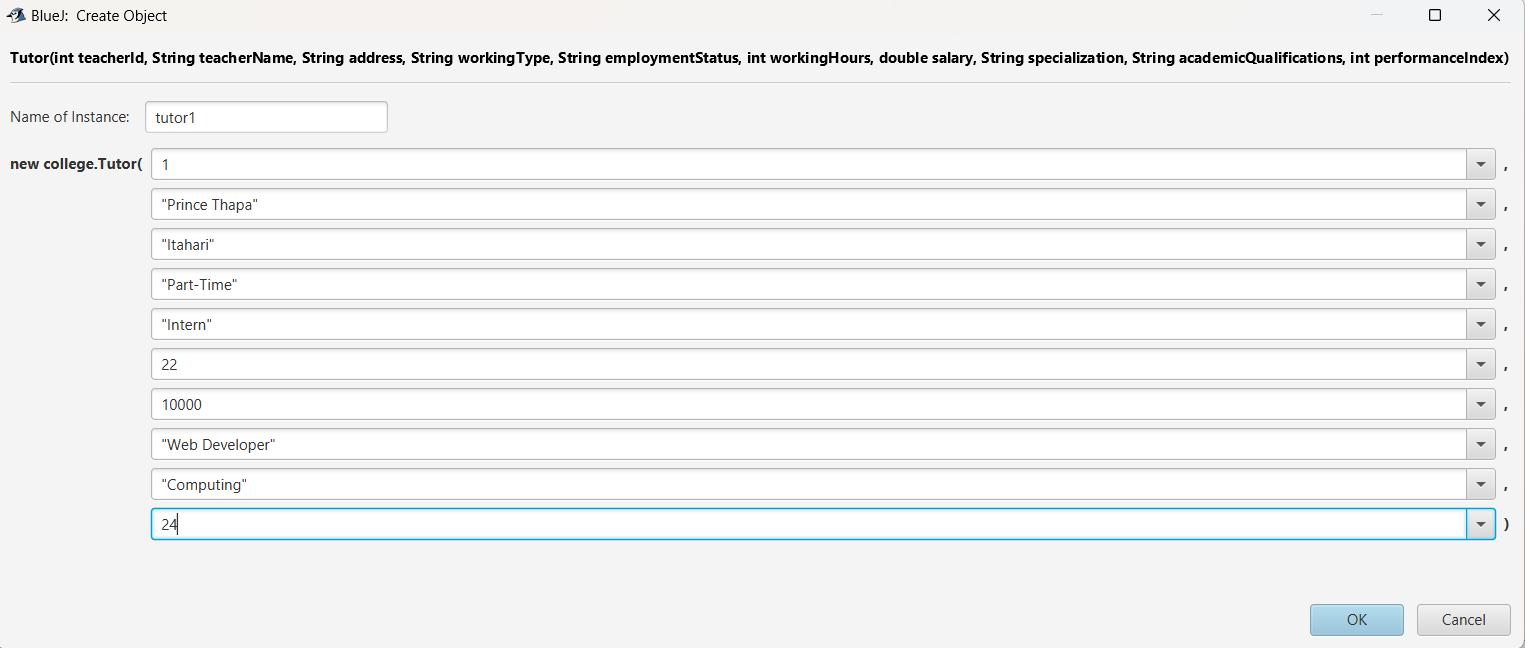


Figure 12: Creating Object of Tutor class.

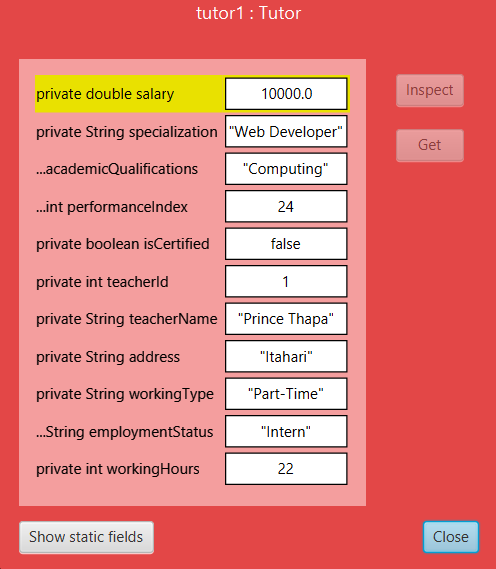


Figure 13: Insepcting of Object of Tutor before setting Salary.

A screenshot of a computer

Description automatically generated

Figure 14: Setting Salary

A screenshot of a computer

Description automatically generated

Figure 15: Re-Inspecting object after setting salary.

A screenshot of a computer

Description automatically generated

Figure 16:Actual display of Tutor Class.

1. Error Detection and correction
   1. Syntax Error:

Everything in a computer is designed in a concrete syntax form. If your input does not match that set of syntax, there are high chances of you facing a syntax error. You can say that syntax error is a mistake in the input by the user therefore, the computer cannot answer the input question.

In my coding while compiling there is syntax error, i.e. there is missing of semicolon while declaring instance variable.

A screenshot of a computer

Description automatically generated

Figure 17: Displaying syntax error in code.

To solve this error I have again put semicolon and compile is now successful.

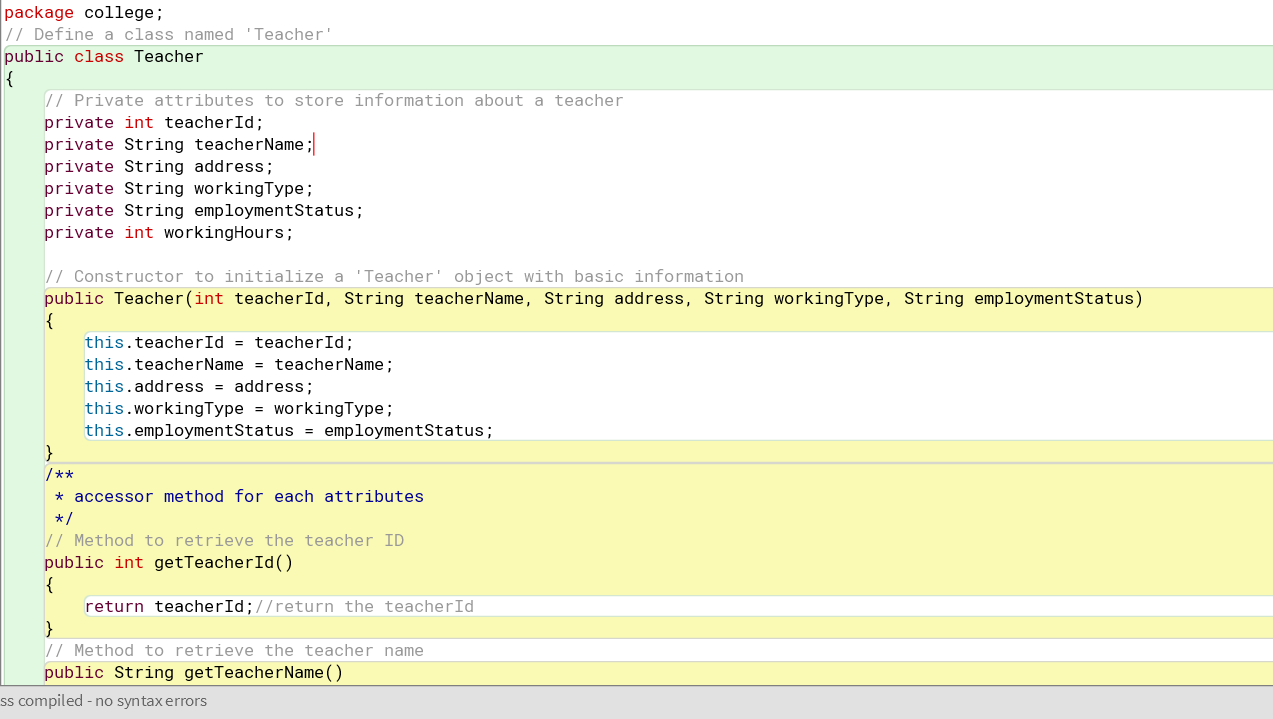


Figure 18: Screenshot after correction of syntax error.

* 1. Semantic error:

The semantic error can be caused by using the incorrect variable, operator, or order of operations. While creating a method named getDepartment()\* I mistakenly wrote a different return type variable which was yearsOfExperience when there supposed to be department. Here is the screenshot of program with wrong return type variable.

A yellow and black sign

Description automatically generated

Figure 19: Displaying semantic error in code.

When I encountered this error\* I simply corrected return type variable with department and this solved the issue. Here is the screenshot of program after changes.

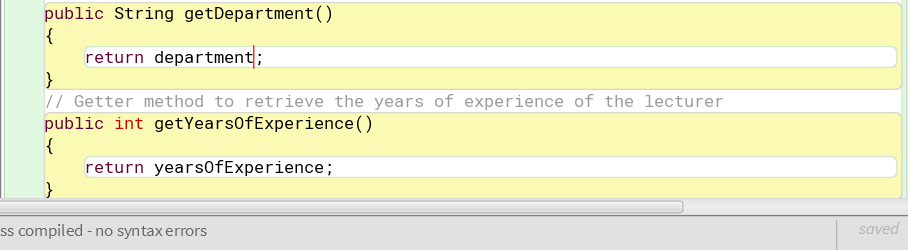


Figure 20: Screenshot after correction of semantic error.

* 1. Logical Error: