LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC4001NI Programming

COURSEWORK-1

Assessment Weightage & Type

30% Individual Coursework

Semester and Year

Spring 2021

Student Name: Aashna Shrestha

Group: C13

London Met ID: 20048800

College ID: NP01CP4S210103

Assignment Due Date: 23rd May, 2021

Assignment Submission Date: 23rd May, 2021

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

Table of Contents

List of Figures

List of Tables

1. Introduction

The project contains a program with three classes i.e., Course, AcademicCourse and NonAcademicCourse. The Course class is a super class of AcademicCourse and NonAcademicCourse. The Course class contains the basic details of the available courses, the AcademicCourse class contains details about the academic courses and the NonAcademicCourse class contains details about the non-academic courses. In all three classes, there is a constructor to initialize different attributes of the classes, accessor/ getter methods, mutator/setter methods and a display method to output the individual course details. In the AcademicCourse class, there is a method to register new course if it has not been registered already. In the NonAcademicCourse class, there are methods to register as well as remove any courses.

2. Class Diagram

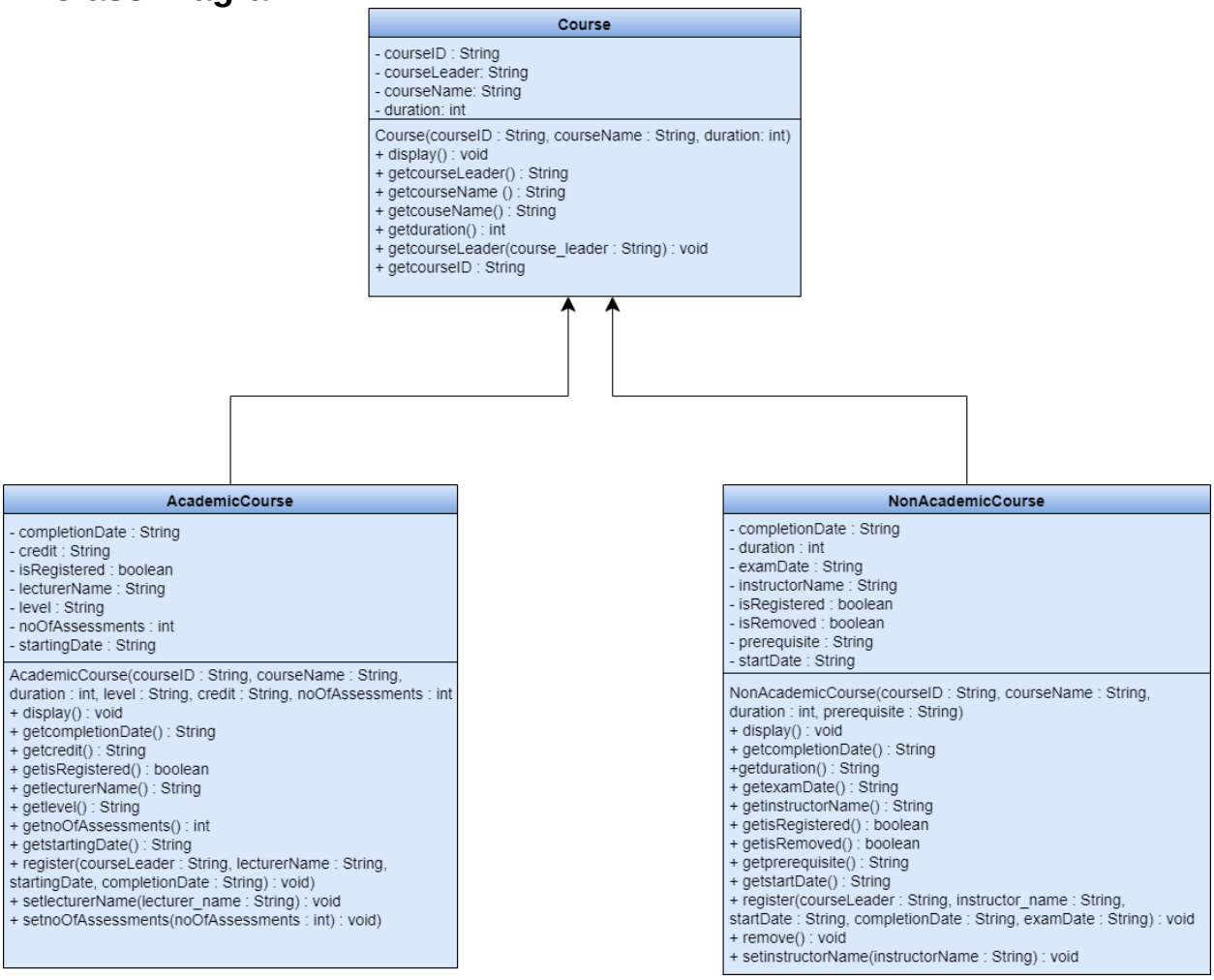

Figure 1 Class Diagram of Course, AcademicCourse and NonAcademicCourse

3. Pseudocode

Course
START

```
CREATE class Course
READ four variables: courseID, courseName, courseLeader, duration
CREATE getcourseID()
DO
        RETURN this.courseID
END DO
CREATE gercourseName()
DO
        RETURN this.courseName
END DO
CREATE getduration()
DO
        RETURN this.duration
END DO
CREATE getcourseLeader()
DO
        RETURN this.courseLeader
END DO
CREATE setcourseLeader(course_leader)
DO
        INITIALIZE this.courseLeader to course_leader
END DO
CREATE DISPLAY
DO
        PRINT “Course ID: ” + getcourseID()
        PRINT “Course Name: ” + getcourseName()
        PRINT “Duration: ” + getduration()
IF course != “”
        PRINT this.courseLeader
```

```
ELSE
        PRINT "The course leader has not been assigned"
ENDIF
END DO
END
```

AcademicCourse

```
START
CREATE class AcademicCourse
READ seven variables: lecturerName, level, credit, startingDate, completionDate,
noOfAssements, isRegistered
CREATE getlecturerName()
DO
        RETURN lecturerName
END DO
CREATE getlevel()
DO
        RETURN this.level
END DO
CREATE getcredit()
DO
        RETURN this.credit
END DO
CREATE getstartingDate()
DO
        RETURN this.startingDate
END DO
CREATE getcompletionDate()
DO
        RETURN this.completionDate
END DO
```

```
CREATE getnoOfAssements()
DO
    RETURN this.noOfAssessments
END DO
CREATE getisRegistered()
DO
    RETURN this.isRegistered
END DO
CREATE register(course_leader, lecturer_name, starting_date, completion_date)
DO
    IF isRegistered == true
        PRINT “You have registered for the course”
        PRINT “Instructor Name: ” + getlecturerName()
        PRINT “Starting Date: ” + getstartingDate()
        PRINT “Completion Date: ” + getcompletionDate()
    ELSE
        CALL Course.setcourseLeader(courseLeader)
        INITIALIZE this.lecturerName to lecturer_name
        INITIALIZE this.startingDate to starting_date
        INITIALIZE this.completionDate to completion_date
        INITIALIZE this.isRegistered to true

        PRINT “You have registered for the course.”
        PRINT “Lecturer: ” + getlecturerName()
        PRINT “Starting Date: ” + getstartingDate()
        PRINT “Completion Date” + getcompletionDate()
    ENDIF
END DO
CREATE display()
DO
    CALL Course.display()
```

```
        IF isRegistered == true
                PRINT "Lecturer: " + getlecturerName()
                PRINT "level:" + getlevel()
                PRINT "credit: " + getcredit()
                PRINT "Starting Date: " + getstartingDate()
                PRINT "Completion Date: " + getcompletionDate()
                PRINT "Number of Assessments: " + getnoOfAssessments()
        ENDIF
END DO
END
```

NonAcademicCourse
START

CREATE class NonAcademicCourse

READ eight variables: instructorName, startDate, completionDate, examDate, prerequisite, duration, isRegistered, isRemoved

```
CREATE getinstructorName()
DO
        RETURN this.instructorName
END DO
CREATE getduration()
DO
        RETURN this.duration
END DO

CREATE getstartDate()
DO
        RETURN this.startDate
END DO
```

```
CREATE getcompletionDate()
DO
        RETURN this.completionDate
END DO

CREATE getexamDate()
DO
        RETURN this.examDate
END DO
CREATE getprerequisite()
DO
        RETURN this.prerequisite
END DO
CREATE getisRegistered()
DO
        RETURN this.isRegistered
END DO
CREATE getisRemoved()
DO
        RETURN this.isRemoved
END DO
CREATE register(courseLeader, instructor_name, startDate, completionDate,
examDate)
DO
        IF isRegistered == false
                CALL setinstructorName(instructor_name)
                INITIALIZE this.isRegistered to true
        ELSE
                PRINT "The course has already been registered. Instructor name can not
                be changed."
        END IF
```

```
END DO
CREATE remove()
DO
        IF isRemoved == true
                PRINT “The course has been removed.”
        ELSE
                CALL Course.setcourseLeader("")
                INITIALIZE this.instructorName to ""
                INITIALIZE this.startDate to ""
                INITIALIZE this.completionDate to ""
                INITIALIZE this.examDate to ""
                INITIALIZE this.isRegistered to false
                INITIALIZE this.isRemoved to true
        END IF
CREATE display()
DO
        CALL Course.display()
        IF isRegistered == true
                PRINT "Instructor Name: " + getinstructorName()
                PRINT "Start Date: " + getstartDate()
                PRINT "Completion Date: " + getcompletionDate()
                PRINT "Exam Date: " + getexamDate()
        ELSE
                PRINT "The course has not been registered."
        END IF
END DO
END
```

4. Method Description

Course

Method	Description
String getcourseID()	Returns the attribute courseID
String getcourseName()	Returns the attribute courseName
int getduration()	Returns the attribute duration
String getcourseLeader()	Returns the attribute courseLeader()
void setcourseLeader(String course_leader)	Sets the attribute courseLeader to the parameter course_leader
void display()	Displays the courseID, courseName and duration. It also displays the courseLeader if it has been set.

Table 1 Method Description of Course class

AcademicCourse

Method	Description
String getlecturerName()	Returns the attribute lecturerName
String getlevel()	Returns the attribute level
String getcredit()	Returns the attribute credit
String getstartingDate()	Returns the attribute startingDate
String getcompletionDate()	Returns the attribute completionDate
int getnoOfAssesments()	Returns the attribute noOfAssessments
boolean getisRegistered()	Returns the attribute isRegistered
void setlecturerName(String lecturer_name)	Sets the attribute lecturerName to the parameter lecturer_name
void setnoOfAssessments(int noOfAssessments)	Sets the attribute noOfAssessments to parameter noOfAssessments

void register(String course_leader,String lecturer_name,String starting_date, String completion_date)	Checks whether a course has been registered. Displays the course details including the lecturer's name, start date and completion. If it has not, then sets the attributes to the respective parameters and displays the course details.
void display()	Calls the display method from the super class. It also displays the course details including the lecturer's name, level, credit, starting date, completion date and the number of assessments.

Table 2 Method Description of AcademicCourse class

NonAcademicCourse

Method	Description
String getinstructorName()	Returns the attribute instructorName
int duration()	Returns the attribute duration
String getstartDate()	Returns the attribute startDate
String getcompletionDate()	Returns the attribute completionDate
String getexamDate()	Returns the attribute examDate
String prerequisite()	Returns the attribute prerequisite
boolean getisRegistered()	Returns the attribute isRegistered
boolean getisRemoved()	Returns the attribute isRemoved
void setinstructorName(instructor_name)	Sets the attribute instructorName to parameter instructor_name if the course has been registered.

void register(String courseLeader, String instructor_name, String startDate,String completionDate,String examDate)	Checks if the course has been registered. If it has not been registered, then calls the method setinstructorName(instructor_name) to set the attribute instructorName to the parameter instructor_name.
void remove()	Checks if the course has been removed, and removes it if it has not been removed.
void display()	Calls the display method from the super class. It also displays the course details including the instructor's name, start date, completion date and exam date if the course has been registered.

Table 3 Method Description of NonAcademicCourse class

5. Testing

Test 1 - To inspect the class AcademicCourse, register a course and re-inspect the class

Test No.	1
Objective	To inspect the class AcademicCourse, register a course and re-inspect the class.

Action	• The constructor for the AcademicCourse class is called with the following parameters: courseID = “1” courseName = “Computing” duration = 5 level = “3” credit = “4” noOfAssessments = 3 • Inspection of the class AcademicCourse. • The method remove is called with the given parameters: course_leader = “Lisa Nielson” lecturer_name = “Nial Thomas” starting_date = “05/01/2021” completion_date = “10/06/2021” • The class is re-inspected.
Expected Result	The academic course will be registered.
Output	The course is registered.
Conclusion	The test has been completed successfully.

Table 4 Test to inspect the class AcademicCourse, register a course and re-inspect the class

Output Result:

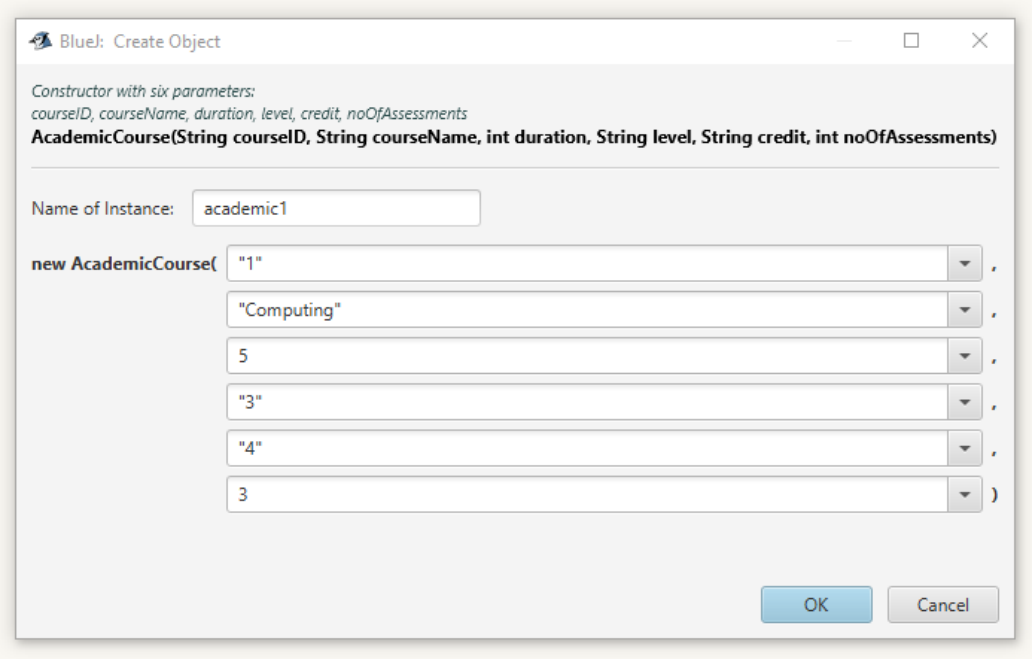

Figure 2 Assigning values to attributes in AcademicCourse class

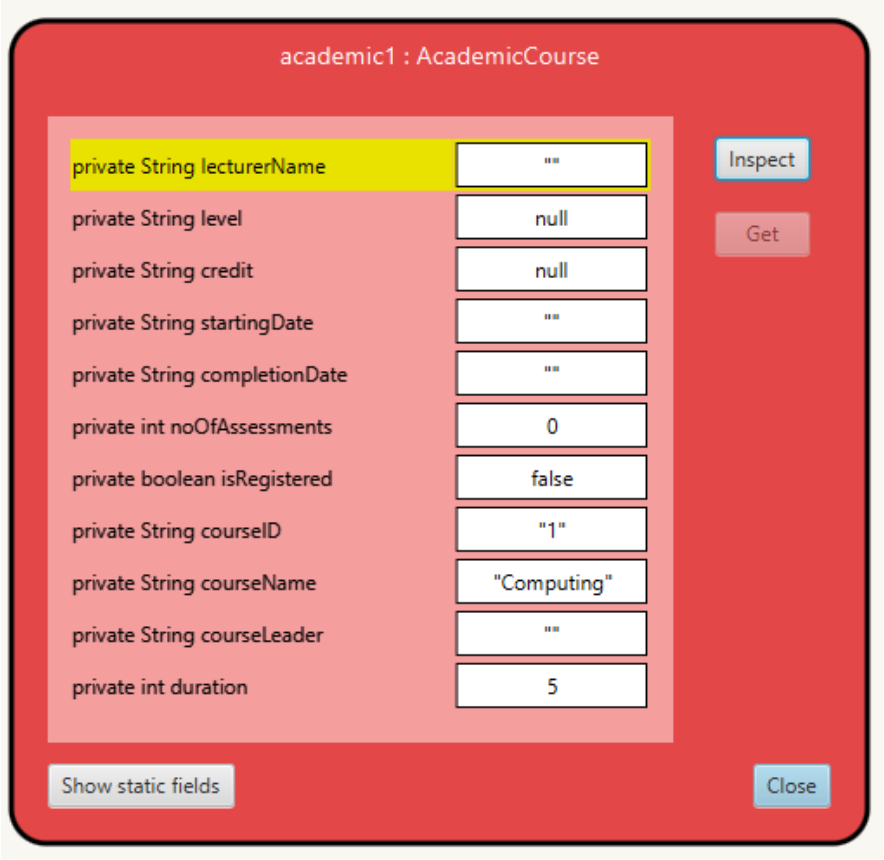

Figure 3 Inspection of AcademicCourse class

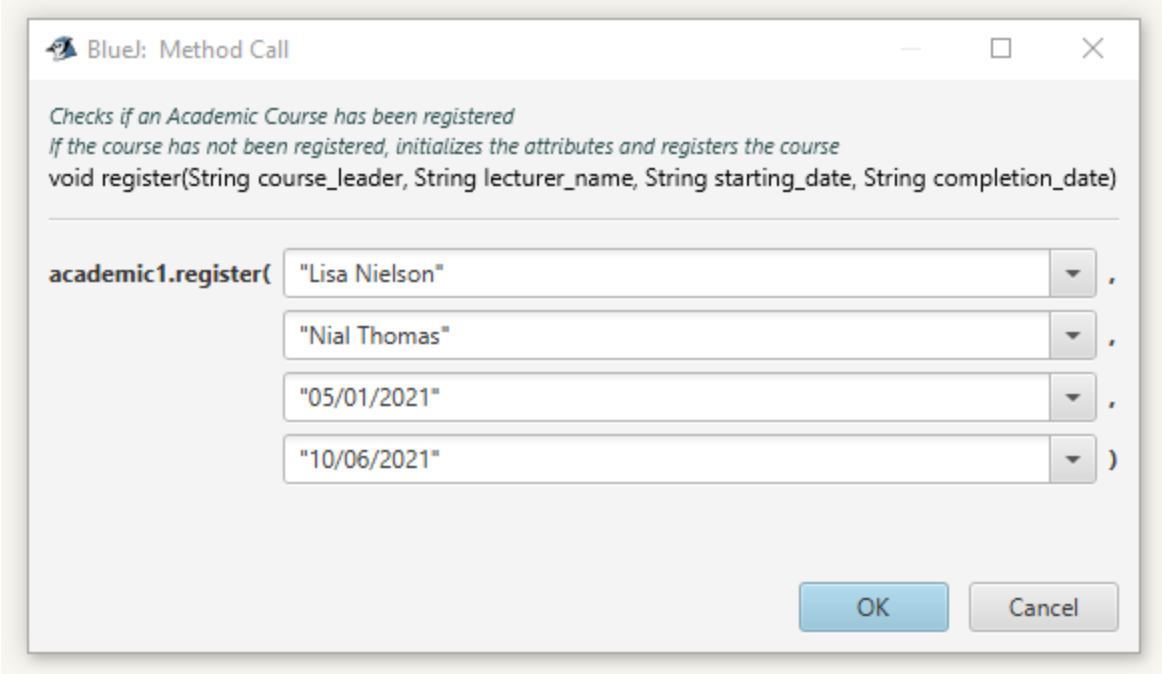

Figure 4 Calling the register method and assigning values to the parameters

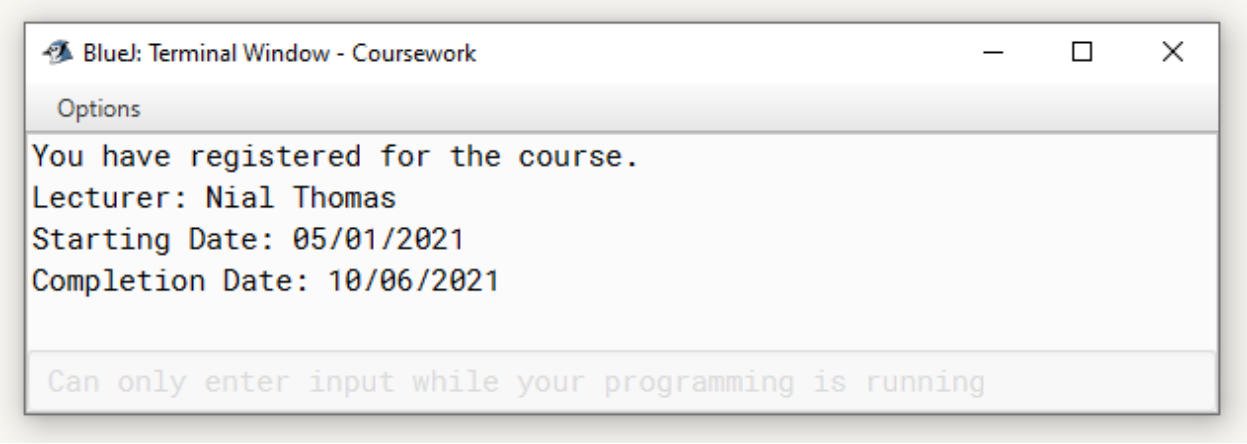

Figure 5 Output of register method

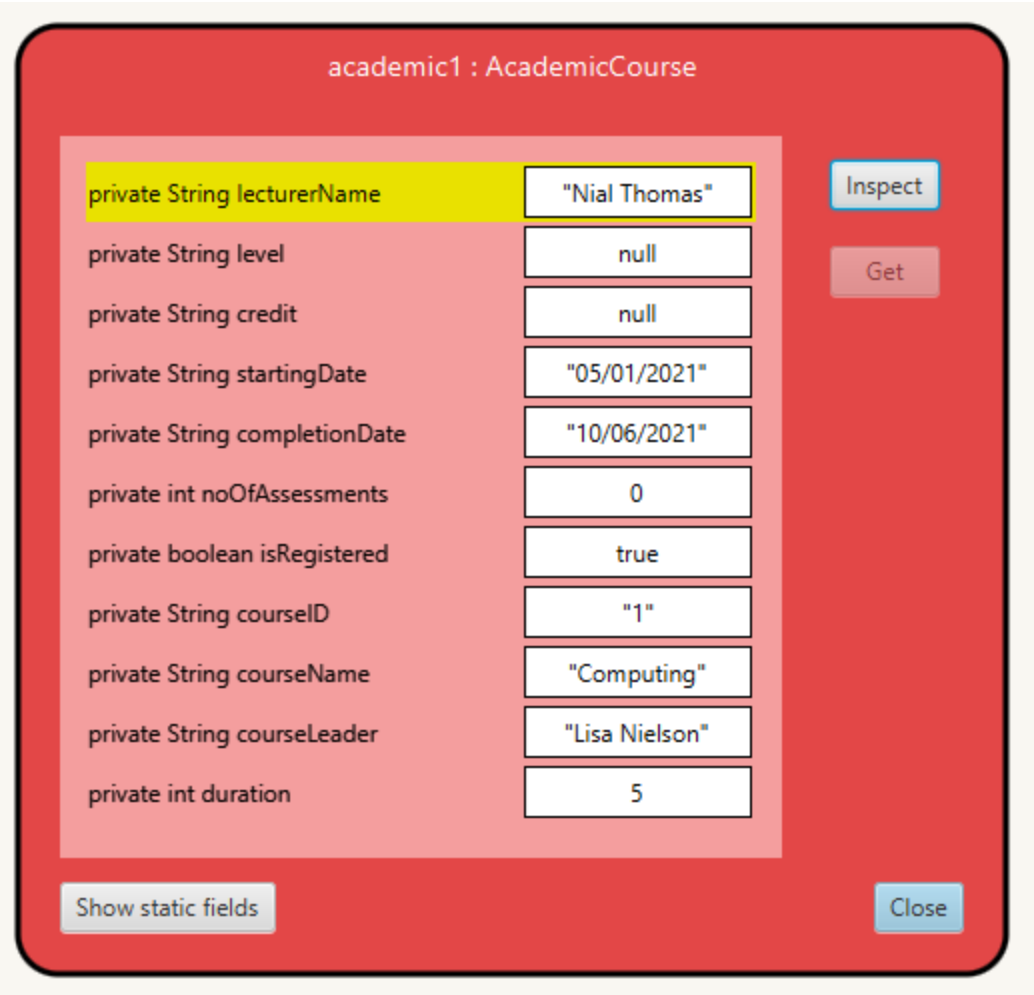

Figure 6 Inspection of AcademicCourse class after registering a course

Test 2 - To inspect the class NonAcademicCourse, register a course and re-inspect the class

Test No.	2
Objective	To inspect the class NonAcademicCourse, register a course and re-inspect the class
Action	• The constructor of NonAcademicCourse class is called with the following parameters: courseID = “2” courseName = “Presentation Skills” duration = 3

	prerequisite = “Students must be familiar with the use of MS Powerpoint” • Inspection of the class NonAcademicCourse • The method register is called with the following parameters: courseLeader = “Alessa Brown” instructor_name = “Rick Hardy” startDate = “05/06/2021” completionDate = “11/09/2021” examDate = “20/09/2021” • The class is re-inspect
Expected Result	The non - academic course will be registered.
Output	The course is registered.
Conclusion	The test has been completed successfully.

Table 5 Test to inspect the class NonAcademicCourse, register a course and re-inspect the class

Output Result:

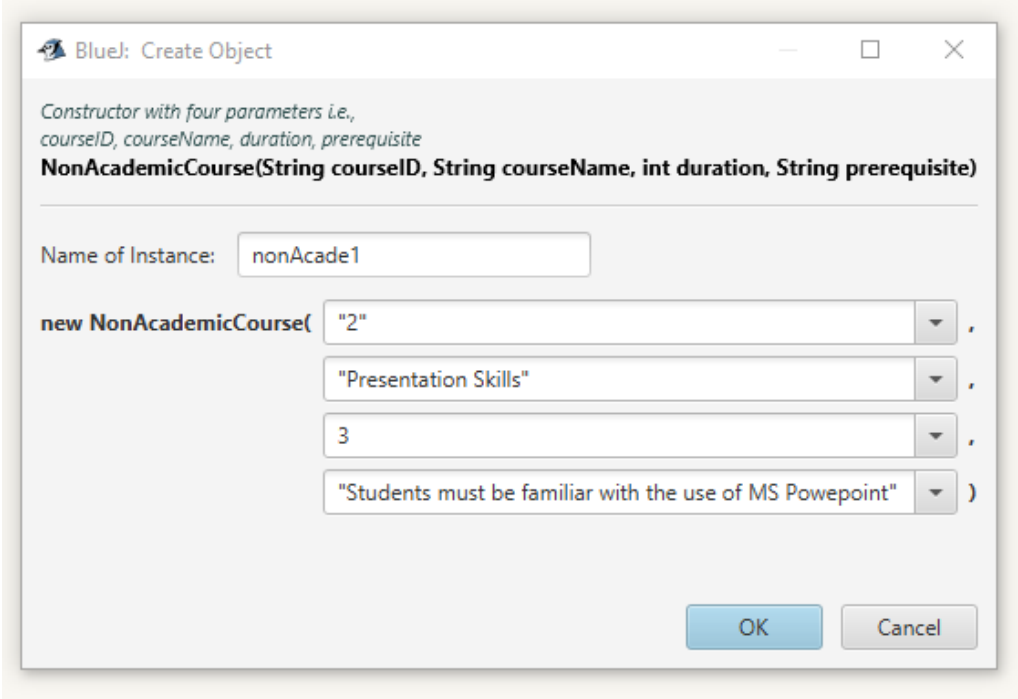

Figure 7 Assigning values to attributes in NonAcademicCourse class

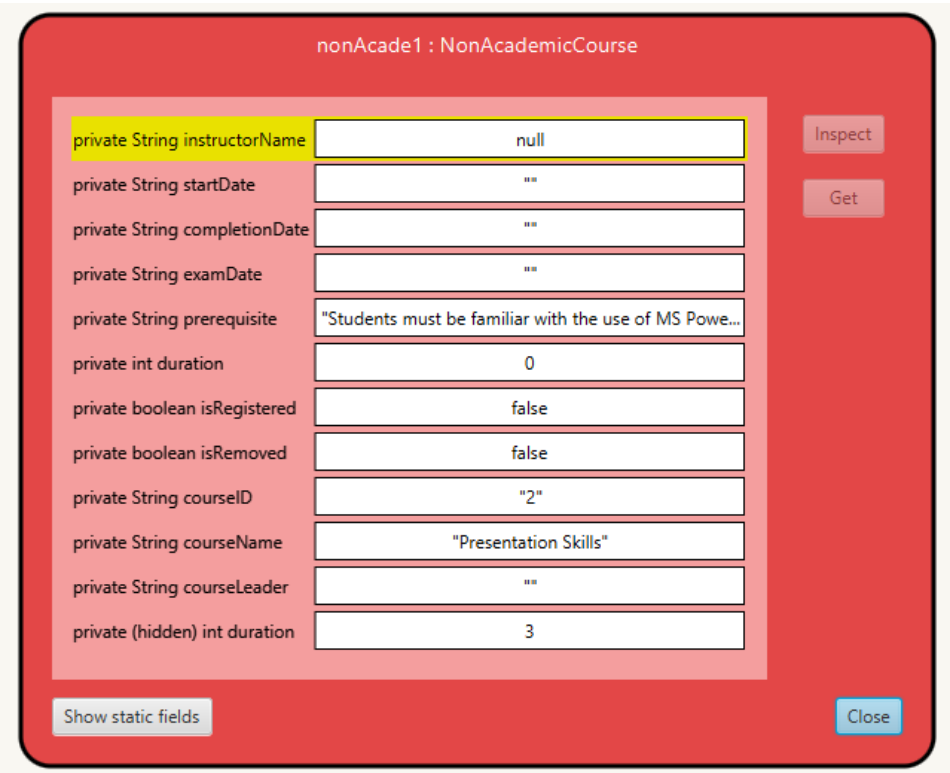

Figure 8 Inspection of NonAcademicCourse class

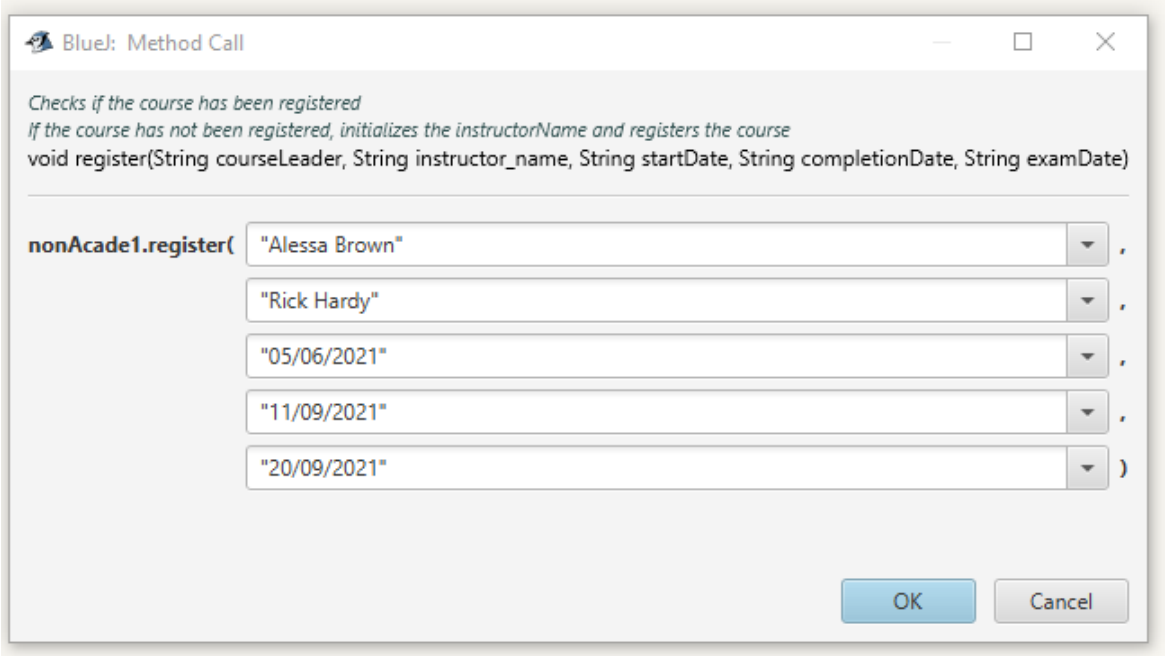

Figure 9 Calling the register method of the NonAcademicCourse class and assigning values to the parameters

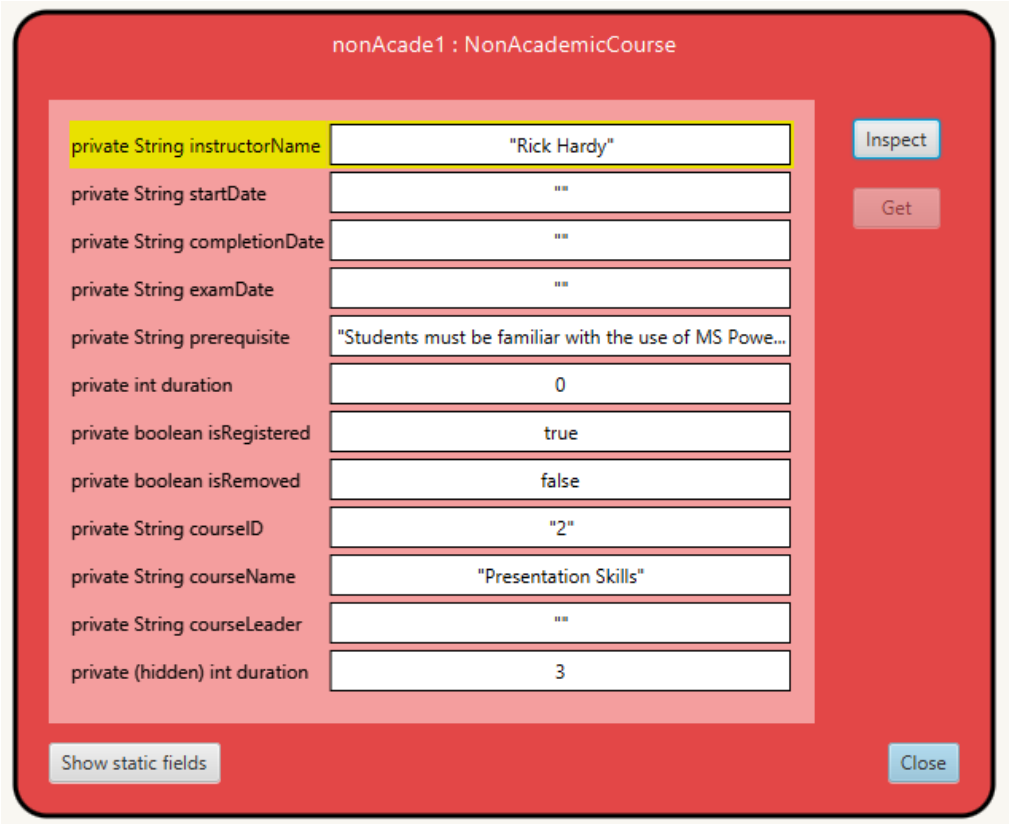

Figure 10 Re-inspecting the NonAcademicCourse class after registering a course

Test 3 - To remove a course and inspect the NonAcademicCourse class

Test No.	3
Objective	To remove a course and inspect the NonAcademicCourse class.
Action	• The NonAcademicCourse class is inspected. • The remove() method is called. • The class is re-inspected.
Expected Result	The non-academic course will be removed.
Output	The non-academic course is removed.
Conclusion	The test has been completed successfully.

Table 6 Test to remove a course and inspect the NonAcademicCourse class

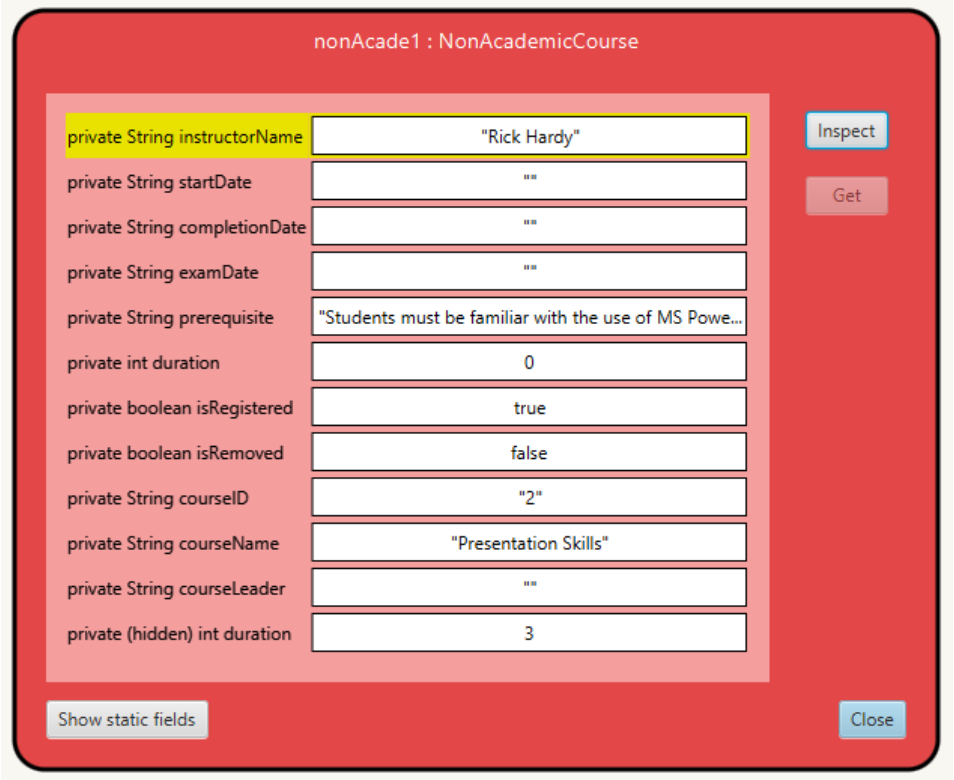

Figure 11 Inspecting the NonAcademicCourse class after registering a course

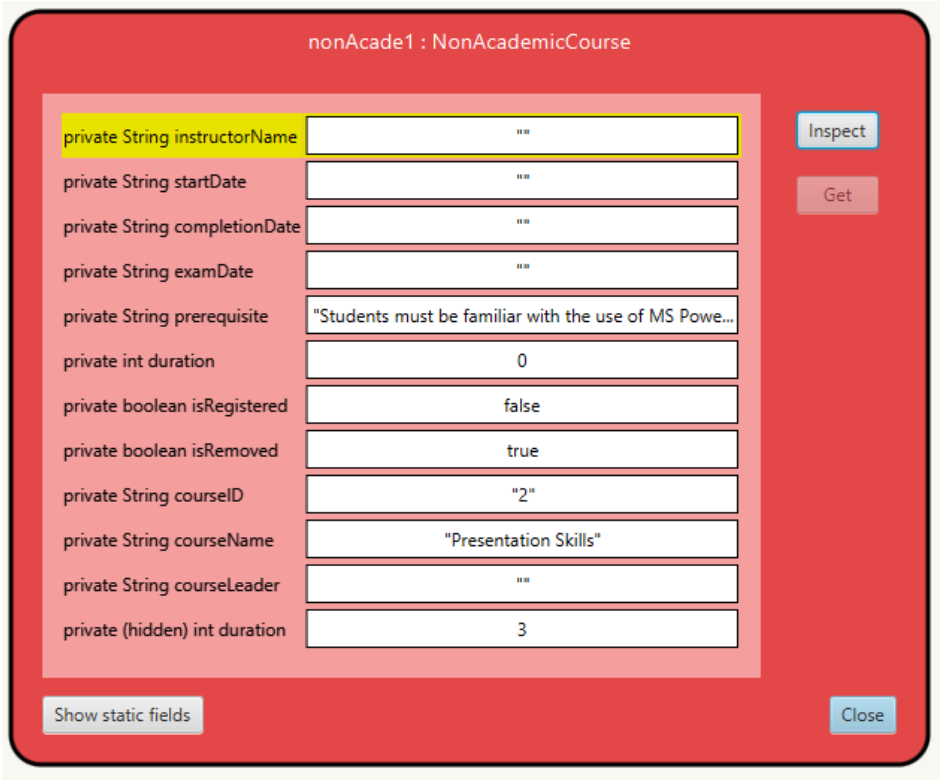

Figure 12 Re-inspecting the NonAcademicCourse class after removing a course

Test 4 - To display the course details of AcademicCourse and NonAcademicCourse class

Test No.	4
Objective	To display the course details of AcademicCourse and NonAcademicCourse class.
Action	• Call the display() method of AcademicCourse class • Call the display() method of NonAcademicCourse class
Expected Result	The details of academic course and non-academic course will be displayed.
Output	The details of academic course and non-academic course is displayed.
Conclusion	The test has been completed successfully.

Table 7 Test to display the course details of AcademicCourse and NonAcademicCourse class

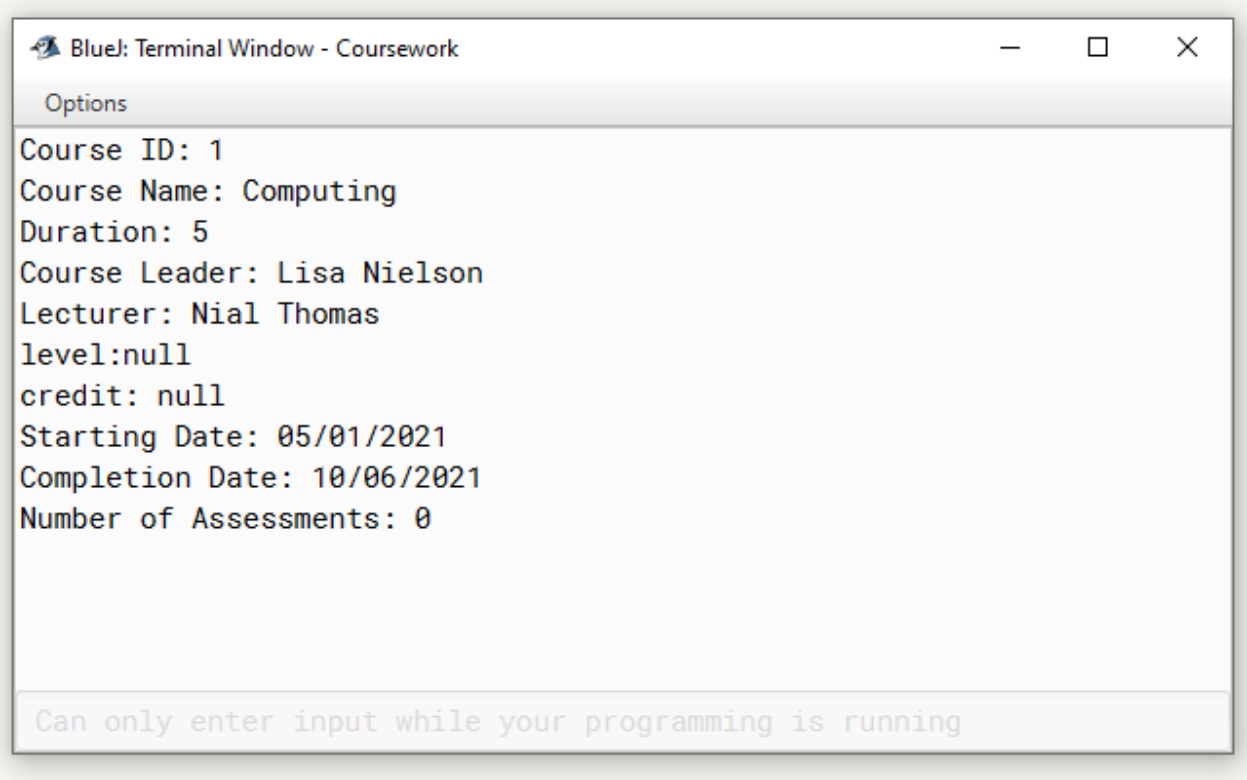

Figure 13 Calling the display() method of the AcademicCourse class

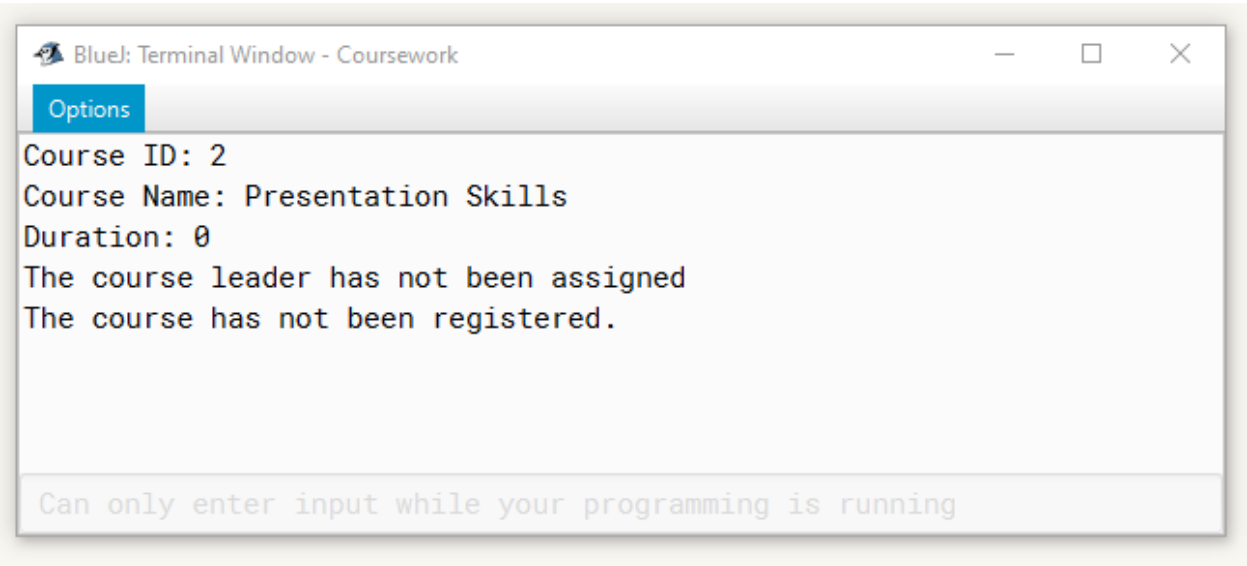

Figure 14 Calling the display() method of the NonAcademicCourse class

6. Error

Syntax Error

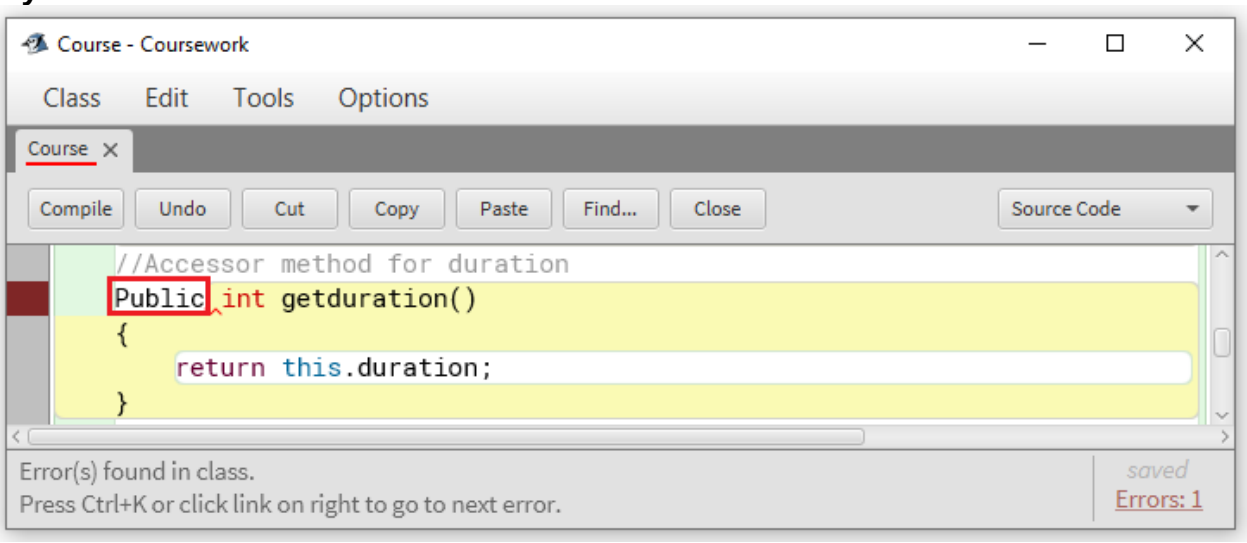

Figure 15 Syntax Error

The keyword public has been capitalized. The program does not compile since the wrong syntax has been used.

Solution: The keyword should be written in lower case i.e. 'public' instead of 'Public'.

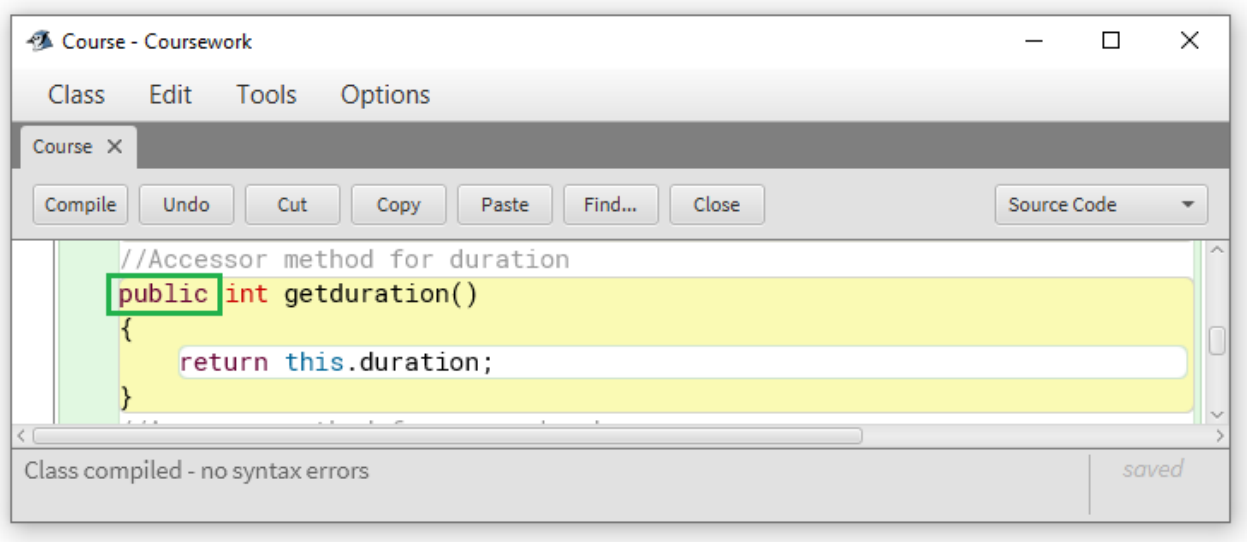

Figure 16 Correction of Syntax Error

Semantic Error

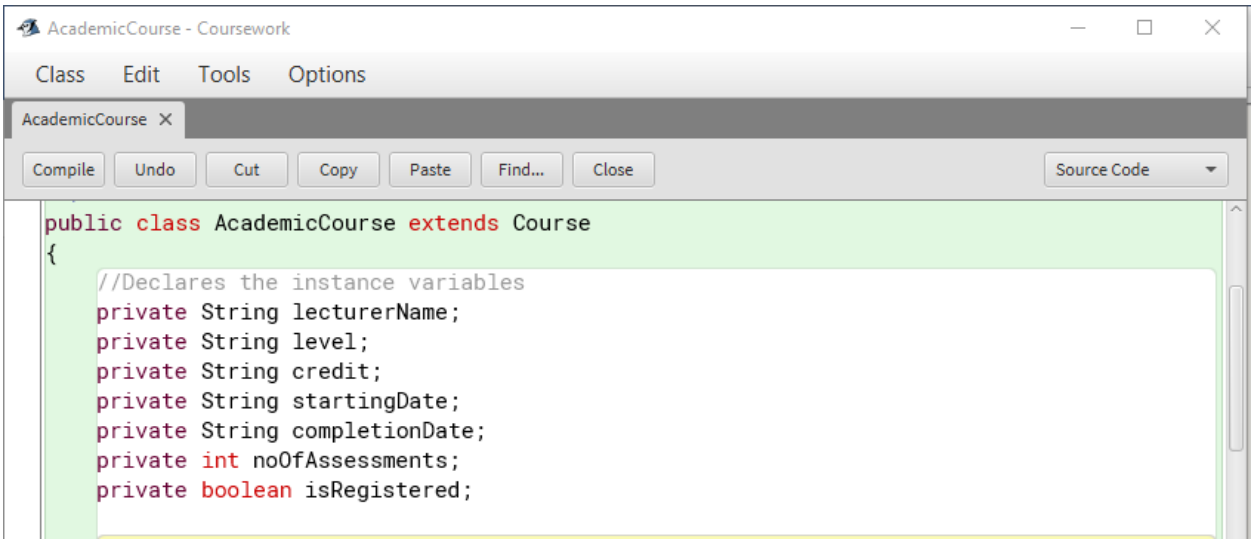

Figure 17 Declaration of instance variables

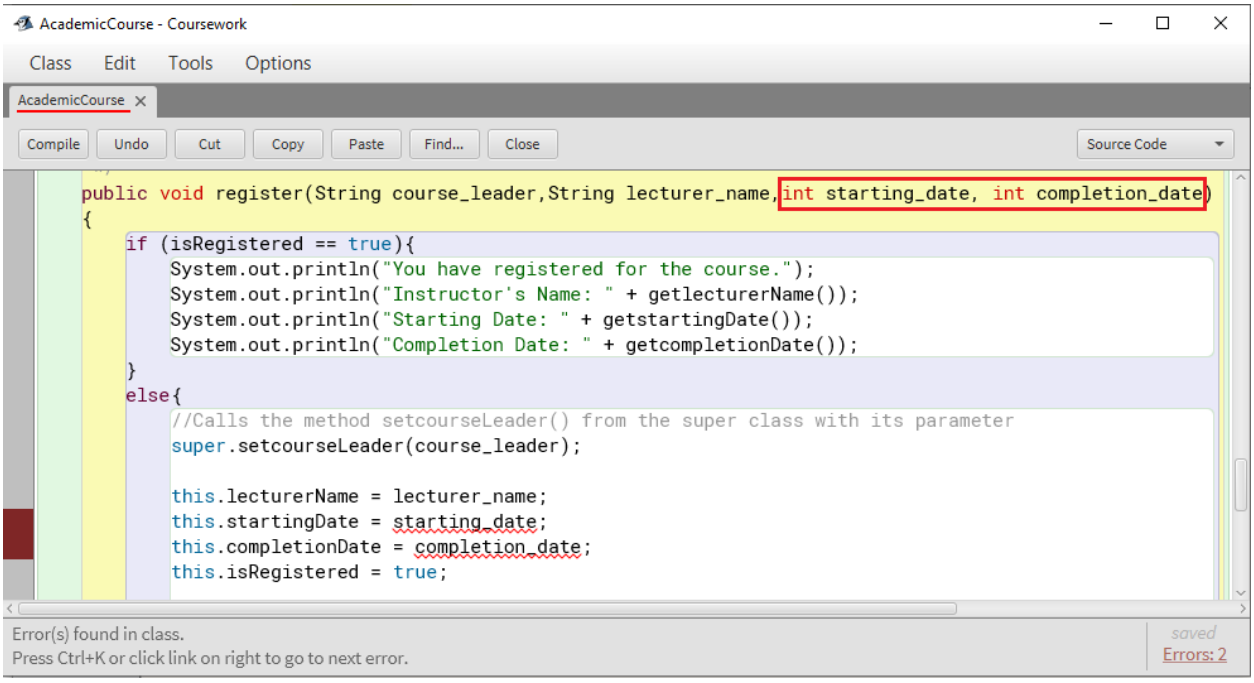

Figure 18 Semantic Error

The instance variables startingDate and completionDate have been declared as String. In the register method, they are initialized to the parameters starting_date and completion_date respectively; however, the variables starting_date and completion_date have been declared as int.

Solution: The data that is being initialized to a variable must have the same data type as the variable. Here, starting_date and completion_date must be declared as String.

```
public void register(String course_leader,String lecturer_name,String starting_date, String completion_date)
{
    if (isRegistered == true){
        System.out.println("You have registered for the course.");
        System.out.println("Instructor's Name: " + getlecturerName());
        System.out.println("Starting Date: " + getstartingDate());
        System.out.println("Completion Date: " + getcompletionDate());
    }
    else{
        //Calls the method setcourseLeader() from the super class with its parameter
        super.setcourseLeader(course_leader);

        this.lecturerName = lecturer_name;
        this.startingDate = starting_date;
        this.completionDate = completion_date;
        this.isRegistered = true;
```

Figure 19 Correction of Semantic Error

Logic Error

```
/* Constructor with four parameters i.e.,
 * courseID, courseName, duration, prerequisite
   */
NonAcademicCourse(String courseID, String courseName, int duration, String prerequisite)
{
    super(courseID, courseName, duration);
    this.instructorName = prerequisite;
    this.startDate = "";
    this.completionDate = "";
    this.examDate = "";
    this.isRegistered = false;
    this.isRemoved = false;
}
```

Figure 20 Logic Error

The parameter prerequisite has been assigned to the instance variable instructorName. The program has been compiled successfully; however, the program is logically incorrect. The instuctorName must be assigned with the instructor's name and the prerequisite of the course should be assigned to the attribute prerequisite.

Solution: The parameter prerequisite should be assigned to the instance variable prerequisite.

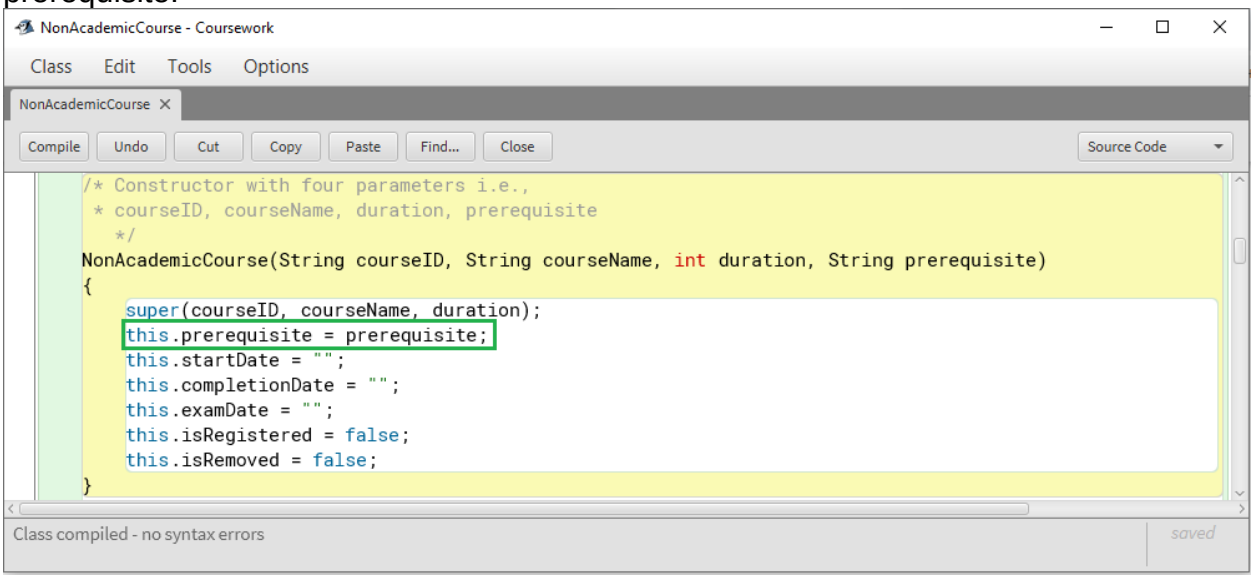

Figure 21 Correction of Logic Error

7. Conclusion

The project includes java codes for three classes i.e., Course, AcademicCourse and NonAcademicCourse. The codes were written according to the guidelines provided. Along with the program codes, a class diagram has been prepared for all three classes. The class diagram shows the different attributes, their data type, methods, parameters accepted by the methods, return type of each method and the type of access modifiers used. Pseudocode has also been written for the program, in which each line of the program has been explained.

The program was compiled after being written. Three types of errors were found in the program: syntax error, semantic error and logic error. The errors have been corrected and the program has been compiled successfully. Four tests were performed for the project. In the first and the second tests, the AcademicCourse class and the NonAcademic course class were inspected. The register() method was called from both the classes and they were re-inspected. In the third test, the remove() method was called from the NonAcademicCourse class, then the class was inspected again. In the last test, the display() method from both AcademicCourse class and

NonAcademicCourse class were called to print the final output. The expected results of each test were obtained. Hence, all the tests were carried out successfully.

The project helped to show different approaches to a program. It clarified the use of variables, methods, classes, and various syntaxes. There were difficulties while compiling the program due to frequent errors. Some codes compiled well, but did not give the required output. The project helped to identify such errors and deal with them.

8. Appendix

Course.java

```
public class Course
{
    //Declares the instance variables
    private String courseID;
    private String courseName;
    private String courseLeader;
    private int duration;

    /* Constructor with 3 parameters: courseID, courseName, duration
     * Initialize the attributes */
    Course(String courseID, String courseName, int duration)
    {
        this.courseID = courseID;
        this.courseName = courseName;
        this.duration = duration;
        this.courseLeader = "";
    }
    //Accessor method for courseID
    public String getcourseID()
    {
        return this.courseID;
    }
    //Accessor method for courseName
    public String getcourseName()
    {
        return this.courseName;
    }
    //Accessor method for duration
    public int getduration()
```

```
    {
        return this.duration;
    }
    //Accessor method for courseLeader
    public String getcourseLeader()
    {
        return this.courseLeader;
    }
    //Mutator method for courseLeader
    public void setcourseLeader(String course_leader)
    {
        this.courseLeader = course_leader;
    }
    //Displays the course details and name of the course leader if the leader has been
assigned
    public void display()
    {
        System.out.println("Course ID: " + getcourseID());
        System.out.println("Course Name: " + getcourseName());
        System.out.println("Duration: " + getduration());
        if (courseLeader != "")
        {
        System.out.println("Course Leader: " + getcourseLeader());
        }
        else
        {
            System.out.println("The course leader has not been assigned");
        }
    }
}
```

AcademicCourse.java

```
public class AcademicCourse extends Course
{
    //Declares the instance variables
    private String lecturerName;
    private String level;
    private String credit;
    private String startingDate;
    private String completionDate;
    private int noOfAssessments;
    private boolean isRegistered;

    /* Constructor with six parameters:
     * courseID, courseName, duration, level, credit, noOfAssessments
     */
    AcademicCourse(String courseID, String courseName, int duration,String level,
String credit, int noOfAssessments)
    {
        // Calls the super class, Course
        super(courseID, courseName, duration);

        //Initialize the instance variables
        this.lecturerName ="";
        this.startingDate = "";
        this.completionDate = "";
        this.isRegistered = false;
    }
    //Accessor method of lecturerName
    public String getlecturerName()
    {
        return this.lecturerName;
```

```
    }
    //Accessor method of level
    public String getlevel()
    {
        return this.level;
    }
    //Accessor method of credit
    public String getcredit()
    {
        return this.credit;
    }
    //Accessor method of startingDate
    public String getstartingDate()
    {
        return this.startingDate;
    }
    //Accessor method of completionDate
    public String getcompletionDate()
    {
        return this.completionDate;
    }
    //Accessor method of noOfAssessments
    public int getnoOfAssessments()
    {
        return this.noOfAssessments;
    }
    //Accessor method of isRegistered
    public boolean getisRegistered()
    {
        return this.isRegistered;
    }
```

```
    //Mutator method of lecturerName
    public void setlecturerName(String lecturer_name)
    {
        this.lecturerName = lecturer_name;
    }
    //Mutator method of noOfAssessments
    public void setnoOfAssessments(int noOfAssessments)
    {
        this.noOfAssessments = noOfAssessments;
    }
    /* Checks if an Academic Course has been registered
     * If the course has not been registered, initializes the attributes and registers the
course
     */
    public void register(String course_leader,String lecturer_name,String
starting_date, String completion_date)
    {
        if (isRegistered == true){
            System.out.println("You have registered for the course.");
            System.out.println("Instructor's Name: " + getlecturerName());
            System.out.println("Starting Date: " + getstartingDate());
            System.out.println("Completion Date: " + getcompletionDate());
        }
        else{
            //Calls the method setcourseLeader() from the super class with its parameter
            super.setcourseLeader(course_leader);

            this.lecturerName = lecturer_name;
            this.startingDate = starting_date;
            this.completionDate = completion_date;
            this.isRegistered = true;
```

```
            System.out.println("You have registered for the course.");
            System.out.println("Lecturer: " + getlecturerName());
            System.out.println("Starting Date: " + getstartingDate());
            System.out.println("Completion Date: " + getcompletionDate());
        }
    }
    //Displays the course details
    public void display()
    {
        //Calls the display() method from the super class
        super.display();
        if (isRegistered == true)
        {
        System.out.println("Lecturer: " + getlecturerName());
        System.out.println("level:" + getlevel());
        System.out.println("credit: " + getcredit());
        System.out.println("Starting Date: " + getstartingDate());
        System.out.println("Completion Date: " + getcompletionDate());
        System.out.println("Number of Assessments: " + getnoOfAssessments());
        }
    }
}
```

NonAcademicCourse.java

```
public class NonAcademicCourse extends Course
{
    //Declares the instance variables
    private String instructorName;
    private String startDate;
    private String completionDate;
    private String examDate;
    private String prerequisite;
    private int duration;
    private boolean isRegistered;
    private boolean isRemoved;

    /* Constructor with four parameters i.e.,
     * courseID, courseName, duration, prerequisite
      */
    NonAcademicCourse(String courseID, String courseName, int duration, String
prerequisite)
    {
        super(courseID, courseName, duration);
        this.prerequisite = prerequisite;
        this.startDate = "";
        this.completionDate = "";
        this.examDate = "";
        this.isRegistered = false;
        this.isRemoved = false;
    }
    //Accessor method of instructor method
    public String getinstructorName()
    {
```

```
        return this.instructorName;
    }
    //Accessor method of duration
    public int getduration()
    {
        return this.duration;
    }
    //Accessor method of startDate
    public String getstartDate()
    {
        return this.startDate;
    }
    //Accessor method of completionDate
    public String getcompletionDate()
    {
        return this.completionDate;
    }
    //Accessor method of examDate
    public String getexamDate()
    {
        return this.examDate;
    }
    //Accessor method of prerequisite
    public String getprerequisite()
    {
        return this.prerequisite;
    }
    //Accessor method of isRegistered
    public boolean getisRegistered()
    {
        return this.isRegistered;
```

```
    }
    //Accessor method of isRemoved
    public boolean getisRemoved()
    {
        return this.isRemoved;
    }
    /*Mutator method of instructorName
     *Initializes the instructorName if the course has not been registered
     */
    public void setinstructorName(String instructor_name)
    {
        if (isRegistered == false){
            this.instructorName = instructor_name;
        }
        else{
            System.out.println("Update failed. Changing the instructor name is not
possible.");
        }
    }
    /* Checks if the course has been registered
     * If the course has not been registered, initializes the instructorName and
registers the course
       */
    public void register(String courseLeader, String instructor_name, String
startDate,String completionDate,String examDate)
    {
        if (isRegistered == false){
            // Calls the method setinstructorName with its parameter.
            setinstructorName(instructor_name);
            this.isRegistered = true;
        }
```

```
        else{
            System.out.println("The course has already been registered. Instructor name
can not be changed.");
        }
    }
    //Removes the course
    public void remove()
    {
        if(isRemoved == true){
            System.out.println("The course has been removed.");
        }
        else{
            //Calls the setcourseLeader() method from the super class with its parameter
            super.setcourseLeader("");

            //Initailize the instance variables to remove the course
            this.instructorName = "";
            this.startDate = "";
            this.completionDate = "";
            this.examDate = "";
            this.isRegistered = false;
            this.isRemoved = true;
        }
    }
    //Displays the details of the course
    public void display()
    {
        //Calls the display() method from the super class.
        super.display();
        if (isRegistered == true){
            System.out.println("Instructor Name: " + getinstructorName());
```

```
            System.out.println("Start Date: " + getstartDate());
            System.out.println("Completion Date: " + getcompletionDate());
            System.out.println("Exam Date: " + getexamDate());
        }
        else{
            System.out.println("The course has not been registered.");
        }
    }
}
```