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: Rabina Shrestha

Group: C13

London Met ID: 20049416

College ID: NP01CP4S210039

Assignment Due Date: 23rd May 2021.

Assignment Submission Date: 21st 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 submissions will be treated as non-submission and a mark of zero will be awarded.

Contents

1. Introduction. 1

2. Class Diagram. 2

2.1 Class Diagram of Course. 3

2.2 Class Diagram of Academic Course. 3

2.3 Class Diagram Of Non-Academic Course. 4

3. Pseudocode. 5

3.1 Pseudocode of the class Course: 6

3.2 Pseudocode of the class AcademicCourse: 8

3.3 Pseudocode of the class NonAcademicCourse: 10

4. Method Description. 14

4.1 Course.java 14

4.2 AcademicCourse.java 15

4.3 NonAcademicCourse.java 16

5. Testing (Inspection) 18

5.1 Test 1: Inspect AcademicCourse class, register an academic course, and re-inspect the AcademicCourse Class. 18

Output: 19

5.2 Test 2: Inspect NonAcademicCourse class, register a non-academic course and re-inspect the NonAcademicCourse Class. 21

Output: 22

5.3 Test 3: Inspect NonAcademicCourse class again, change the status of 24

isRemoved to true and re-inspect the NonAcademicCourse class. 24

Output: 25

5.4 Test 4: Display the detail of AcademicCourse and NonAcademicCourse class. . 27

Output: 28

6. Errors: 30
6.1. Error 1: Syntax Error. 30
Finding the error: 30
Description of the error: 30
Analyzing and solving the error: 31
How to solve: 31
6.2. Error 2: Semantic Error. 31
Finding the error: 31
Description of the error: 32
Analyzing and solving the error: 32
How to solve: 32
6.3. Error 3: Logical Error. 32
Error: 32
The values which were given to the program: 33
Finding the error: 33
Description of the error: 34
Solving the error: 34
Executing the program and now the desired output is shown: 34
How to solve: 34
7. Conclusion. 36
8. Bibliography 37
9. Appendix. 38
Course.java 38
AcademicCourse.java 41
NonAcademicCourse.java 46

List of Figures

Figure 1: Class Diagram. 2
Figure 2: Class Diagram Of Course. 3
Figure 3: Class Diagram Of Academic Course. 3
Figure 4: Class Diagram Of Non-Academic Course. 4
Figure 5: Assigning values to AcademicCourse class. 19
Figure 6: Inspecting the AcademicCourse. 19
Figure 7: Assigning values to register the AcademicCourse class. 20
Figure 8: Inspecting the AcademicCourse after registering. 20
Figure 9: Assigning values to NonAcademicCourse class. 22
Figure 10: Inspecting the NonAcademicCourse. 22
Figure 11: Assigning values to register the NonAcademicCourse class. 23
Figure 12: Inspecting the NonAcademicCourse after registering. 23
Figure 13: Assigning values to NonAcademicCourse class. 25
Figure 14: Inspecting the NonAcademicCourse. 25
Figure 15: Changing isRemoved status to true. 26
Figure 16: Inspecting the NonAcademicCourse after isremoved is changed to true. 26
Figure 17: Display of AcademicCourse. 28
Figure 18: Display of NonAcademicCourse. 29
Figure 19: Syntax error. 30
Figure 20: Solving the error. 31
Figure 21: Semantic Error. 31
Figure 22: Solving the error. 32
Figure 23: Logical Error. 32
Figure 24: Values given to the program. 33
Figure 25: Error found. 33
Figure 26: Error solved. 34
Figure 27: Desired output is shown. 34

List of Tables

Table 1: Course.java 14
Table 2: AcademicCourse.java 16
Table 3: NonAcademicCourse.java 17
Table 4: Test 1. 18
Table 5: Test 2. 21
Table 6: Test 3. 24
Table 7: Test 4. 28

1. Introduction.

One of the modules that Computing students' study in Semester 1 is "Programming". The assignment was given in the 8th week which weights 30% of the overall module. It was given to create a base so that the students can learn proper implementation of object-oriented concept of Java while facing practical problems in the real world.

This concept includes:

a. Creating a class to represent a Course.
b. Along with its two subclasses to represent an Academic course and a Non-academic Course respectively.

A new project is created in BlueJ and has three new classes: Course, Academic Course and Non-Academic Course. The Course class has four attributes and is the superclass, whereas The AcademicCourse has seven attributes and The NonAcademicCourse has eight attributes. Each attribute has a corresponding accessor method.

The report is evidence based with proper description and diagrams. It follows a standard format and has a transparent body of work including evaluation and reflection along with the difficulties encountered. Writing a report would enhance the horizon and help polish the problem-solving skills.

2. Class Diagram.

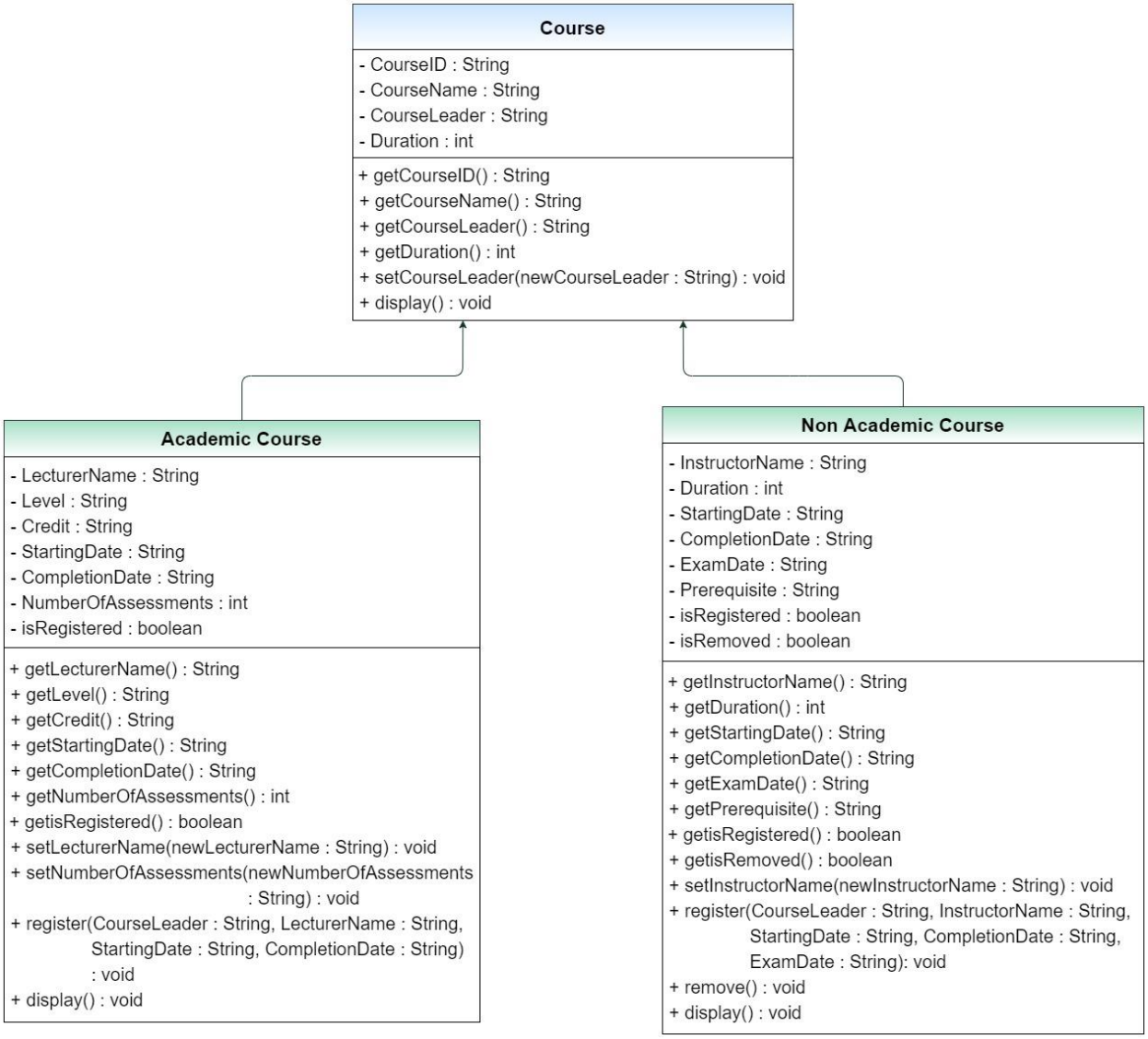

Figure 1: Class Diagram.

2.1 Class Diagram of Course.

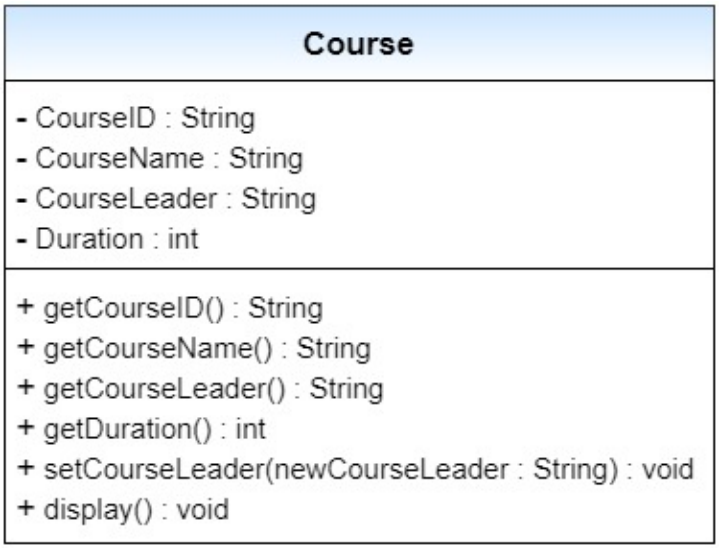

Figure 2: Class Diagram Of Course.

2.2 Class Diagram of Academic Course.

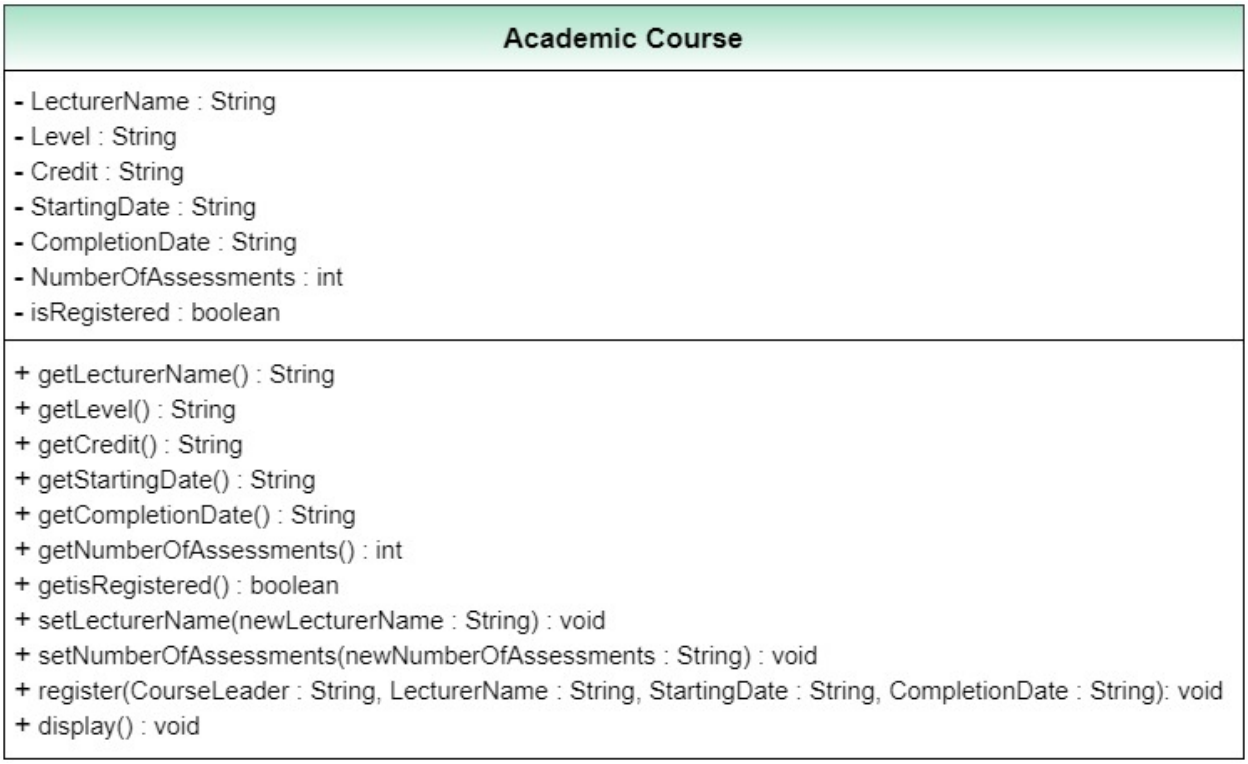

Figure 3: Class Diagram Of Academic Course.

2.3 Class Diagram Of Non-Academic Course.

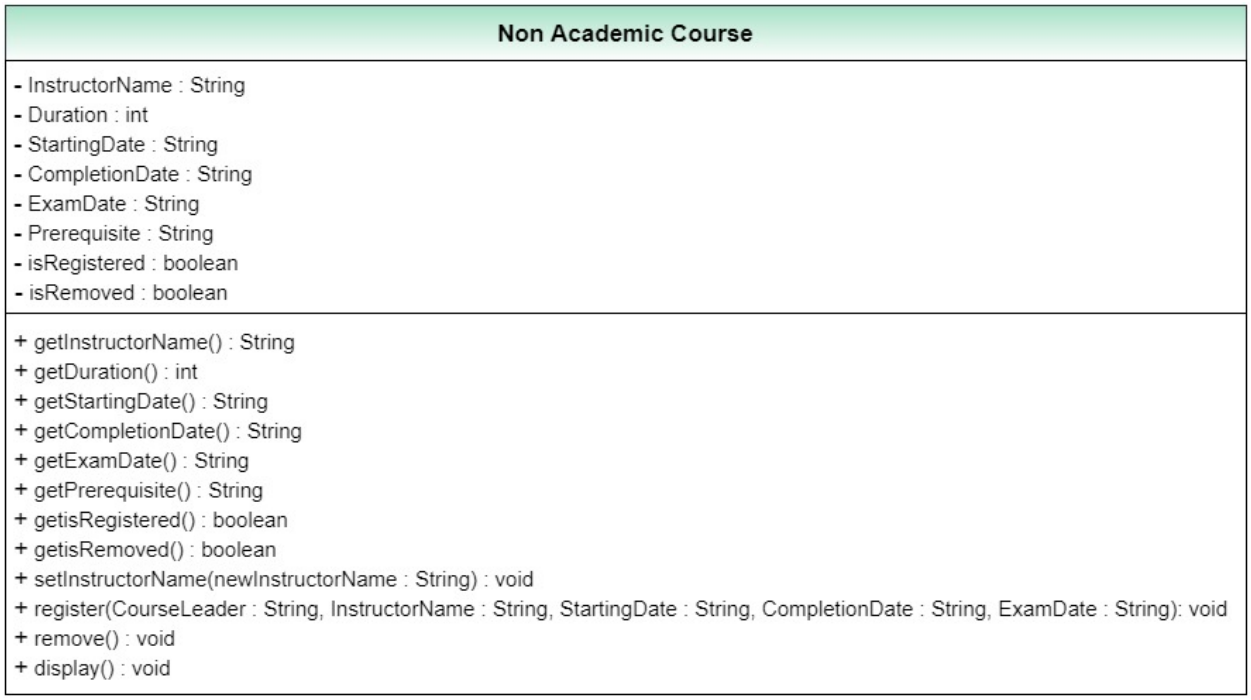

Figure 4: Class Diagram Of Non-Academic Course.

3. Pseudocode.

Pseudocode is a procedure that enables the programmer to represent the execution of an algorithm. Pseudocode consists of short but informative text which is usually written in plain English as it does not contain any programming languages even non-programmers can recognize the working of a program. The main goal of pseudocode is to give programmers a basic sketch which will make the coding step considerably easier. Although programmers cannot compile or execute the pseudocode, it helps them understand the data flow of the program.

3.1 Pseudocode of the class Course:

```
START
    CREATE Course class
        READ four instance variables as CourseID, CourseName,
    CourseLeader, and Duration.

    CREATE getCourseID() as String type
    DO
        GET this.CourseID
    END DO

    CREATE getCourseName() as String type
    DO
        GET this.CourseName
    END DO

    CREATE getCourseLeader() as String type
    DO
        GET this.CourseLeader
    END DO

    CREATE getDuration() as int type
    DO
        GET this.Duration
    END DO

    CREATE setCourseLeader(newCourseLeader) as String type
    DO
        INITIALIZE this.CourseLeader to newCourseLeader
    END DO
```

```
        CREATE display()
        DO
                PRINT “Course ID: “ + getCourseID()
                PRINT “Course Name: “ + getCourseName()
                PRINT “Course Duration: “ + getDuration()
                IF CourseLeader != “”
                        PRINT “Course Leader: “ + getCourseLeader()
                END IF
        END DO
END
```

3.2 Pseudocode of the class AcademicCourse:

```
START
    CREATE AcademicCourse class EXTENDS Course
        READ seven instance variables as LecuturerName, Level, Credit,
    StartingDate, CompletionDate, NumberOfAssessments, and isRegistered.

    CREATE getLecturerName() as String type
    DO
        GET this.LecturerName
    END DO

    CREATE getLevel() as String type
    DO
        GET this.Level
    END DO

    CREATE getCredit() as String type
    DO
        GET this.Credit
    END DO

    CREATE getStartingDate() as String type
    DO
        GET this.StartingDate
    END DO

    CREATE getCompletionDate() as String type
    DO
        GET this.CompletionDate
    END DO
```

```
CREATE getNumberOfAssessments() as int type
DO
        GET this.NumberOfAssessments
END DO

CREATE getisRegistered() as boolean type
DO
        GET this.isRegistered
END DO

CREATE setLecturerName(newLecturerName) as String type
DO
        INITIALIZE this.LecturerName to newLecturerName
END DO

CREATE setNumberOfAssessments(newNumberOfAssessments) as int type
DO
        INITIALIZE this.NumberOfAssessments to newNumberOfAssessments
END DO

CREATE register(READ CourseLeader, LecturerName, StartingDate,
CompletionDate)
DO
        IF this.isRegistered == true
                PRINT "This course is already registered. The details of the
        course: "
                PRINT "Lecturer Name: " + this.LecturerName()
                PRINT "Starting Date: " + this.StartingDate()
                PRINT "Completion Date: " + this.CompletionDate()
        ELSE
```

```
                CALL super class course.setCourseLeader(CourseLeader)
                INITIALIZE this.LecturerName to LecturerName
                INITIALIZE this.StartingDate to StartingDate
                INITIALIZE this.CompletionDate to CompletionDate
                INITIALIZE this.isRegistered to true
            END IF
        END DO

        CREATE display()
        DO
            CALL super class course .display()
            IF this.isRegistered == true
                PRINT "Lecturer Name: " + getLecturerName()
                PRINT "Level: " + getLevel()
                PRINT "Credit: " + getCredit()
                PRINT "Starting Date: " + getStartingDate()
                PRINT "Completion Date: " + getCompletionDate()
                PRINT "Number of Assessments: " +
            getNumberOfAssessments()
            END IF
        END DO
END
```

3.3 Pseudocode of the class NonAcademicCourse:

```
START
        CREATE NonAcademicCourse class EXTENDS Course.
            READ seven instance variables as InstructorName, Duration, StartingDate,
        CompletionDate, ExamDate, Prerequisite, isRegistered and isRemoved.
```

```
CREATE getInstructorName() as String type
DO
        GET this.InstructorName
END DO

CREATE getDuration() as int type
DO
        GET this.Duration
END DO

CREATE getStartingDate() as String type
DO
        GET this.StartingDate
END DO

CREATE getCompletionDate() as String type
DO
        GET this.CompletionDate
END DO

CREATE getExamDate() as String type
DO
        GET this.ExamDate
END DO

CREATE getPrerequisite () as String type
DO
        GET this.Prerequisite
END DO
```

```
CREATE getisRegistered() as boolean type
DO
        GET this.isRegistered
END DO

CREATE getisRemoved() as boolean type
DO
        GET this.isRemoved
END DO

CREATE setInstructorName(newInstructorName) as String type
DO
        IF this.isRegistered == false
                INITIALIZE this.InstructorName to newInstructorName
        ELSE
                PRINT "The Instructor Name is already registered, cannot update
Instructor Name”
        END IF
END DO

CREATE register(READ CourseLeader, InstructorName, StartingDate,
CompletionDate, ExamDate)
DO
        IF this.isRegistered == false
                INITIALIZE setInstructorName(InstructorName)
                INITIALIZE this.isRegistered to true
        ELSE
                PRINT “This course is already registered. “
        END IF
END DO
```

```
        CREATE remove()
        DO
                IF this.isRemoved == true
                        PRINT “The course is already removed. “
                ELSE
                        CALL super class course .setCourseLeader
                        INTIALIZE this.InstructorName to “”
                        INTIALIZE this.StartingDate to ""
                        INTIALIZE this.CompletionDate to ""
                        INTIALIZE this.ExamDate to ""
                        INTIALIZE this.isRegistered to false
                        INTIALIZE this.isRemoved to true
                END IF
        END DO

        CREATE display()
        DO
                CALL super class course .display()
                IF this.isRegistered == true
                        PRINT "Instructor Name: " + getInstructorName()
                        PRINT "Starting Date: " + getStartingDate()
                        PRINT "Completion Date: " + getCompletionDate()
                        PRINT "Exam Date: " + getExamDate()
                END IF
        END DO
END
```

4. Method Description.

4.1 Course.java

Method	Description
String getCourseID()	It is an accessor method which returns the Course ID of the Course.
String getCourseName()	It is an accessor method which returns the Course Name of the Course.
String getCourseLeader()	It is an accessor method which returns the Course Leader of the Course.
int getDuration()	It is an accessor method which returns the Duration of the Course.
void setCourseLeader (String newCourseLeader)	Setter method used to assign a new Course Leader for the Course using the parameter values.
void display()	Method to display the details of the Course. Course ID, Course Name, Duration, and Course Leader (if assigned) will be displayed.

Table 1: Course.java

4.2 AcademicCourse.java

Method	Description
String getLecturerName()	It is an accessor method which returns the Lecturer Name of the Academic Course.
String getLevel()	It is an accessor method which returns the Level of the Academic Course.
String getCredit()	It is an accessor method which returns the Credit of the Academic Course.
String getStartingDate()	It is an accessor method which returns the Starting Date of the Academic Course.
String getCompletionDate()	It is an accessor method which returns the Completion Date of the Academic Course.
String getNumberOfAssessments()	It is an accessor method which returns the Number of Assessments of the Academic Course.
boolean getisRegistered()	It is an accessor method which returns the Registered Status of the Academic Course.
void setLecturerName(String newLecturerName)	Setter method used to assign a new Lecturer Name for the Academic Course using the parameter values.
void setNumberOfAssessments(int newNumberOfAssessments)	Setter method used to assign a new Number of Assessments for the Academic Course using the parameter values.

void register (String CourseLeader, String LecturerName, String StartingDate, String CompletionDate)	Method used to Register required information like the Course Leader, Lecturer Name, Starting Date and the Completion Date of the Academic Course.
void display()	Method to display the details of the Academic Course. It will also call the method in Course class to display the Course ID, Course Name, Duration and Course Leader (if assigned). If it is registered then Lecturer Name, Level, Credit, Starting Date, Completion Date and Number of Assessment will be displayed.

Table 2: AcademicCourse.java

4.3 NonAcademicCourse.java

Method	**Description**
String getInstructorName()	It is an accessor method which returns the Instructor Name of the Non-Academic Course.
int getDuration()	It is an accessor method which returns the Duration of the Non-Academic Course.
String getStartingDate()	It is an accessor method which returns the Starting Date of the Non-Academic Course.
String getCompletionDate()	It is an accessor method which returns the Completion Date of the Non-Academic Course.
String getExamDate()	It is an accessor method which returns the Exam Date of the Non-Academic Course.

String getPrerequisite()	It is an accessor method which returns the Prerequisite of the Non-Academic Course.
boolean getisRegistered()	It is an accessor method which returns the Registered Status of the Non-Academic Course.
boolean getisRemoved()	It is an accessor method which returns the Removed Status of the Non-Academic Course.
void setInstructorName(String newInstructorName)	Setter method used to assign a new Instructor Name for the Non-Academic Course using the parameter values.
void register (String CourseLeader, String InstructorName, String StartingDate, String CompletionDate, String Exam Date)	Method used to Register required information like the Course Leader, Instructor Name, Starting Date, Completion Date, and the Exam Date of the Non-Academic Course.
void remove()	Method used to remove Non-Academic Course.
void display()	Method to display the details of the Non-Academic Course. It will also call the method in Course class to display the Course ID, Course Name, and Duration. If it is registered then Instructor Name, Starting Date, Completion Date and Exam Date will be displayed.

Table 3: NonAcademicCourse.java

5. Testing (Inspection)

5.1 Test 1: Inspect AcademicCourse class, register an academic course, and re-inspect the AcademicCourse Class.

Test No.	**1.**
Objective:	To inspect AcademicCourse class, register an academic course, and re-inspect AcademicCourse class.
Action:	➢ The AcademicCourse is called with the following arguments: CourseID = “CS4001NI” CourseName = “Programming” Duration = 1 Level = “4” Credit = “30” NumberOfAssessments = 3 ➢ Inspection of the AcademicCourse class. ➢ void register is called with the following arguments: CourseLeader = “Dhruba Sen” LecturerName = “Roshan Tandukar” StartingDate = “8 Week” CompletionDate = “12 Week” ➢ Re-Inspection of the AcademicCourse class.
Expected Result:	Academic Course would be registered.
Actual Result:	Academic Course was registered.
Conclusion:	The test was successful.

Table 4: Test 1.

Output:

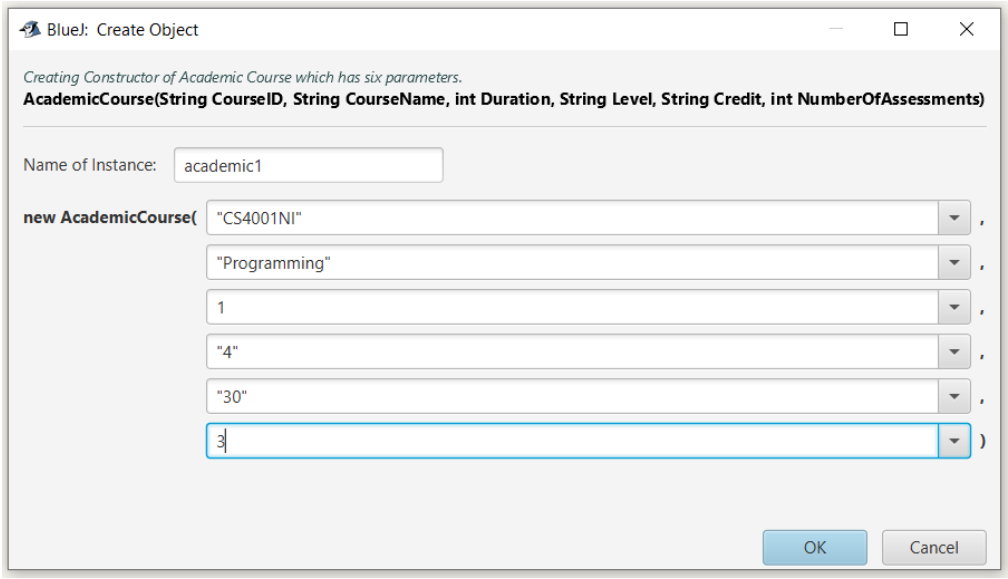

Figure 5: Assigning values to AcademicCourse class.

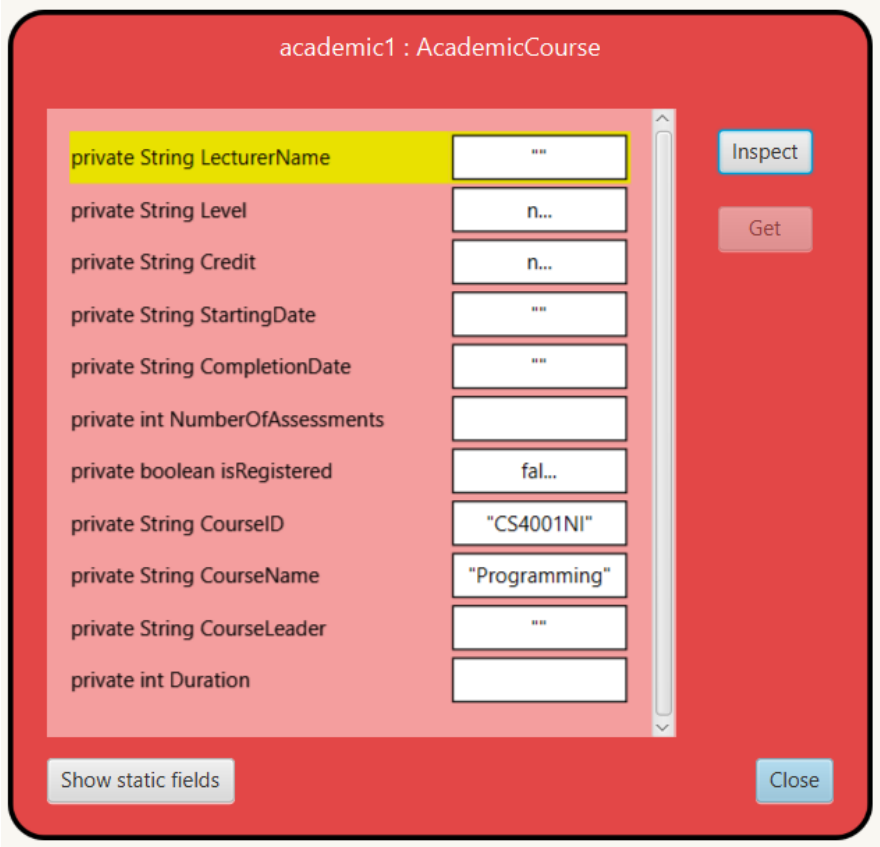

Figure 6: Inspecting the AcademicCourse.

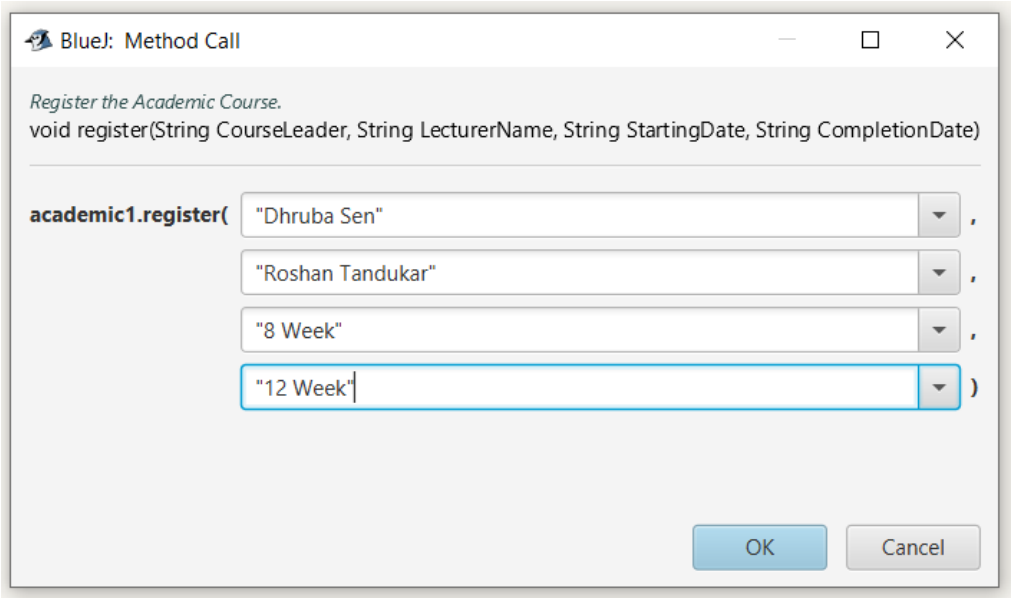

Figure 7: Assigning values to register the AcademicCourse class.

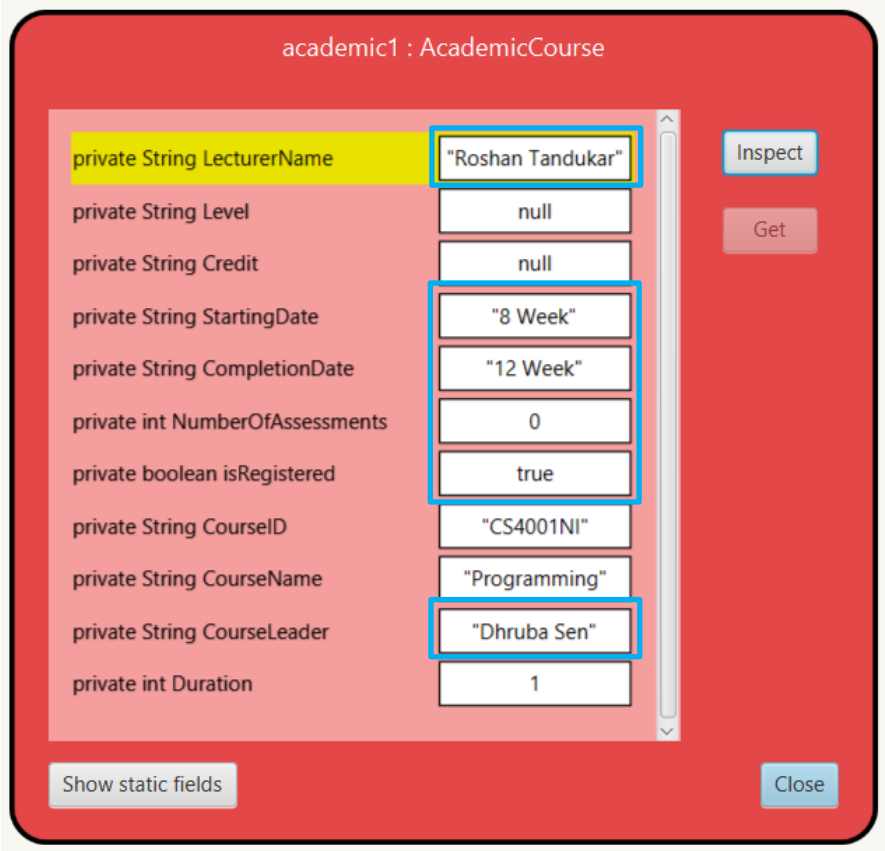

Figure 8: Inspecting the AcademicCourse after registering.

5.2 Test 2: Inspect NonAcademicCourse class, register a non-academic course and re-inspect the NonAcademicCourse Class.

Test No.	**2.**
Objective:	To inspect NonAcademicCourse class, register a Non-Academic Course, and re-inspect NonAcademicCourse class.
Action:	➢ The NonAcademicCourse is called with the following arguments: CourseID = “NC5002EP” CourseName = “Event Planning” Duration = 2 Prerequisite = “Leadership Skills” ➢ Inspection of the NonAcademicCourse class. ➢ void register is called with the following arguments: CourseLeader = “Mira Shrestha” InstructorName = “Serene Gauchan” StartingDate = “12 Week” CompletionDate = “16 Week” ExamDate = “17 Week” ➢ Re-Inspection of the NonAcademicCourse class.
Expected Result:	Non-Academic Course would be registered.
Actual Result:	Non-Academic Course was registered.
Conclusion:	The test was successful.

Table 5: Test 2.

Output:

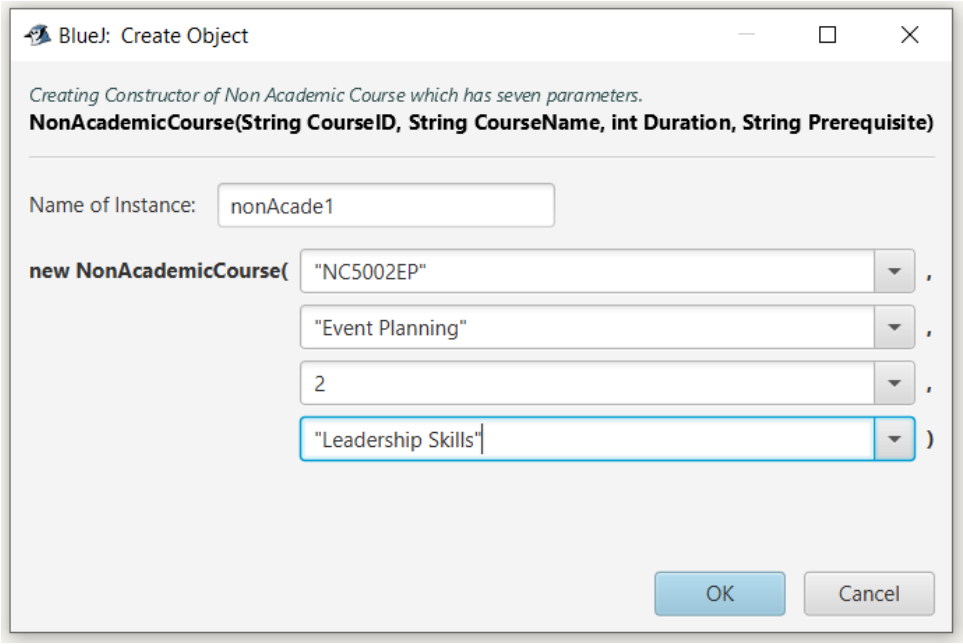

Figure 9: Assigning values to NonAcademicCourse class.

Figure 10: Inspecting the NonAcademicCourse.

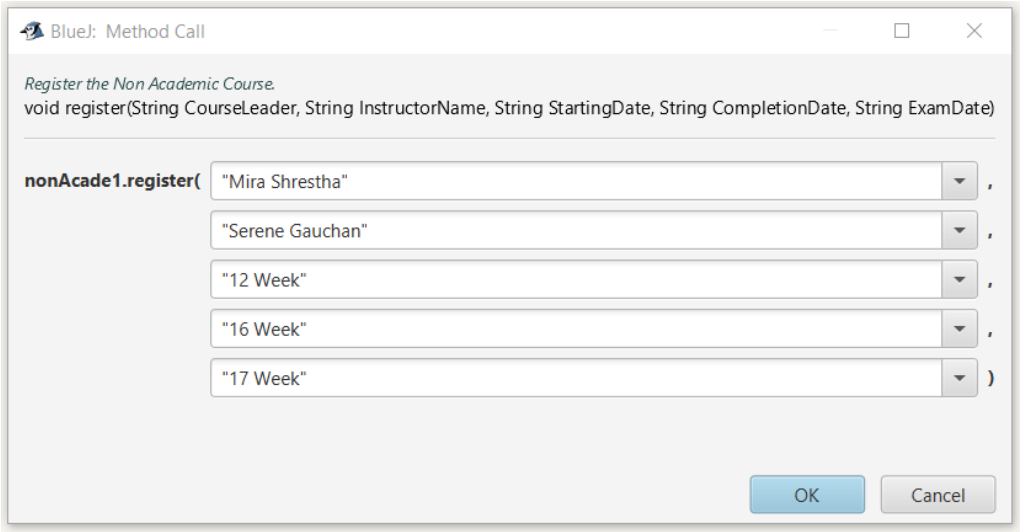

Figure 11: Assigning values to register the NonAcademicCourse class.

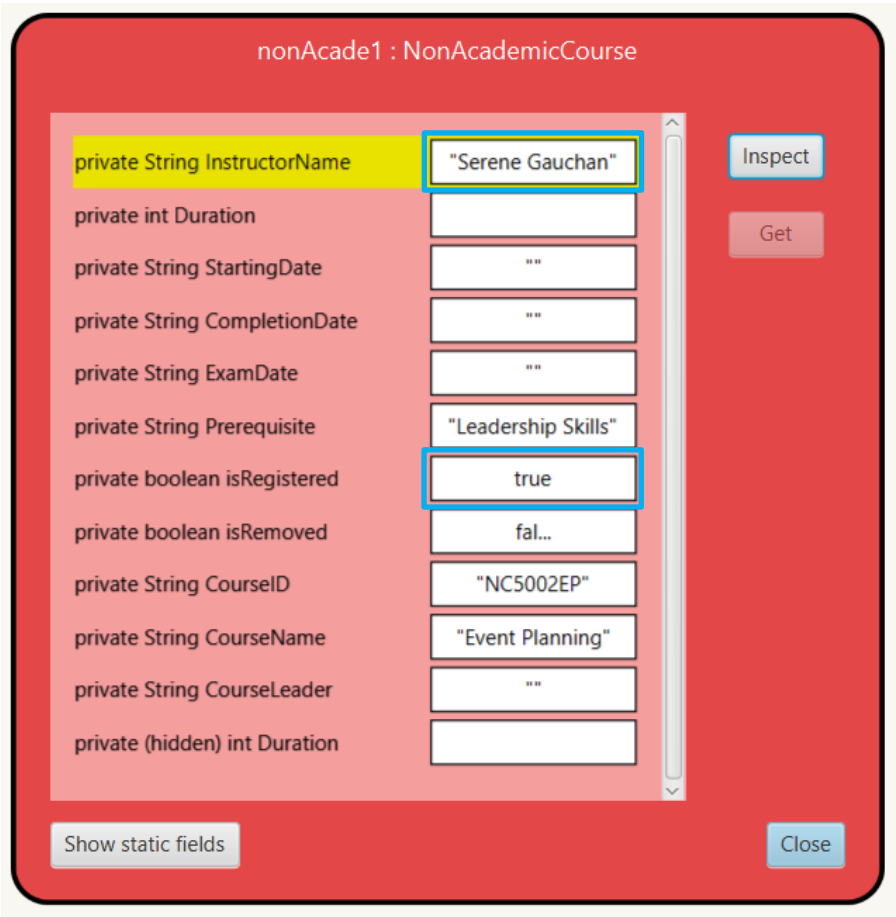

Figure 12: Inspecting the NonAcademicCourse after registering.

5.3 Test 3: Inspect NonAcademicCourse class again, change the status of isRemoved to true and re-inspect the NonAcademicCourse class.

Test No.	**3.**
Objective:	To inspect NonAcademicCourse class, change status of isRemoved to true, and re-inspect NonAcademicCourse class.
Action:	➢ The NonAcademicCourse is called again. With the argument values being the same as above: CourseID = “NC5002EP” CourseName = “Event Planning” Duration = 2 Prerequisite = “Leadership Skills” ➢ Inspection of the Non-AcademicCourse class. ➢ Changing the isRemoved status from false to true. ➢ Re-Inspection of the Non-AcademicCourse class.
Expected Result:	isRemoved status should show true instead of false.
Actual Result:	isRemoved status showed true instead of false.
Conclusion:	The test was successful.

Table 6: Test 3.

Output:

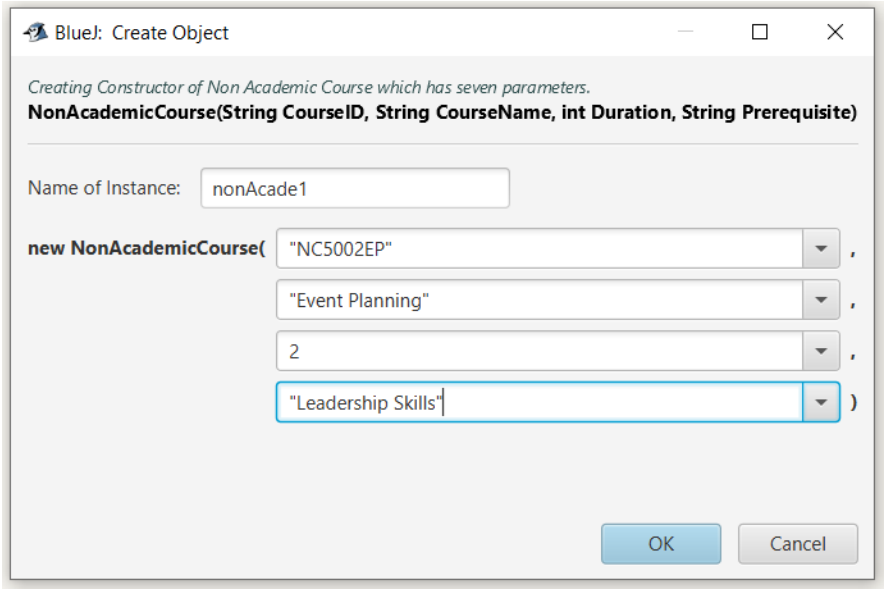

Figure 13: Assigning values to NonAcademicCourse class.

Figure 14: Inspecting the NonAcademicCourse.

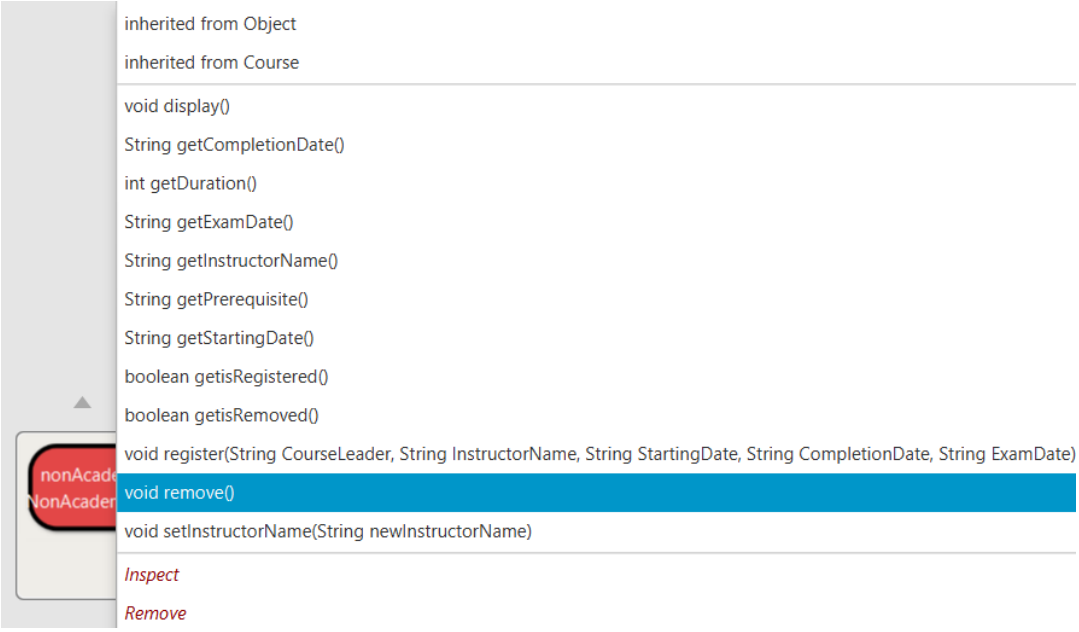

Figure 15: Changing isRemoved status to true.

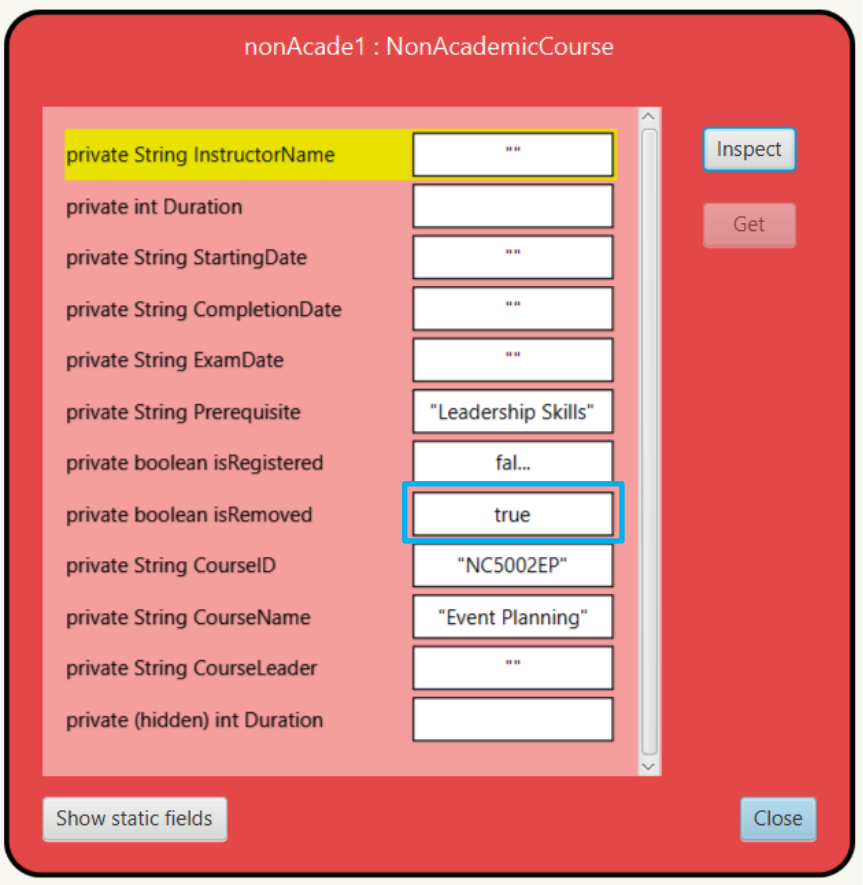

Figure 16: Inspecting the NonAcademicCourse after isremoved is changed to true.

5.4 Test 4: Display the detail of AcademicCourse and NonAcademicCourse class.

Test No.	**4.**
Objective:	To display details of AcademicCourse class and NonAcademicCourse class.
Action:	➢ The AcademicCourse is called again. With argument values being same as test 1: CourseID = “CS4001NI” CourseName = “Programming” Duration = 1 Level = “4” Credit = “30” NumberOfAssessments = 3 ➢ void register was also called with the following arguments: CourseLeader = “Dhruba Sen” LecturerName = “Roshan Tandukar” StartingDate = “8 Week” CompletionDate = “12 Week” ➢ The details of AcademicCourse is now called by using void display(). ➢ The NonAcademicCourse is called again. With argument values being same as test 2,3: CourseID = “NC5002EP” CourseName = “Event Planning” Duration = 2 Prerequisite = “Leadership Skills” CourseLeader = “Mira Shrestha” InstructorName = “Serene Gauchan” StartingDate = “12 Week”

	CompletionDate = “16 Week” ExamDate = “17 Week” ➢ The isRemoved status was also changed to true. ➢ The details of NonAcademicCourse is now called by using void display().
Expected Result:	Details of both Academic Course and Non-Academic Course should be displayed.
Actual Result:	Details of both Academic Course and Non-Academic Course was displayed.
Conclusion:	The test was successful.

Table 7: Test 4.

Output:

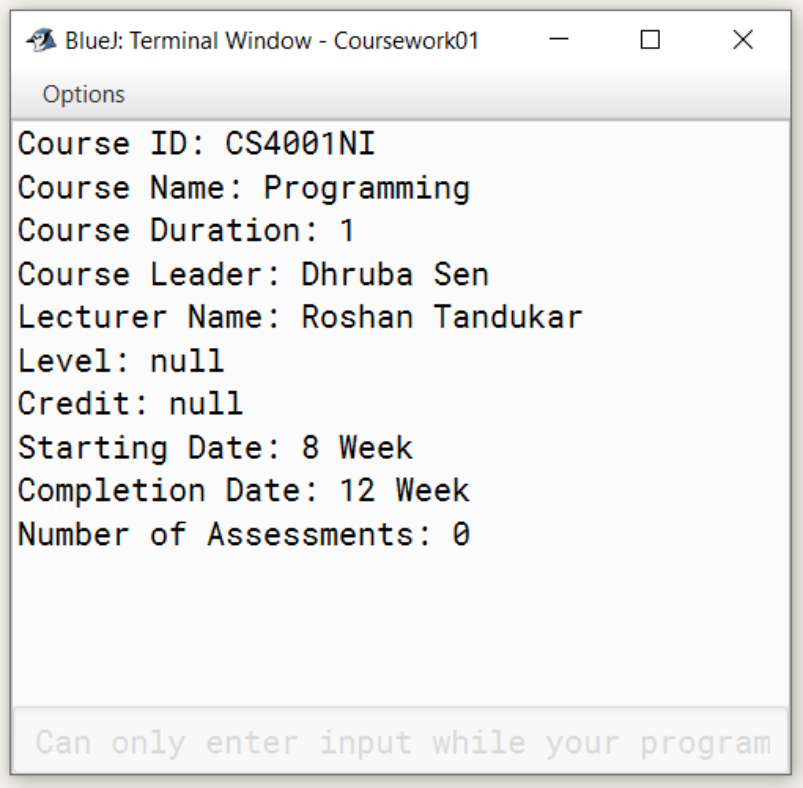

Figure 17: Display of AcademicCourse.

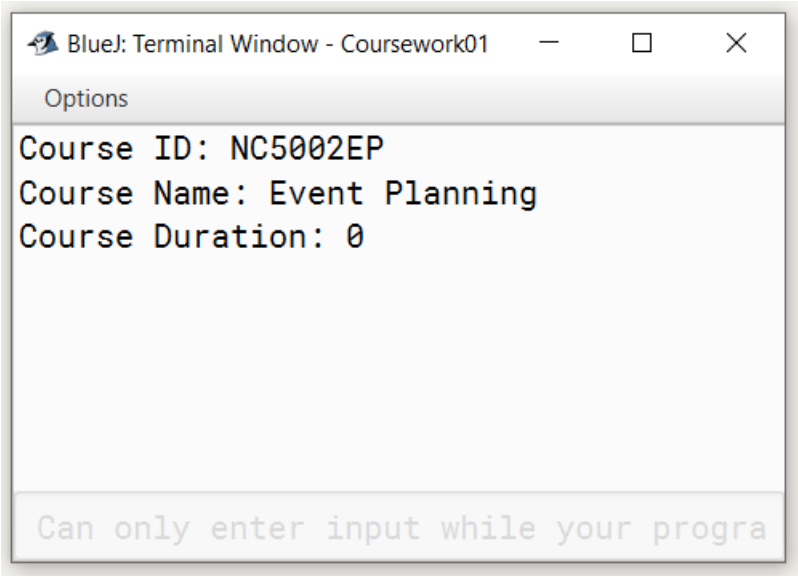

Figure 18: Display of NonAcademicCourse.

6. Errors:

Having some errors in the program could prevent execution of the program or may give an incorrect output. Although errors are bad, experiencing them develops skills in a long run. A programmer starts to get comfortable crossing the bugs they have created and quickly fixes them.

Here are some errors I dealt with, while programming the coursework:

6.1. Error 1: Syntax Error.

Finding the error:

```
/**
 * Register the Academic Course.
 */

public void register(String CourseLeader, String LecturerName , String StartingDate, String CompletionDate);
{
    if (this.isRegistered == true){
        System.out.println("This course is already registered.The details of the course: ");
        System.out.println("Lecturer Name: " + this.LecturerName);
        System.out.println("Starting Date: " + this.StartingDate);
        System.out.println("Completion Date: " + this.CompletionDate);
    }
    else {
        super.setCourseLeader(CourseLeader);
        /* CourseLeader is called from the parent class with course leader name
           as a parameter. */
        this.LecturerName = LecturerName;
        this.StartingDate = StartingDate;
        this.CompletionDate = CompletionDate;
        this.isRegistered = true;
    }
}
Error(s) found in class.
Press Ctrl+K or click link on right to go to next error.
saved
Errors: 2
```

Figure 19: Syntax error.

Description of the error: While writing the program and making sure to add semicolons “;” in order to execute the programs properly, by mistake a semicolon was added where it wasn’t required, which is shown in the above Figure 19.

Analyzing and solving the error:

```java
/**
 * Register the Academic Course.
 */

public void register(String CourseLeader, String LecturerName , String StartingDate, String CompletionDate)
{
    if (this.isRegistered == true){
        System.out.println("This course is already registered.The details of the course: ");
        System.out.println("Lecturer Name: " + this.LecturerName);
        System.out.println("Starting Date: " + this.StartingDate);
        System.out.println("Completion Date: " + this.CompletionDate);
    }
    else {
        super.setCourseLeader(CourseLeader);
        /* CourseLeader is called from the parent class with course leader name
           as a parameter. */
        this.LecturerName = LecturerName;
        this.StartingDate = StartingDate;
        this.CompletionDate = CompletionDate;
        this.isRegistered = true;
    }
}
```

Class compiled - no syntax errors saved

Figure 20: Solving the error.

How to solve: Remove the extra semicolon where it was not required. This type of error is known as "**Syntax Error**" which is caused due to the programmer not following the syntax of the programming language.

6.2. Error 2: Semantic Error.

Finding the error:

```java
/**
 * Return the Registered status of the Non Academic Course.
 */

public String getisRegistered()
{
    return this.isRegistered;
}
```

Figure 21: Semantic Error.

Description of the error: While writing the program, the data type of isRegistered was written as String instead of the data type assigned in the starting which was boolean, the error can be seen in the above Figure 23.

Analyzing and solving the error:

```
/**
 * Return the Registered status of the Non Academic Course.
 */

public boolean getisRegistered()
{
    return this.isRegistered;
}
```

Figure 22: Solving the error.

How to solve: Assign the correct data type to isRegistered, which is boolean. This type of error is known as "**Semantic Error**" which is caused due to the programmer improperly using the Java statements.

6.3. Error 3: Logical Error.

Error:

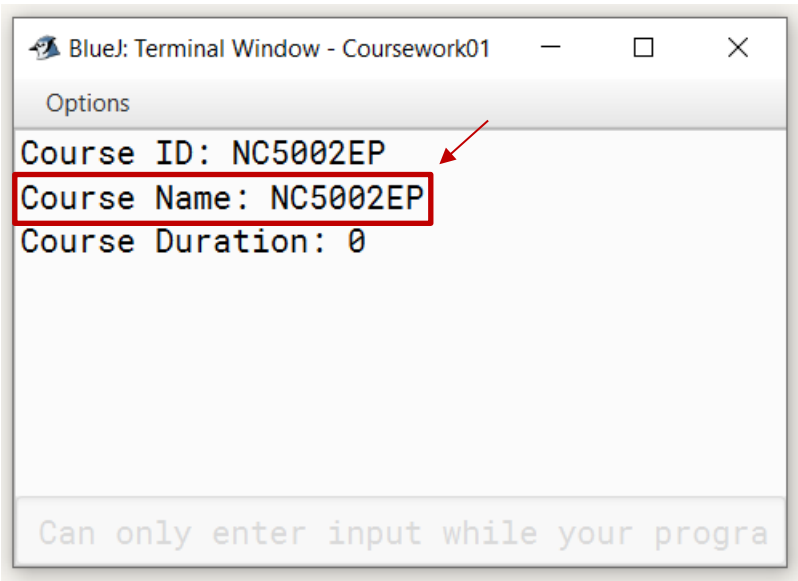

Figure 23: Logical Error.

The values which were given to the program:

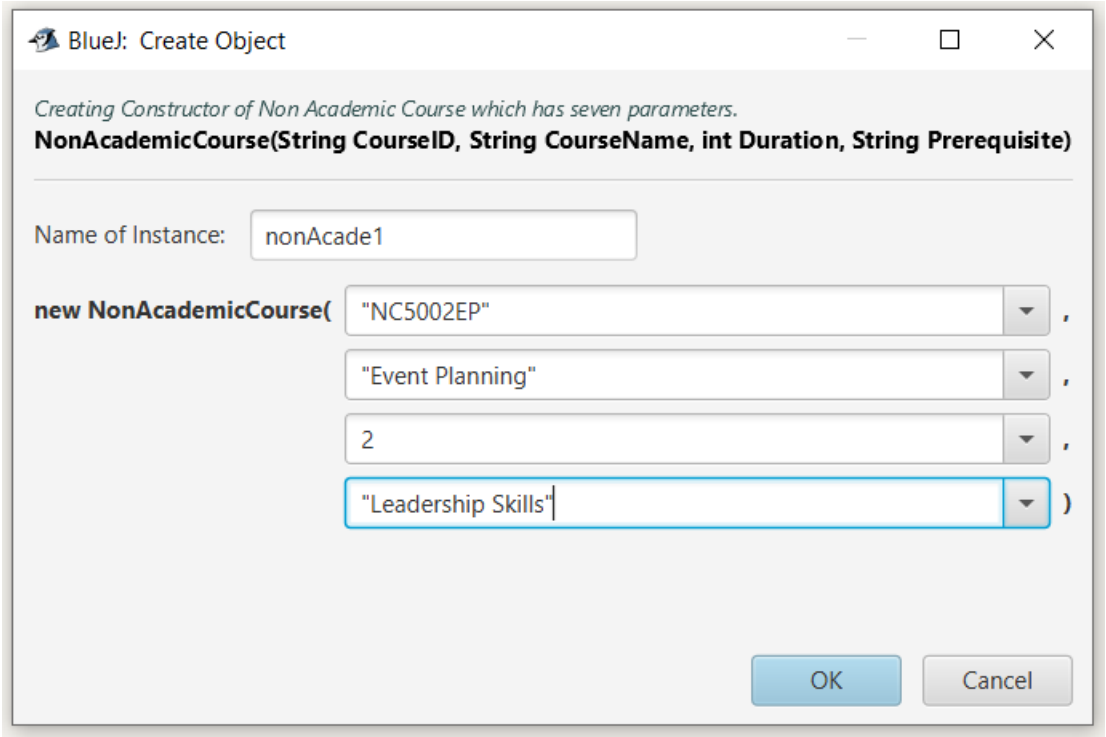

Figure 24: Values given to the program.

Finding the error:

```
/**
 * Display the Course details and Course Leader if assigned.
 */

public void display()
{
    System.out.println("Course ID: " + getCourseID());
    System.out.println("Course Name: " + getCourseID());
    System.out.println("Course Duration: " + getDuration());

    if (CourseLeader != ""){
        System.out.println("Course Leader: " + getCourseLeader());
    }
}
```

Figure 25: Error found.

Description of the error: The output that should have been printed when "CourseName: "was shown the Course Name not the Course ID. The error was due to a mistake which can be seen in the above Figure 23.

Solving the error:

```
/**
 * Display the Course details and Course Leader if assigned.
 */

public void display()
{
    System.out.println("Course ID: " + getCourseID());
    System.out.println("Course Name: " + getCourseName());
    System.out.println("Course Duration: " + getDuration());

    if (CourseLeader != ""){
        System.out.println("Course Leader: " + getCourseLeader());
    }
}
```

Figure 26: Error solved.

Executing the program and now the desired output is shown:

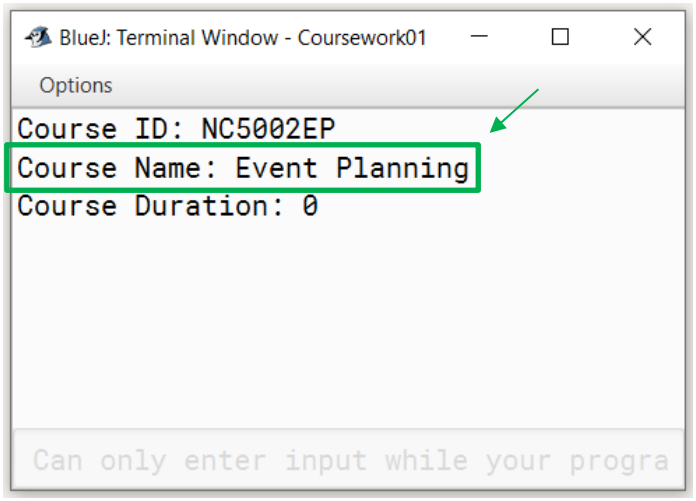

Figure 27: Desired output is shown.

How to solve: The error was found and was corrected immediately, by changing the CourseID to CourseName and the desired output was shown. This type of error is known as "**Logical Error**" which is caused due to mistake in logic. The program was

compiled and executed without showing any errors, but the output did not generate the requested result. Logical errors are hardest to track due to the above reason.

7. Conclusion.

This coursework helped to understand the Object-oriented concepts of Java and its implementation in real-life scenario. It also assisted in creating pseudocodes as it gives programmers a basic sketch which can make the coding step considerably easier by making the data flow of the program clear. It led us to develop our inspection skills while dealing with academiccourse class and nonacademiccourse class.

Programs and errors come hand in hand, but one single error may prevent execution of the program or may give an incorrect output. Thus, a programmer must know how to deal with errors. This coursework guided us to deal with errors and avoid them at the same time.

With the massive experience while completing the assignment, came the difficulties which were rigid but with the present situation it is difficult to get proper guidance from the teachers as we have to rely on electronic means of communication. Despite that, the teachers were very helpful as they guided us as much as possible and I am very thankful for that.

Along with the teachers, the lecture slides and the recordings provided to us was of a big help as it helped us understand the concepts properly. Some sites like geeksforgeeks, textexpander, and w3school also helped to ease the confusions.

Overall, I have tried my way best to complete the coursework successfully and learn the best out of it. I tried to overcome the difficulties and manage accordingly.

8. Bibliography

Osbourn, . T., 2020. *textexpander.* [Online]
Available at: https://textexpander.com/blog/the-7-most-common-types-of-errors-in-programming-and-how-to-avoid-them
[Accessed 14 May 2021].

theprogrammedwords, 2021. *GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/
[Accessed 12 May 2021].

W3Schools, 1999. *W3Schools.* [Online]
Available at: https://www.w3schools.com/java/java_methods_param.asp
[Accessed 11 May 2021].

9. Appendix.

Course.java

```
/**
 * A course class represents a real world course.
 * Course is used as an abstract superclass of:
 * Academic Course and Non Academic Course.
 * @author (Rabina Shrestha)
 * @version (5.0.0)
 */
public class Course
{
    //Four Attributes / Instance Variables of Course Class.

    private String CourseID;
    private String CourseName;
    private String CourseLeader;
    private int Duration;

    /**
     * Creating Constructor of Course with CourseID, Name, and Duration.
     */

    Course(String CourseID, String CourseName, int Duration)
    {
        this.CourseID = CourseID;
        this.CourseName = CourseName;
        this.CourseLeader = "";
        this.Duration = Duration;
    }
```

```
// Using Accessor Method / Getters Method for each attributes.

/**
 * Return the Course ID of the Course.
 */

public String getCourseID()
{
    return this.CourseID;
}

/**
 * Return the Course Name of the Course.
 */

public String getCourseName()
{
    return this.CourseName;
}

/**
 * Return the Course Leader of the Course.
 */

public String getCourseLeader()
{
    return this.CourseLeader;
}

/**
```

```
 * Return the Duration of the Course.
 */

public int getDuration()
{
    return this.Duration;
}

// Parameter passed to method in order to set a new name as the Course Leader.

/**
 * Set a new Course Leader for the Course.
 */

public void setCourseLeader(String newCourseLeader)
{
    this.CourseLeader = newCourseLeader;
}

/**
 * Display the Course details and Course Leader if assigned.
 */

public void display()
{
    System.out.println("Course ID: " + getCourseID());
    System.out.println("Course Name: " + getCourseName());
    System.out.println("Course Duration: " + getDuration());

    if (CourseLeader != ""){
        System.out.println("Course Leader: " + getCourseLeader());
```

```
        }
    }
}
```

AcademicCourse.java

```
/**
 * A class representing Academic Course.
 * Academic Course is used as a subclass of Course.
 * @author (Rabina Shrestha)
 * @version (5.0.0)
 */
public class AcademicCourse extends Course
{
    // Seven Attributes / Instance Variables of Academic Course.

    private String LecturerName;
    private String Level;
    private String Credit;
    private String StartingDate;
    private String CompletionDate;
    private int NumberOfAssessments;
    private boolean isRegistered;

    /**
     * Creating Constructor of Academic Course which has six parameters.
     */

    AcademicCourse(String CourseID,String CourseName, int Duration,String Level,
String Credit, int NumberOfAssessments)
```

```
{
    super(CourseID, CourseName, Duration);
    this.LecturerName = "";
    this.StartingDate = "";
    this.CompletionDate = "";
    this.isRegistered = false;
}

// Using Accessor Method / Getters Method for each attributes.

/**
 * Return the Lecturer Name of the Academic Course.
 */

public String getLecturerName()
{
    return this.LecturerName;
}

/**
 * Return the Level of the Academic Course.
 */

public String getLevel()
{
    return this.Level;
}

/**
 * Return the Credit of the Academic Course.
 */
```

```
public String getCredit()
{
    return this.Credit;
}

/**
 * Return the Starting Date of the Academic Course.
 */

public String getStartingDate()
{
    return this.StartingDate;
}

/**
 * Return the Completion Date of the Academic Course.
 */

public String getCompletionDate()
{
    return CompletionDate;
}

/**
 * Return the Number of Assessments of the Academic Course.
 */

public int getNumberOfAssessments()
{
    return this.NumberOfAssessments;
}
```

```
    /**
     * Return the Registered Status of the Academic Course.
     */

    public boolean getisRegistered()
    {
        return this.isRegistered;
    }

    // Parameter passed to method in order to change and set new Lecturer Name and
new Number of Assessments.

    /**
     * Set a new Lecturer Name for the Academic Course.
     */

    public void setLecturerName(String newLecturerName)
    {
        this.LecturerName = newLecturerName;
    }

    /**
     * Set a new Number of Assessments for the Academic Course.
     */

    public void setNumberOfAssessments(int newNumberOfAssessments)
    {
        this.NumberOfAssessments = newNumberOfAssessments;
    }
```

```
    /* Method used to register any particular academic course.
       Method has four parameters. */

    /**
     * Register the Academic Course.
     */

    public void register(String CourseLeader, String LecturerName , String StartingDate,
String CompletionDate)
    {
        if (this.isRegistered == true){
            System.out.println("This course is already registered.The details of the course:
");
            System.out.println("Lecturer Name: " + this.LecturerName);
            System.out.println("Starting Date: " + this.StartingDate);
            System.out.println("Completion Date: " + this.CompletionDate);
        }
        else {
            super.setCourseLeader(CourseLeader);
            /* CourseLeader is called from the parent class with course leader name
               as a parameter. */
            this.LecturerName = LecturerName;
            this.StartingDate = StartingDate;
            this.CompletionDate = CompletionDate;
            this.isRegistered = true;
        }
    }

    /**
     * Display the Academic Course details.
     */
```

```
    public void display()
    {
        super.display();
        /* Calling method in Course class to display
           CourseID, CourseName, Duration, and CourseLeader. */

        if (this.isRegistered == true)
        {
            System.out.println("Lecturer Name: " + 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: " + getNumberOfAssessments());
        }
    }
}
```

NonAcademicCourse.java

```
/**
 * A class representing Non Academic Course.
 * Non Academic Course is used as a subclass of Course.
 * @author (Rabina Shrestha)
 * @version (5.0.0)
 */
public class NonAcademicCourse extends Course
{
    // Seven Attributes / Instance Variables of Non Academic Course.
```

```
    private String InstructorName;
    private int Duration;
    private String StartingDate;
    private String CompletionDate;
    private String ExamDate;
    private String Prerequisite;
    private boolean isRegistered;
    private boolean isRemoved;

    /**
     * Creating Constructor of Non Academic Course which has seven parameters.
     */

    NonAcademicCourse(String CourseID, String CourseName, int Duration, String
Prerequisite)
    {
        super(CourseID, CourseName, Duration);
        this.InstructorName = "";
        this.Prerequisite = Prerequisite;
        this.StartingDate = "";
        this.CompletionDate = "";
        this.ExamDate = "";
        this.isRegistered = false;
        this.isRemoved = false;
    }

    // Using Accessor Method / Getters Method for each attributes.

    /**
     * Return the Instructor Name of the Non Academic Course.
     */
```

```
public String getInstructorName()
{
    return this.InstructorName;
}

/**
 * Return the Duration of the Non Academic Course.
 */

public int getDuration()
{
    return this.Duration;
}

/**
 * Return the Starting Date of the Non Academic Course.
 */

public String getStartingDate()
{
    return this.StartingDate;
}

/**
 * Return the Completion Date of the Non Academic Course.
 */

public String getCompletionDate()
{
    return this.CompletionDate;
}
```

```
/**
 * Return the Exam Date of the Non Academic Course.
 */

public String getExamDate()
{
    return this.ExamDate;
}

/**
 * Return the Prerequisite of the Non Academic Course.
 */

public String getPrerequisite()
{
    return this.Prerequisite;
}

/**
 * Return the Registered status of the Non Academic Course.
 */

public boolean getisRegistered()
{
    return this.isRegistered;
}

/**
 * Return the Removed status of the Non Academic Course.
 */
```

```
    public boolean getisRemoved()
    {
        return this.isRemoved;
    }

    // Parameter passed to method in order to change and set new Instructor Name.

    /**
     * Set a new Instructor Name for the Non Academic Course.
     */

    public void setInstructorName(String newInstructorName)
    {
        if (this.isRegistered == false){
            this.InstructorName = newInstructorName;
        }
        else {
            System.out.println("The Instructor Name is already registered, cannot update
instructor name");
        }
    }

    /* Method used to register the non academic course.
       Method has five parameters.*/

    /**
     * Register the Non Academic Course.
     */

    public void register(String CourseLeader, String InstructorName, String StartingDate,
String CompletionDate, String ExamDate)
```

```
{
    if (this.isRegistered == false){
        setInstructorName(InstructorName);
        this.isRegistered = true;
    }
    else {
        System.out.println("The course is already registered.");
    }
}

// Method used to remove the non academic course.

/**
 * Remove the Non Academic Course.
 */

public void remove()
{
    if (this.isRemoved == true){
        System.out.println("The course is already removed.");
    }
    else {
        super.setCourseLeader("");
        this.InstructorName = "";
        this.StartingDate = "";
        this.CompletionDate = "";
        this.ExamDate = "";
        this.isRegistered = false;
        this.isRemoved = true;
    }
}
```

```
    /**
     * Display the Non Academic Course details.
     */

    public void display()
    {
        super.display();
        /* Calling method in Course class to display
           CourseID, CourseName, and Duration. */

        if (this.isRegistered == true) {
            System.out.println("Instructor Name: " + getInstructorName());
            System.out.println("Starting Date: " + getStartingDate());
            System.out.println("Completion Date: " + getCompletionDate());
            System.out.println("Exam Date: " + getExamDate());
        }
    }
}
```