



slington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC4001NI Programming

COURSEWORK-1

Assessment Weightage & Type

30% Individual Coursework

Year and Semester

Spring 2021

Student Name: Niwahang Angbuhang

Group: C9

London Met ID: 20048942

College ID: NP01CP4S210237

Assignment Due Date: 23rd May 2021

Assignment Submission Date: 22nd 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.

Table of Contents

1. Introduction	1
2. Class Diagram	2
2.1 Class Diagram of Course Class	2
2.2 Class Diagram of Academic Course	2
2.3 Class Diagram of Non-Academic Course	3
2.4 Final Class Diagram	4
3. Pseudocode	5
3.1 Pseudocode for Course class	5
3.2 Pseudocode for Academic Course class	6
3.3 Pseudocode for Non-Academic Course class	8
4. Method Description	10
5. Testing	15
5.1 Test 1 – To inspect AcademicCourse class, register an academic course and re-inspect the AcademicCourse again	15
5.2 Test 2 – To inspect NonAcademicCourse class, register a non-academic course and re-inspect the NonAcademicCourse again	18
5.3 Test 3 – To inspect NonAcademicCourse class again, change the status of isRemoved to true and re-inspect the NonAcademicCourse	20
5.4 Test 4 – To display the detail of AcademicCourse and NonAcademicCourse classes	22
6. Different Error Detection and Correction	24
6.1 Syntax Error	24
6.2 Semantic Error	25
6.3 Logical Error	26
7. Conclusion	28
7.1 Evaluation of the work	28
7.2 What we have learned	28
7.3 Difficulties encountered	28
7.4 How we overcame those difficulties	28
Appendix	29
Course Class	29
AcademicCourse Class	30
NonAcademicCourse Class	32

List of Figures

Figure 1 Class Diagram of Course Class	2
Figure 2 Class Diagram of Academic Course.....	2
Figure 3 Class Diagram of Non-Academic Course	3
Figure 4 Final Class Diagram	4
Figure 5 Assigning data in AcademicCourse Class.....	15
Figure 6 Inspecting AcademicCourse Class.....	16
Figure 7 Registering AcademicCourse Class	16
Figure 8 Inspecting AcademicCourse Class again.....	17
Figure 9 Assigning Data in NonAcademicCourse Class	18
Figure 10 Inspecting NonAcademicCourse Class	19
Figure 11 Registering NonAcademicCourse Class	19
Figure 12 Inspecting NonAcademicCourse Class again.....	20
Figure 13 Inspecting NonAcademicCourse Class again.....	21
Figure 14 Using remove method	21
Figure 15 Inspecting NonAcademicCourse Class again.....	22
Figure 16 Displaying details of AcademicCourse Class	23
Figure 17 Displaying Details of NonAcademicCourse Class	23
Figure 18 Syntax Error Detection	24
Figure 19 Syntax Error Correction.....	24
Figure 20 Semantic Error Detection	25
Figure 21 Semantic Error Correction	25
Figure 22 Logical Error Detection.....	26
Figure 23 Logical Error Display (1)	26
Figure 24 Logical Error Display (2).....	27
Figure 25 Logical Error Correction	27

List of Tables

Table 1 Test Table 1	15
Table 2 Test Table 2	18
Table 3 Test Table 3	20
Table 4 Test Table 4	22

1. Introduction

This is the first coursework of programming module. We are required to make a program for an educational institute. The program is created using Java and BlueJ compiler. The program helps to keep records of the courses and the lecturers of the courses.

In this project, three different classes are created: Course, AcademicCourse and NonAcademicCourse. We have created a method that registers an academic course and non-academic course in the institute. The user must input the respective values for the CourseName, CourseLeader, LecturerName and others for it to be registered. Similarly, a remove method and display method are also created.

We also have to make a class diagram and pseudocode in the report section of the coursework. The description of the methods used should also be done. We also have to test our program and show the resulted output in this report. The error detection and correction must be shown. Finally, the conclusion is written along with the appendix of the coding part.

2. Class Diagram

2.1 Class Diagram of Course Class

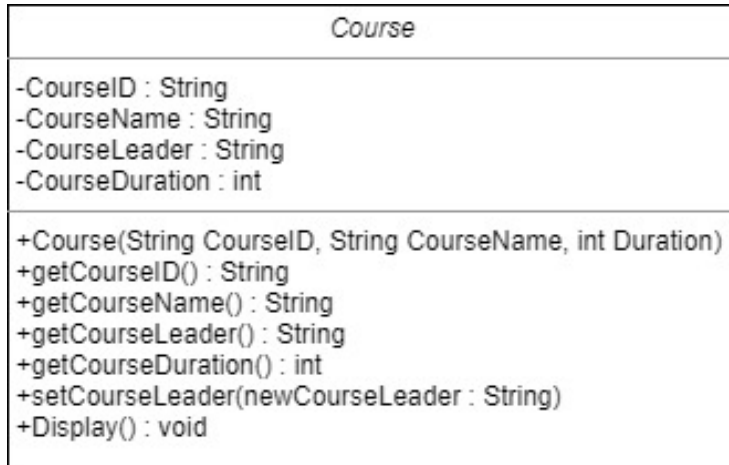


Figure 1 Class Diagram of Course Class

2.2 Class Diagram of Academic Course

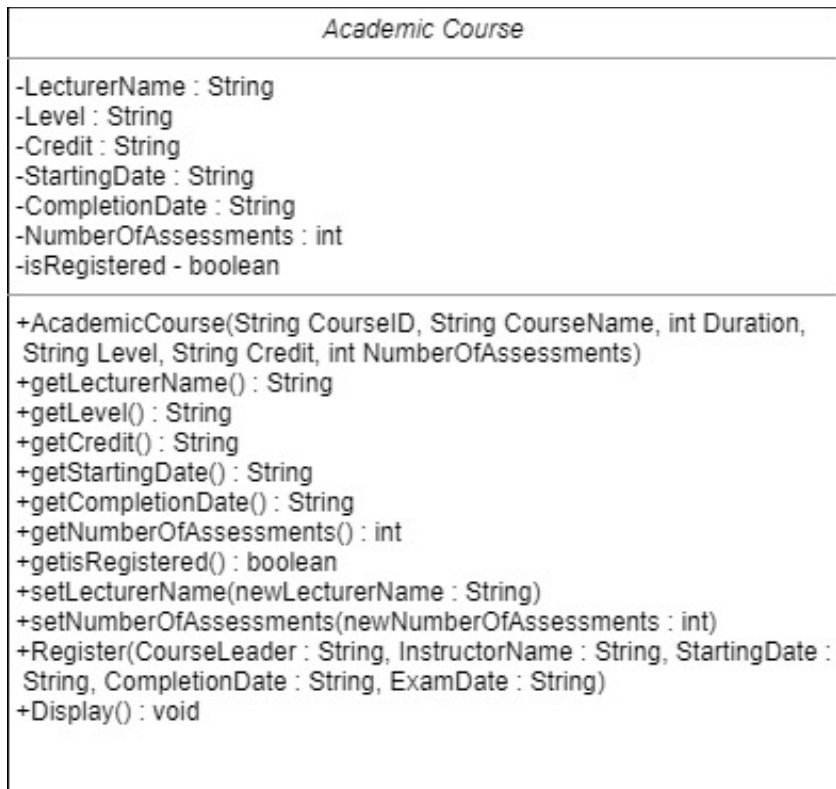


Figure 2 Class Diagram of Academic Course

2.3 Class Diagram of Non-Academic Course

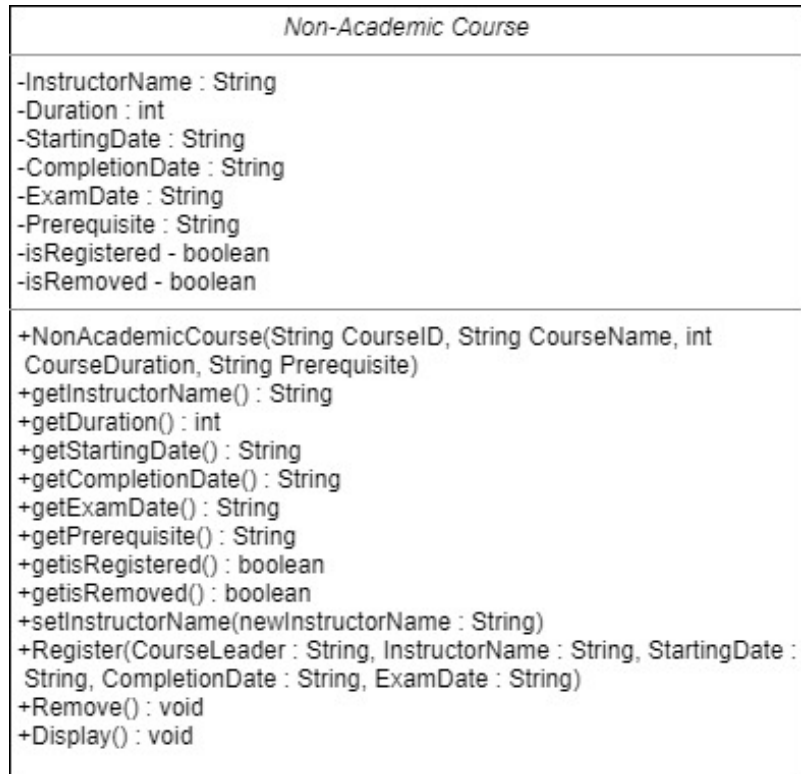


Figure 3 Class Diagram of Non-Academic Course

2.4 Final Class Diagram

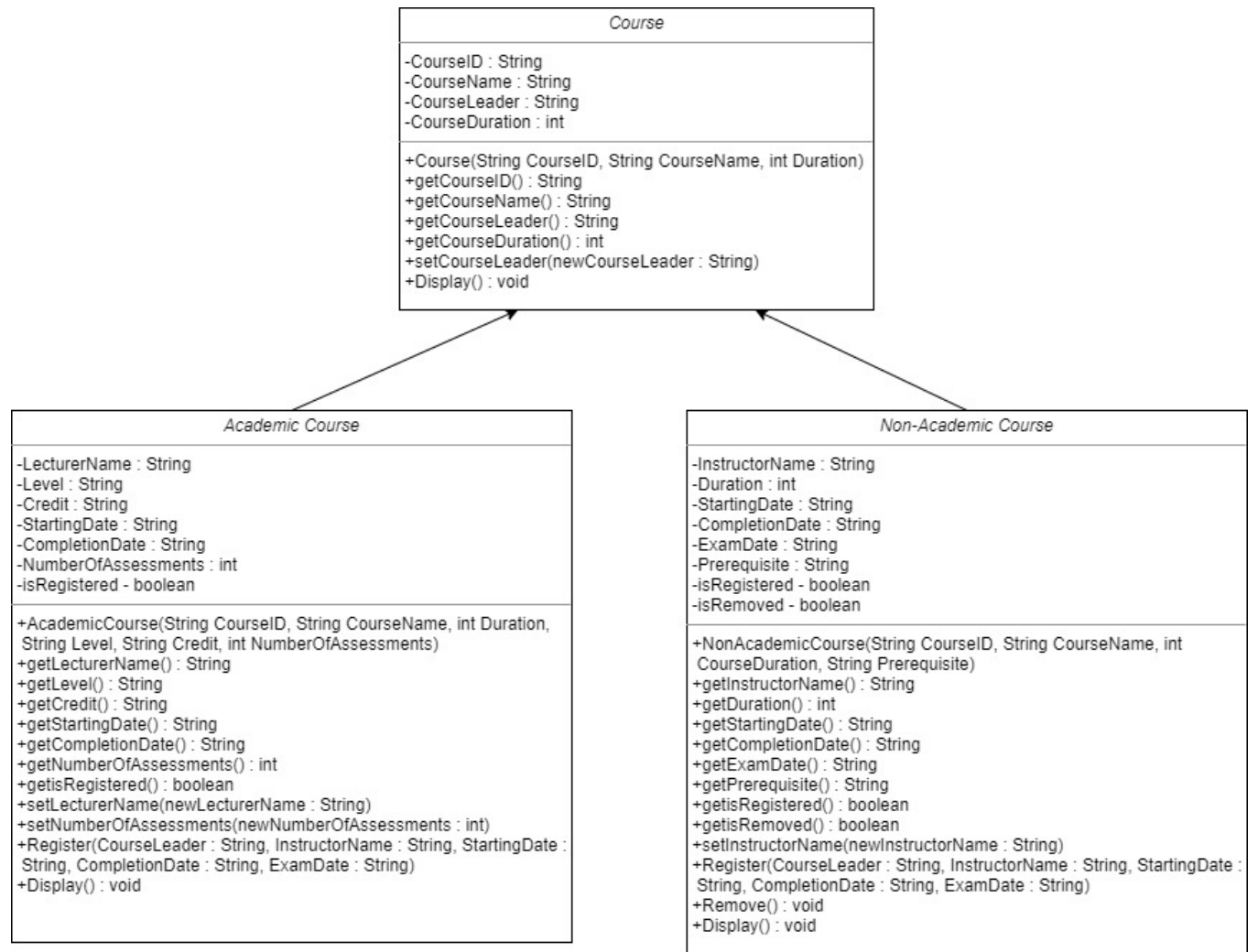


Figure 4 Final Class Diagram

3. Pseudocode

Pseudocode can be said as the readable version of the code. It is written in a way so that anyone can understand the context of the code. It allows the programmer to represent the implementation of an algorithm. Pseudocode does not have any syntax like the programming language and therefore it can't be compiled or interpreted by the computer.

3.1 Pseudocode for Course class

CREATE class Course

DECLARE instance variable CourseID of String type

DECLARE instance variable CourseName of String type

DECLARE instance variable CourseLeader of String type

DECLARE instance variable CourseDuration of int type

CREATE Constructor Course(**PASS** parameter CourseID of String type, CourseName of String type, CourseDuration of int type)

ASSIGN parameter CourseID to instance variable CourseID

ASSIGN parameter CourseName to instance variable CourseName

ASSIGN parameter CourseDuration to instance variable CourseDuration

ASSIGN CourseLeader to null

CREATE method getCourseID with return type String

RETURN CourseID

CREATE method getCourseName with return type String

RETURN CourseName

CREATE method getCourseLeader with return type String

RETURN CourseLeader

CREATE method getCourseDuration with return type int

RETURN CourseDuration

CREATE method setCourseLeader (**PASS** parameter newCourseLeader of String Type)

ASSIGN parameter CourseLeader to instance variable newCourseLeader

CREATE method Display

PRINT CourseID


```
PRINT CourseName
PRINT CourseDuration
IF (CourseLeader is null)
    PRINT CourseLeader
ENDIF
```

3.2 Pseudocode for Academic Course class

CREATE class AcademicCourse

```
DECLARE instance variable LecturerName of String type
DECLARE instance variable Level of String type
DECLARE instance variable Credit of String type
DECLARE instance variable StartingDate of String type
DECLARE instance variable CompletionDate of String type
DECLARE instance variable NumberOfAssessments of int type
DECLARE instance variable isRegistered of boolean type
```

CREATE Constructor AcademicCourse (**PASS** parameter CourseID of String type, CourseName of String type, CourseDuration of int type, Level of String type, Credit of String type, NumberOfAssessments of int type)

```
    CALL constructor from parent class
    ASSIGN parameter CourseID to instance variable CourseID
    ASSIGN parameter CourseName to instance variable CourseName
    ASSIGN parameter CourseDuration to instance variable CourseDuration
    ASSIGN parameter Level to instance variable Level
    ASSIGN parameter NumberOfAssessments to instance variable
    NumberOfAssessments
    ASSIGN parameter Credit to instance variable Credit
    ASSIGN LecturerName to null
    ASSIGN StartingDate to null
    ASSIGN CompletionDate to null
    ASSIGN isRegistered to false
```

CREATE method getLecturerName with return type String
 RETURN LecturerName

CREATE method getLevel with return type String
 RETURN Level

CREATE method getCredit with return type String
 RETURN Credit

CREATE method getStartingDate with return type String
 RETURN StartingDate

CREATE method getCompletionDate with return type String
RETURN CompletionDate

CREATE method getNumberOfAssessments with return type int
RETURN NumberOfAssessments

CREATE method getisRegistered with return type boolean
RETURN isRegistered

CREATE method setLecturerName (**PASS** parameter newLecturerName of String Type)
ASSIGN parameter newLecturerName to instance variable LecturerName

CREATE method setNumberOfAssessments (**PASS** parameter newNumberOfAssessments of String Type)
ASSIGN parameter newNumberOfAssessments to instance variable NumberOfAssessments

CREATE method Register (**PASS** parameter CourseLeader of String type, LecturerName of String type, StartingDate of String type, CompletionDate of String type)

IF (isRegistered is true)

PRINT "Academic course is already registered"
PRINT LecturerName
PRINT StartingDate
PRINT CompletionDate

ELSE

CALL setCourseLeader method from parent class
ASSIGN parameter LecturerName to instance variable LecturerName
ASSIGN parameter StartingDate to instance variable StartingDate
ASSIGN parameter CompletionDate to instance variable CompletionDate
ASSIGN isRegistered to true

CREATE method Display
CALL Display method from parent class
IF (isRegistered is true)
PRINT LecturerName
PRINT Level
PRINT Credit
PRINT StartingDate
PRINT CompletionDate
PRINT NumberOfAssessments
ENDIF

3.3 Pseudocode for Non-Academic Course class

CREATE NonAcademicCourse

DECLARE instance variable InstructorName of String type
DECLARE instance variable CourseDuration of int type
DECLARE instance variable StartingDate of String type
DECLARE instance variable CompletionDate of String type
DECLARE instance variable ExamDate of String type
DECLARE instance variable Prerequisite of String type
DECLARE instance variable isRegistered of boolean type
DECLARE instance variable isRemoved of boolean type

CREATE Constructor NonAcademicCourse (**PASS** parameter CourseID of String type, CourseName of String type, CourseDuration of int type, Prerequisite of String type)

CALL constructor from parent class
ASSIGN parameter CourseID to instance variable CourseID
ASSIGN parameter CourseName to instance variable CourseName
ASSIGN parameter CourseDuration to instance variable CourseDuration
ASSIGN parameter Prerequisite to instance variable Prerequisite
ASSIGN StartingDate to null
ASSIGN CompletionDate to null
ASSIGN ExamDate to null
ASSIGN isRegistered to false
ASSIGN isRemoved to false

CREATE method getInstructorName with return type String
RETURN InstructorName

CREATE method getCourseDuration with int type String
RETURN CourseDuration

CREATE method getStartingDate with return type String
RETURN StartingDate

CREATE method getCompletionDate with return type String
RETURN CompletionDate

CREATE method getExamDate with return type String
RETURN ExamDate

CREATE method getPrerequisite with return type String

RETURN Prerequisite

CREATE method getisRegistered with return type boolean

RETURN isRegistered

CREATE method getisRemoved with return type boolean

RETURN isRemoved

CREATE method setInstructorName (**PASS** parameter newInstructorName of String Type)

IF (isRegistered is true)

PRINT "Since instructor is already registered, it is not possible"

ELSE

ASSIGN parameter newInstructorName to instance variable InstructorName

CREATE method Register (**PASS** parameter CourseLeader of String type, InstructorName of String type, StartingDate of String type, CompletionDate of String type, ExamDate of String type)

IF (isRegistered is true)

PRINT "Non-academic course is already registered"

ELSE

CALL method setCourseLeader from parent class

ASSIGN parameter InstructorName to instance variable LecturerName

ASSIGN parameter StartingDate to instance variable StartingDate

ASSIGN parameter CompletionDate to instance variable CompletionDate

ASSIGN parameter ExamDate to instance variable ExamDate

ASSIGN isRegistered to true

ASSIGN isRemoved to false

CREATE method Remove

IF (isRemoved is true)

PRINT "Non-Academic course is already removed."

ELSE

CALL setCourseLeader from parent class(**PASS** null in instance variable)

ASSIGN instance variable InstructorName to null

ASSIGN instance variable StartingDate to null

ASSIGN instance variable CompletionDate to null

ASSIGN instance variable ExamDate to null

ASSIGN instance variable isRegistered to false

ASSIGN instance variable isRemoved to true

CREATE method Display

CALL Display method from parent class

```
IF (isRegistered is true)
    PRINT InstructorName
    PRINT StartingDate
    PRINT CompletionDate
    PRINT ExamDate
    PRINT Prerequisite
ENDIF
```

4. Method Description

There are some methods used in the codes.

- **Course Class**

- **getCourseID():**

This method is used to access the private attributes and return the values of CourseID. It returns the values in String datatype.

- **getCourseName():**

This method is used to access the private attributes and return the values of CourseName. It returns the values in String datatype.

- **getCourseDuration():**

This method is used to access the private attributes and return the values of CourseDuration. It returns the values in int datatype.

- **getCourseLeader():**

This method is used to access the private attributes and return the values of CourseLeader. It returns the values in String datatype.

- **setCourseLeader(newCourseLeader):**

This method is used to set the value of CourseLeader. It takes a parameter and assigns it as String datatype.

- **Display():**

This method is used to print out the values of the course class. It displays the details of the course class

- **AcademicCourse Class**

- **getLecturerName():**

This method is used to access the private attributes and return the values of LecturerName. It returns the values in String datatype.

- **getLevel():**

This method is used to access the private attributes and return the values of Level. It returns the values in String datatype.

- **getCredit()**

This method is used to access the private attributes and return the values of Credit. It returns the values in String datatype.

- **getStartingDate():**

This method is used to access the private attributes and return the values of StartingDate. It returns the values in String datatype.

- **getCompletionDate():**

This method is used to access the private attributes and return the values of CompletionDate. It returns the values in String datatype.

- **getNumberOfAssessments():**

This method is used to access the private attributes and return the values of NumberOfAssessments. It returns the values in int datatype.

- **getisRegistered():**

This method is used to access the private attributes and return the values of isRegistered. It returns the values in boolean datatype.

- **setLecturerName(newLecturerName):**

This method is used to set the value of LecturerName. It takes a parameter and assigns it as String datatype.

- **setNumberOfAssessments(newNumberOfAssessments):**

This method is used to set the value of NumberOfAssessments. It takes a parameter and assigns it as int datatype.

- **Register(String CourseLeader, String LecturerName, String StartingDate, String Completion Date)**

This method is used to register a course. It takes parameter and checks if it is registered then sets the value from the parameters if it is not registered.

- **Display():**

This method is used to print out the values of the AcademicCourse class. This method also uses the display method from its parent class. It displays the details of the AcademicCourse class.

- **NonAcademicCourse Class**

- **getInstructorName():**

This method is used to access the private attributes and return the values of InstructorName. It returns the values in String datatype.

- **getStartingDate():**

This method is used to access the private attributes and return the values of StartingDate. It returns the values in String datatype.

- **getCompletionDate():**

This method is used to access the private attributes and return the values of CompletionDate. It returns the values in String datatype.

- **getExamDate():**

This method is used to access the private attributes and return the values of ExamDate. It returns the values in String datatype.

- **getPrerequisite():**

This method is used to access the private attributes and return the values of Prerequisite. It returns the values in String datatype.

- **getisRegistered():**

This method is used to access the private attributes and return the values of isRegistered. It returns the values in boolean datatype.

- **getisRemoved():**

This method is used to access the private attributes and return the values of isRemoved. It returns the values in boolean datatype.

- **setInstructorName(newInstructorName):**

This method is used to set the value of InstructorName. It takes a parameter and assigns it as String datatype.

- **Register (String CourseLeader, String InstructorName, String StartingDate, String CompletionDate, String ExamDate)**

This method is used to register a Non-Academic course. It takes parameter and checks if it is registered then sets the value from the parameters if it is not registered.

- **Remove():**

This method is used to remove a course. If the course has already been removed it will display a certain message if not then it will remove the course from the records.

- **Display():**

This method is used to print out the values of the NonAcademicCourse class. This method also uses the display method from its parent class. It displays the details of the NonAcademicCourse class.

5. Testing

5.1 Test 1 – To inspect AcademicCourse class, register an academic course and re-inspect the AcademicCourse again

Test Number	1
Objective:	To inspect AcademicCourse class, register an academic course and re-inspect the AcademicCourse again
Action:	<p>→ The AcademicCourse is called with the following arguments:</p> <p>CourseID = "103"</p> <p>CourseName = "Psychology"</p> <p>CourseDuration = 3</p> <p>Level = "4"</p> <p>Credit = "120"</p> <p>NumberOfAssessments = 6</p> <p>→ Inspection of AcademicCourse Class is done.</p> <p>→ The register method is called with the following arguments:</p> <p>LecturerName = "Jose"</p> <p>CourseLeader = "Savic"</p> <p>StartingDate = "12th May 2021"</p> <p>CompletionDate = "28th April 2024"</p> <p>→ Re-inspection of AcademicCourse class is done.</p>
Expected Result:	The academic course should be registered.
Actual Result:	The academic course was registered.
Conclusion:	The test is successful.

Table 1 Test Table 1

Output Result:

AcademicCourse(String CourseID, String CourseName, int CourseDuration, String Level, String Credit, int NumberOfAssessments)

Name of Instance:

new AcademicCourse(,
 ,
 ,
 ,
 ,
)

OK Cancel

Figure 5 Assigning data in AcademicCourse Class

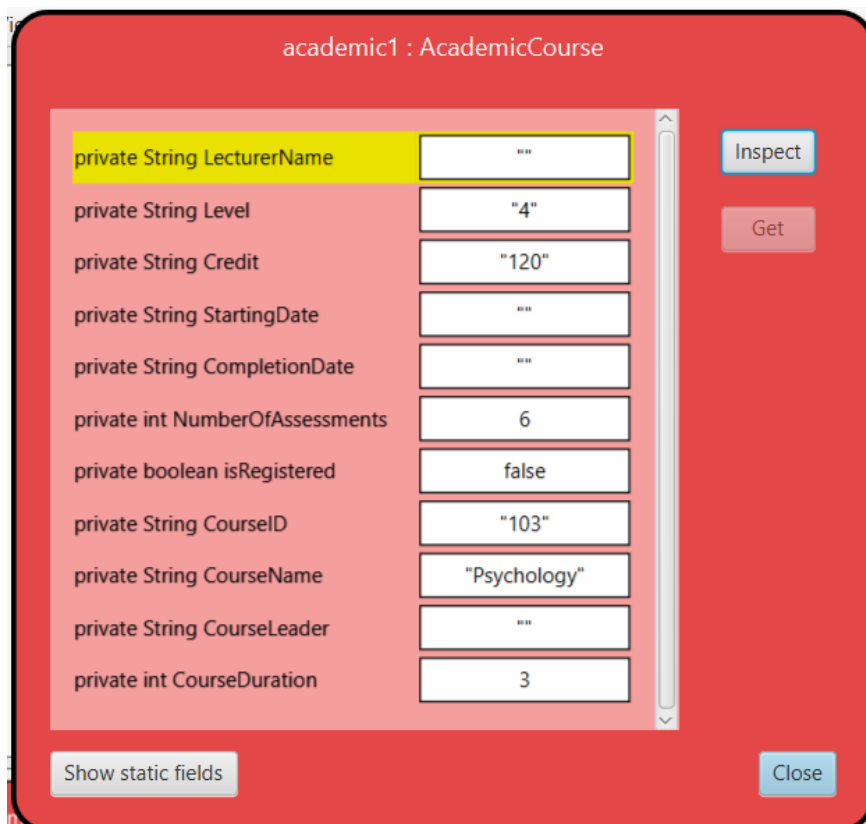


Figure 6 Inspecting AcademicCourse Class

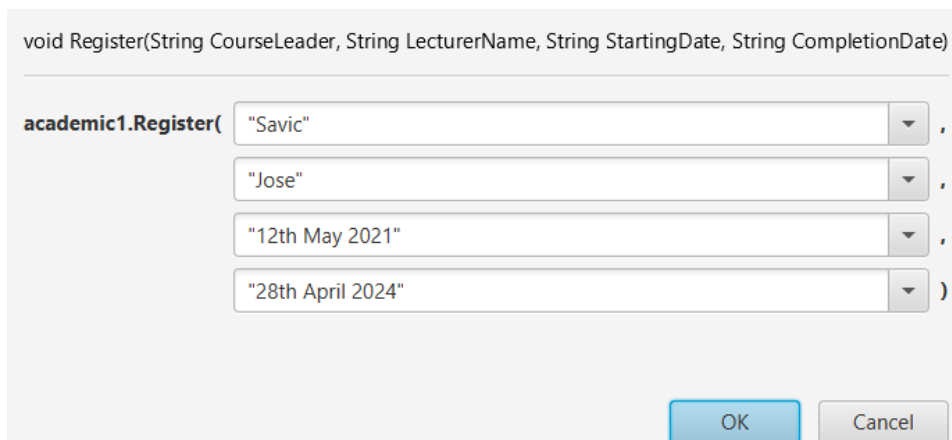


Figure 7 Registering AcademicCourse Class

academic1 : AcademicCourse

private String LecturerName	"Jose"
private String Level	"4"
private String Credit	"120"
private String StartingDate	"12th May 2021"
private String CompletionDate	"28th April 2024"
private int NumberOfAssessments	6
private boolean isRegistered	true
private String CourseID	"103"
private String CourseName	"Psychology"
private String CourseLeader	"Savic"
private int CourseDuration	3

Show static fields

Inspect

Get

Close

Figure 8 Inspecting AcademicCourse Class again

5.2 Test 2 – To inspect NonAcademicCourse class, register a non-academic course and re-inspect the NonAcademicCourse again

Test Number	2
Objective:	To inspect NonAcademicCourse class, register a non-academic course and re-inspect the NonAcademicCourse again
Action:	<ul style="list-style-type: none"> → The NonAcademicCourse is called with the following arguments: CourseID = "321" CourseName = "C++" CourseDuration = 2 Prerequisite = "Fundamentals of Programming" → Inspection of NonAcademicCourse Class is done. → The register method is called with the following arguments: CourseLeader = "Rima" InstructorName = "Pooran" StartingDate = "21st April 2021" CompletionDate = "16th February 2023" ExamDate = "2nd December 2022" → Re-inspection of NonAcademicCourse class is done.
Expected Result:	The non-academic course should be registered.
Actual Result:	The non-academic course was registered.
Conclusion:	The test is successful.

Table 2 Test Table 2

Output Result:

NonAcademicCourse(String CourseID, String CourseName, int CourseDuration, String Prerequisite)

Name of Instance:

new NonAcademicCourse(**,** **,** **,** **)**

Figure 9 Assigning Data in NonAcademicCourse Class

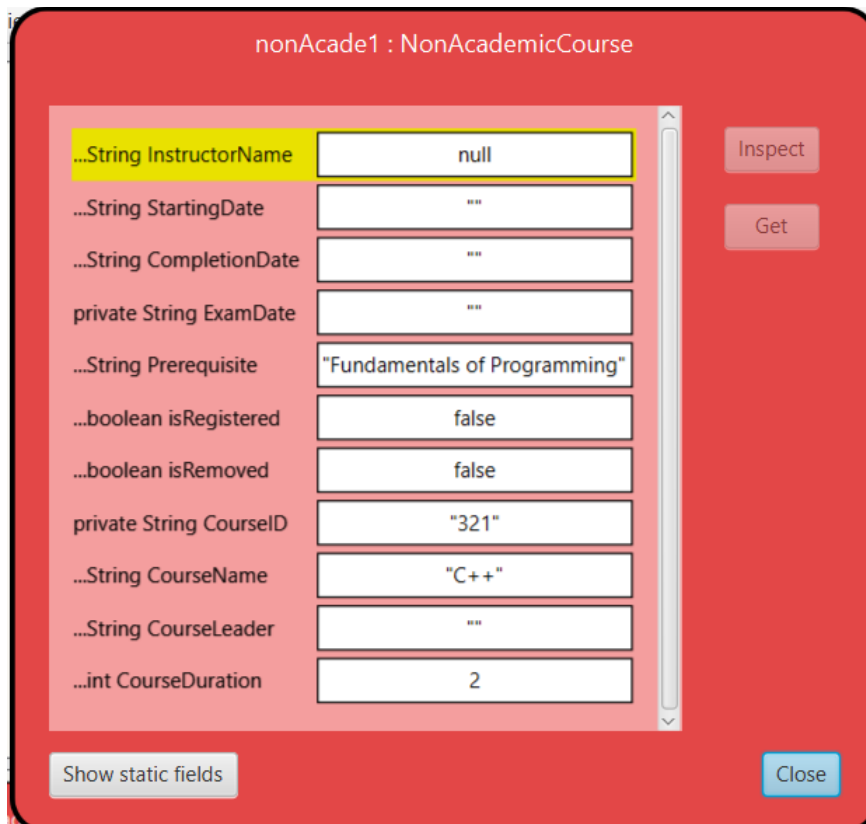


Figure 10 Inspecting NonAcademicCourse Class

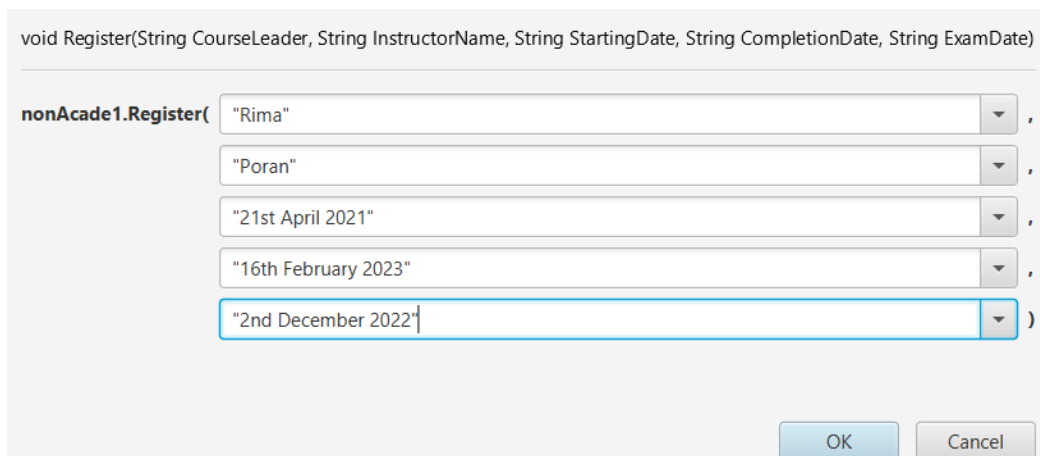


Figure 11 Registering NonAcademicCourse Class

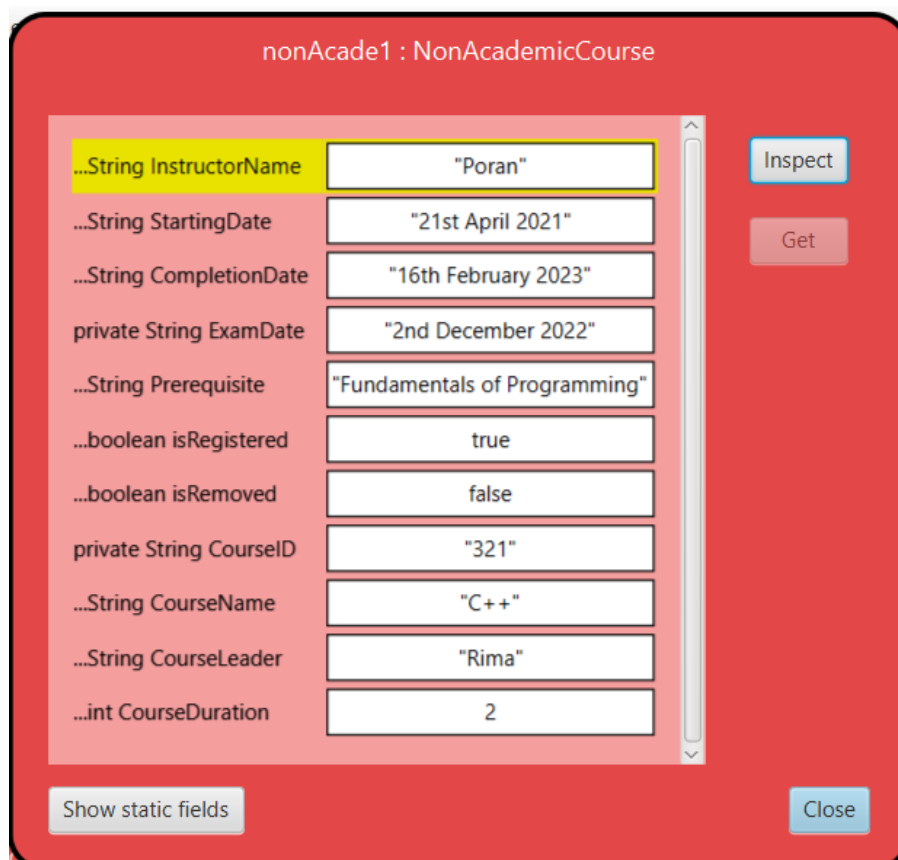


Figure 12 Inspecting NonAcademicCourse Class again

5.3 Test 3 – To inspect NonAcademicCourse class again, change the status of isRemoved to true and re-inspect the NonAcademicCourse

Test Number	3
Objective:	To inspect NonAcademicCourse class again, change the status of isRemoved to true and re-inspect the NonAcademicCourse again
Action:	<ul style="list-style-type: none"> → Inspection of NonAcademicCourse class is done. → The status of isRemoved is changed to true. → Re-inspection of NonAcademicCourse class is done.
Expected Result:	The non-academic course should be removed.
Actual Result:	The non-academic course was removed.
Conclusion:	The test is successful

Table 3 Test Table 3

Output Result:

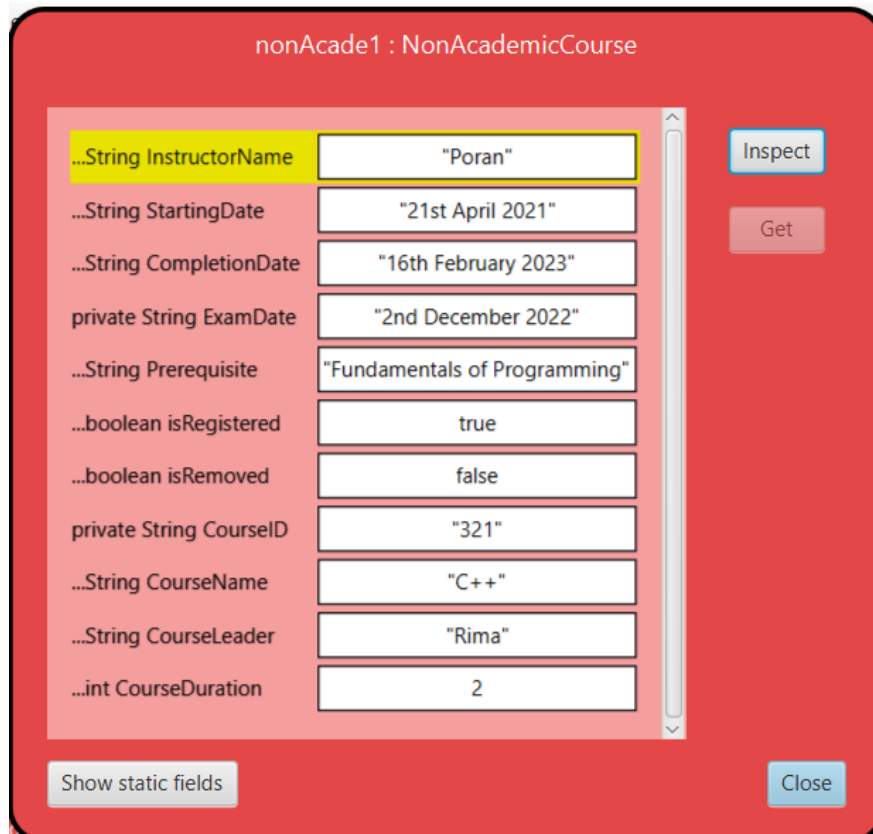


Figure 13 Inspecting NonAcademicCourse Class again

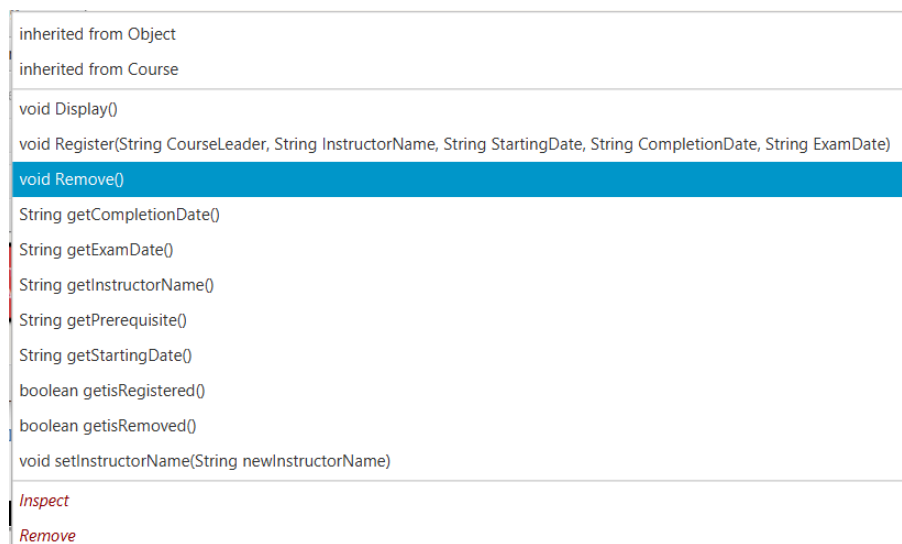


Figure 14 Using remove method

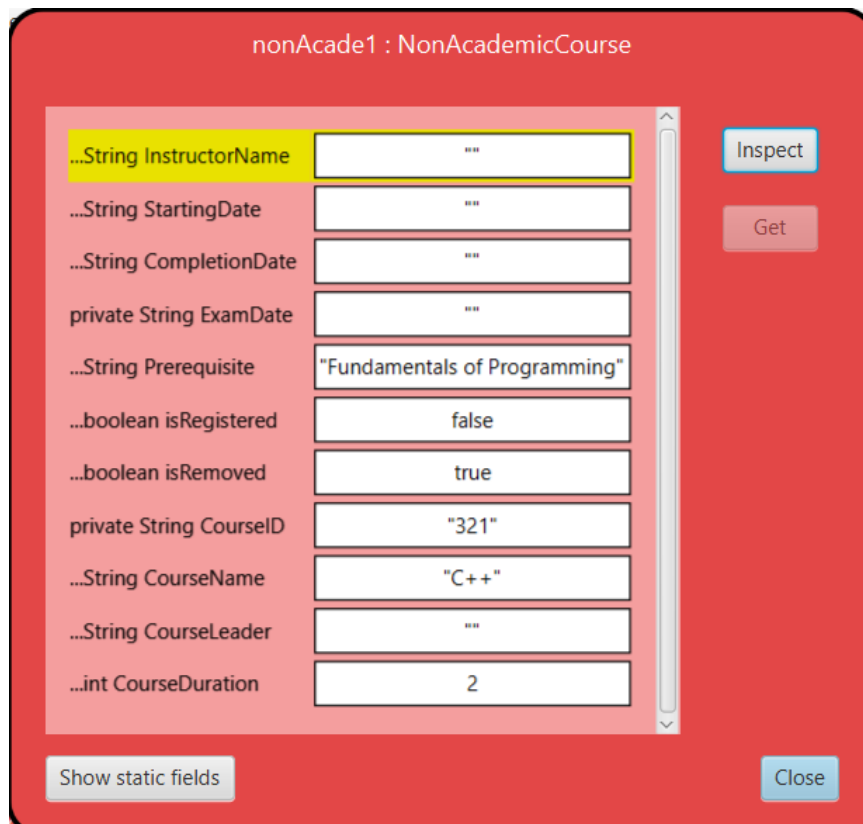


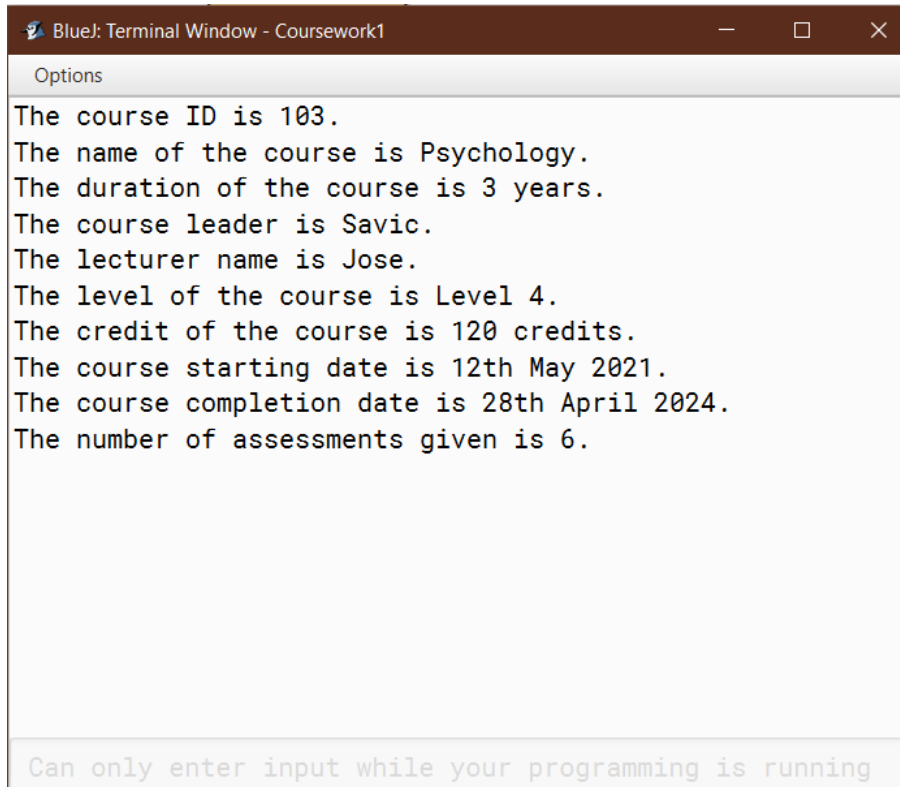
Figure 15 Inspecting NonAcademicCourse Class again

5.4 Test 4 – To display the detail of AcademicCourse and NonAcademicCourse classes

Test Number	4
Objective:	To display the detail of AcademicCourse and NonAcademicCourse classes
Action:	<ul style="list-style-type: none"> → The display method is used in AcademicCourse class. → The display method is used in NonAcademicCourse class.
Expected Result:	The details of AcademicCourse and NonAcademicCourse should be displayed.
Actual Result:	The details of AcademicCourse and NonAcademicCourse was displayed.
Conclusion:	The test is successful.

Table 4 Test Table 4

Output Result:

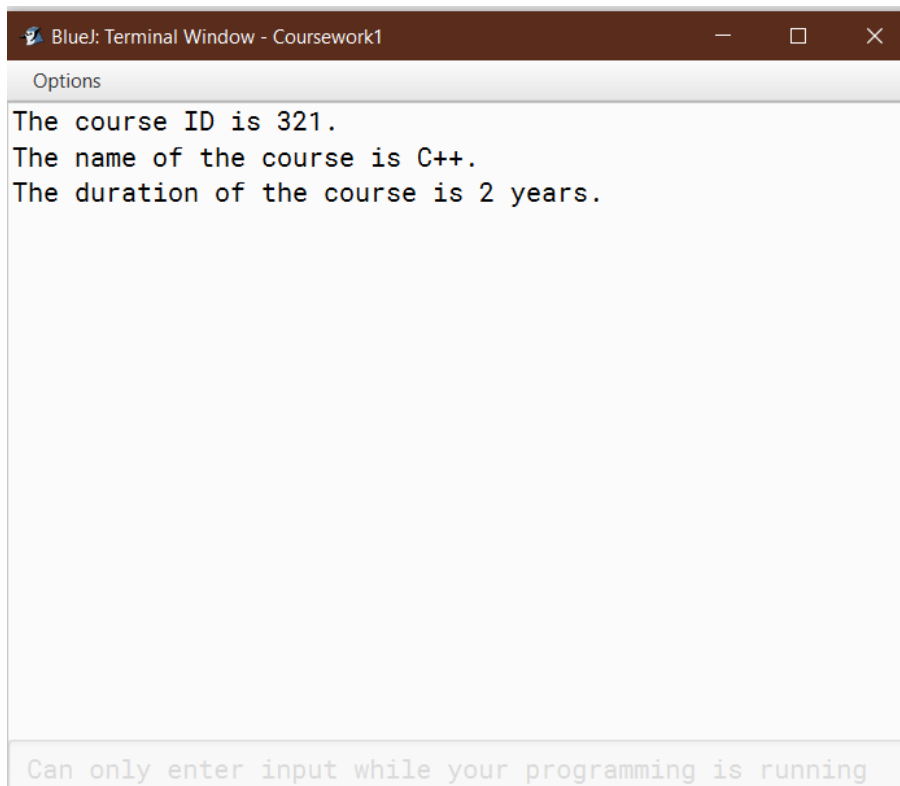


A screenshot of a BlueJ terminal window titled "BlueJ: Terminal Window - Coursework1". The window has a dark brown title bar with standard window controls. Below the title bar is a light gray "Options" bar. The main area of the window is white and contains the following text:

```
The course ID is 103.  
The name of the course is Psychology.  
The duration of the course is 3 years.  
The course leader is Savic.  
The lecturer name is Jose.  
The level of the course is Level 4.  
The credit of the course is 120 credits.  
The course starting date is 12th May 2021.  
The course completion date is 28th April 2024.  
The number of assessments given is 6.
```

At the bottom of the window, there is a light gray status bar with the text: "Can only enter input while your programming is running".

Figure 16 Displaying details of AcademicCourse Class



A screenshot of a BlueJ terminal window titled "BlueJ: Terminal Window - Coursework1". The window has a dark brown title bar with standard window controls. Below the title bar is a light gray "Options" bar. The main area of the window is white and contains the following text:

```
The course ID is 321.  
The name of the course is C++.  
The duration of the course is 2 years.
```

At the bottom of the window, there is a light gray status bar with the text: "Can only enter input while your programming is running".

Figure 17 Displaying Details of NonAcademicCourse Class

6. Different Error Detection and Correction

There were some errors encountered during the coding process.

6.1 Syntax Error

Syntax error is the error when the code is not understood by the compiler because it is not written in the specific manner. Missing a semi colon or not capitalizing a specific word can give out these types of errors. These errors are quite common.

```
public class AcademicCourse extends Course
{
    private String LecturerName;
    private String Level;
    private String Credit;
    private String StartingDate;
    private String CompletionDate;
    private int NumberOfAssessments
    private boolean isRegistered;
```

Figure 18 Syntax Error Detection

```
public class AcademicCourse extends Course
{
    private String LecturerName;
    private String Level;
    private String Credit;
    private String StartingDate;
    private String CompletionDate;
    private int NumberOfAssessments;
    private boolean isRegistered;
```

Figure 19 Syntax Error Correction

6.2 Semantic Error

Semantic error is given when the wrong operator is used in the code. This type of error is also quite common and easy to find out.

```
public void Display()  
{  
    System.out.println("The course ID is " + CourseID + ".");  
    System.out.println("The name of the course is " + CourseName + ".");  
    System.out.println("The duration of the course is " + CourseDuration + ".");  
    if (CourseLeader!= "")  
    {  
        System.out.println("The course leader is " + CourseLeader + ".");  
    }  
}
```

Figure 20 Semantic Error Detection

```
public void Display()  
{  
    System.out.println("The course ID is " + CourseID + ".");  
    System.out.println("The name of the course is " + CourseName + ".");  
    System.out.println("The duration of the course is " + CourseDuration + ".");  
    if (CourseLeader!="")  
    {  
        System.out.println("The course leader is " + CourseLeader + ".");  
    }  
}
```

Figure 21 Semantic Error Correction

6.3 Logical Error

When the code runs smoothly but doesn't give the required output, there are some errors in the code. This is called a logical error. These errors are quite hard to spot when debugging.

```
public void Remove()
{
    if(isRemoved = true)
    {
        System.out.println("The non-academic course has already been remove
    }
    else
    {
        super.setCourseLeader("");
        this.InstructorName = "";
        this.StartingDate = "";
        this.CompletionDate = "";
        this.ExamDate = "";
        this.isRegistered = false;
        this.isRemoved = true;
    }
}
```

Figure 22 Logical Error Detection

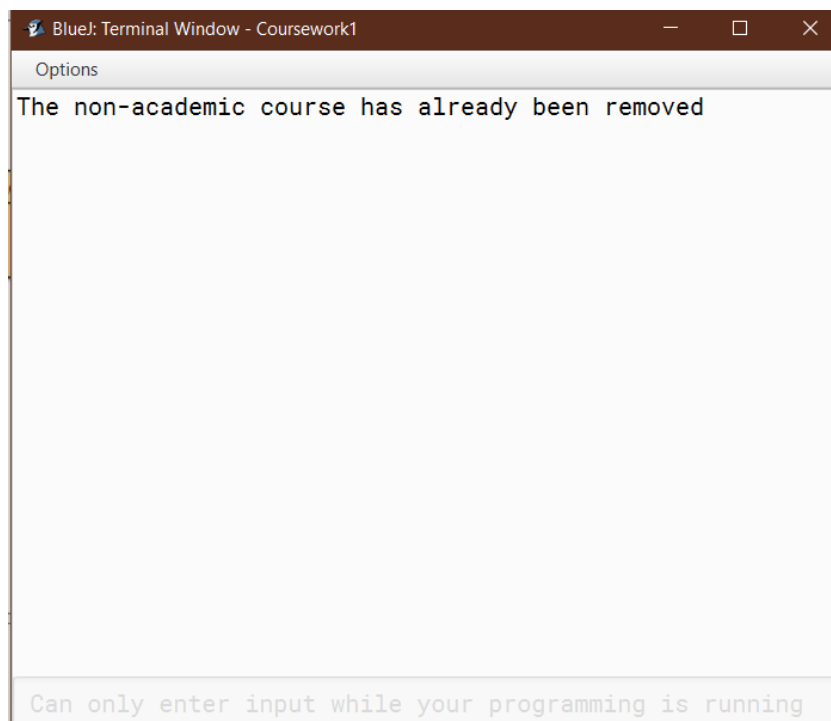


Figure 23 Logical Error Display (1)

```

BlueJ: Terminal Window - Coursework1
Options
The course ID is 123.
The name of the course is Computing.
The duration of the course is 3 years.
The course leader is Niwahang.
The instructor name is Labu.
The starting date is 2021.
The completion date is 2025.
The exam date is 2024.
The prerequisite is BIT.
Can only enter input while your programming is running

```

Figure 24 Logical Error Display (2)

```

public void Remove()
{
    if(isRemoved == true)
    {
        System.out.println("The non-academic course has already been remove
    }
    else
    {
        super.setCourseLeader("");
        this.InstructorName = "";
        this.StartingDate = "";
        this.CompletionDate = "";
        this.ExamDate = "";
        this.isRegistered = false;
        this.isRemoved = true;
    }
}

```

Figure 25 Logical Error Correction

7. Conclusion

7.1 Evaluation of the work

In this given coursework, we had to create and implement a real-world program scenario using the object-oriented concept of Java. We had to create a three-class Course class, AcademicCourse class, and NonAcademicCourse class where Course class is the parent class, and AcademicCourse and NonAcademicCourse class are the child classes.

7.2 What we have learned

Throughout the coursework, we implemented most of the things we have learned in Java in an actual program. During the coding process of the program, some of the things that were confusing at first like the accessor method, use of constructors, and parameters, most of the concepts became clear through trials and error.

7.3 Difficulties encountered

There were a lot of problems regarding the coursework. At first, the question was confusing, and I didn't know where to start. I made many errors in the code. The code didn't run as expected as there were many syntax, semantic and logical errors. I missed to finish the statements with the semicolon or some spelling mistakes.

7.4 How we overcame those difficulties

Even though there were many problems, the teachers guided us and explained everything from typing the codes to writing the report. The teachers would reply to our queries in a short time when we send a mail regarding the coursework. I asked some of my friends about the problems I encountered, and they helped me. I searched the internet for many things for the coursework which I had doubts or problems. Overall, the coursework helped me learn and put the codes to actual use which helped me improve in Java.

Appendix

Course Class

```
public class Course
{
    private String CourseID;
    private String CourseName;
    private String CourseLeader;
    private int CourseDuration;

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

    public String getCourseID()
    {
        return CourseID;
    }

    public String getCourseName()
    {
        return CourseName;
    }

    public String getCourseLeader()
    {
        return CourseLeader;
    }

    public int getCourseDuration()
    {
        return CourseDuration;
    }

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

    public void Display()
```



```

    {
        System.out.println("The course ID is " + CourseID + ".");
        System.out.println("The name of the course is " + CourseName + ".");
        System.out.println("The duration of the course is " + CourseDuration + " years.");
        if (CourseLeader!="")
        {
            System.out.println("The course leader is " + CourseLeader + ".");
        }
    }
}

```

AcademicCourse Class

```

public class AcademicCourse extends Course
{
    private String LecturerName;
    private String Level;
    private String Credit;
    private String StartingDate;
    private String CompletionDate;
    private int NumberOfAssessments;
    private boolean isRegistered;

    public AcademicCourse(String CourseID, String CourseName, int CourseDuration,
String Level, String Credit, int NumberOfAssessments)
    {
        super(CourseID, CourseName, CourseDuration);
        this.Level = Level;
        this.NumberOfAssessments = NumberOfAssessments;
        this.Credit = Credit;
        LecturerName = "";
        StartingDate = "";
        CompletionDate = "";
        isRegistered = false;
    }

    public String getLecturerName()
    {
        return LecturerName;
    }

    public String getLevel()
    {
        return Level;
    }

    public String getCredit()

```

```
{
    return Credit;
}

public String getStartingDate()
{
    return StartingDate;
}

public String getCompletionDate()
{
    return CompletionDate;
}

public int getNumberOfAssessments()
{
    return NumberOfAssessments;
}

public boolean getisRegistered()
{
    return isRegistered;
}

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

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

public void Register(String CourseLeader, String LecturerName, String StartingDate,
String CompletionDate)
{
    if(isRegistered == true)
    {
        System.out.println("The academic course has already been registered.");
        System.out.println("The lecturer name is " + LecturerName + ".");
        System.out.println("The course starting date is " + StartingDate + ".");
        System.out.println("The course completion date is " + CompletionDate + ".");
    }
    else
    {

```

```

        super.setCourseLeader(CourseLeader);
        this.LecturerName = LecturerName;
        this.StartingDate = StartingDate;
        this.CompletionDate = CompletionDate;
        isRegistered = true;
    }
}

public void display()
{
    super.Display();
    if(isRegistered == true)
    {
        System.out.println("The lecturer name is " + LecturerName + ".");
        System.out.println("The level of the course is Level " + Level + ".");
        System.out.println("The credit of the course is " + Credit + " credits.");
        System.out.println("The course starting date is " + StartingDate + ".");
        System.out.println("The course completion date is " + CompletionDate + ".");
        System.out.println("The number of assessments given is " +
NumberOfAssessments + ".");
    }
}
}

```

NonAcademicCourse Class

```

public class NonAcademicCourse extends Course
{
    private String InstructorName;
    private String StartingDate;
    private String CompletionDate;
    private String ExamDate;
    private String Prerequisite;
    private boolean isRegistered;
    private boolean isRemoved;

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

```

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

public String getStartingDate()
{
    return StartingDate;
}

public String getCompletionDate()
{
    return CompletionDate;
}

public String getExamDate()
{
    return ExamDate;
}

public String getPrerequisite()
{
    return Prerequisite;
}

public boolean getisRegistered()
{
    return isRegistered;
}

public boolean getisRemoved()
{
    return isRemoved;
}

public void setInstructorName(String newInstructorName)
{
    if(isRegistered == true)
    {
        System.out.println("Since the instructor name is already registered, it is not
possible to change it.");
    }
    else
    {
        this.InstructorName = newInstructorName;
    }
}
```

```
    }  
}  
  
public void Register(String CourseLeader, String InstructorName, String StartingDate,  
String CompletionDate, String ExamDate)  
{  
    if(isRegistered == true)  
    {  
        System.out.println("The non-academic course has already been registered.");  
    }  
    else  
    {  
        super.setCourseLeader(CourseLeader);  
        this.setInstructorName(InstructorName);  
        this.StartingDate = StartingDate;  
        this.CompletionDate = CompletionDate;  
        this.ExamDate = ExamDate;  
        isRegistered = true;  
        isRemoved = false;  
    }  
}  
  
public void Remove()  
{  
    if(isRemoved == true)  
    {  
        System.out.println("The non-academic course has already been removed");  
    }  
    else  
    {  
        super.setCourseLeader("");  
        this.InstructorName = "";  
        this.StartingDate = "";  
        this.CompletionDate = "";  
        this.ExamDate = "";  
        this.isRegistered = false;  
        this.isRemoved = true;  
    }  
}  
  
public void Display()  
{  
    super.Display();  
    if(isRegistered == true)  
    {
```

```
        System.out.println("The instructor name is " + InstructorName + ".");
        System.out.println("The starting date is " + StartingDate + ".");
        System.out.println("The completion date is " + CompletionDate + ".");
        System.out.println("The exam date is " + ExamDate + ".");
        System.out.println("The prerequisite is " + Prerequisite + ".");
    }
}
```