LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CS4001NI Programming

Assessment Weightage & Type

30% Individual Coursework

Year and Semester

2019-20 Autumn

Student Name: Anish Aryal

Group: C3

London Met ID: 20048825

College ID: NP01CP4S210084

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 submissions will be treated as non-submission and a mark of zero will be awarded.

Table of Contents

List of Figures

List of Tables

Introduction

This introduction covers the idea and goals of this project or assignment. Java is a programming language it is used to develop various programs and software according to the different need in real life scenario. The purpose of this task is to use a scenario of actual problems Object-oriented Java concept, which includes the creation of a class Course, along with two subclasses for an academic and a Courses for non-academics respectively. This report contains the information about the class diagram, pseudo codes and description about different methods used in their respective classes.

This java program was written in Blue-J which is a development kit for programming language java which runs with the help of java development kit (JDK) installed on the system. This java program is very useful for any college or school as this program helps to give the detailed information about the courses both academic and non-academic that are in that organization. It also helps to keep the record of those courses with detailed information. This program also makes it very easier for those organization to update those details if any such situation that the college or school need to update its courses or some of its information. Re-entry of the same course multiple times, entry of wrong detail or information can be prevented. This program also helps the students to know about their courses and the leaders for those courses as well as the lecturers of those courses.

Class Diagram

- **Class Diagram for Course Class**

Course
-courseID: String -courseName: String -duration: int -courseLeader: String
+Course (courseID: String, coursename: String, duration:int) +getcourseID (): String +getcourseName (): String +getduration (): int +getcourseLeader (): String +setcourseLeader (courseLeader: String): void +display (): void

Figure 1: Class diagram for Course class

- **Class Diagram for Academic Course**

AcademicCourse
-lecturerName: String -level: String -credit: String -startingDate: String -completionDate: String -NumberOfAssessment: int -isRegistered: boolean
+AcademicCourse(courseID: String, coursename: String, duration: Int, level: String, credit: String, NumberOfAssessment: int) +getLecturerName(): String +getLevel(): String +getCredit(): int +getStartingDate(): String +getCompletionDate(): +getIsRegistered(): +getNumberOfAssessment: +setLecturerName(LecturerName: String): void +setNumberOfAssessment(NumberOfAssessment: int): void +register():void +display(): void

Figure 2: Class diagram for Academic Course

- **Class Diagram for Non-Academic Course**

NonAcademicCourse
-instructorName: String -duration: int -startDate: String -completionDate: String -examDate: -prerequisite: -isRegistered: boolean -isRemoved: boolean
+NonAcademicCourse(courseID: String, coursename: String, prerequisite: String, duration: int) +getInstructorName(): String +getPrerequisite(): String +getstartDate(): int +getComletionDate(): String +getExamDate(): +getIsRegistered(): +getIsRemoved(): +setInstructorName(instructorName: String): void +register(courseLeader: String, instructorName: String, startDate: String, completionDate: String, examdate: String): void +remove():void +display(): void

Figure 3: Class Diagram for Non-Academic course

- **Class Diagram for Whole Cass**

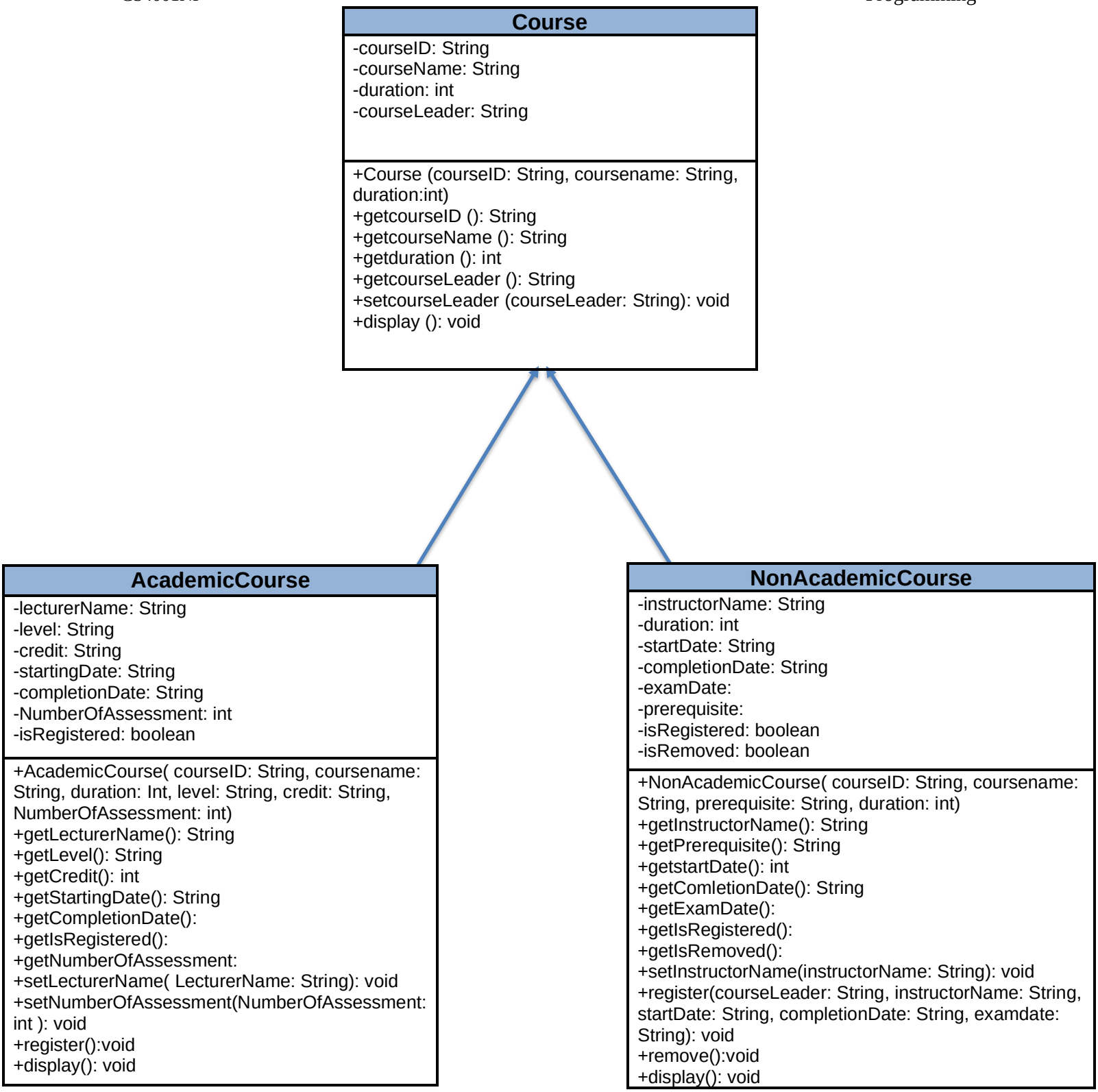

Figure 4: Class Diagram for Whole Class or program

Pseudo Code

1. Pseudo code for Course Class

```
CREATE class Course
        DECLARE instance variable courseID with String type using private modifier
        DECLARE instance variable courseName with String type using private modifier
        DECLARE instance variable courseLeader with String type using private modifier
        DECLARE instance variable duration with int type using private modifier
        CREATE constructor Course (PASS parameters: courseID of String type,
        courseName of String type, duration of int type)
         DO
                ASSIGN parameter courseID to instance variable courseID
                ASSIGN parameter courseName to instance variable courseName
                ASSIGN parameter courseLeader to instance variable courseLeader
                ASSIGN parameter duration to instance variable duration
         ENDDO

         CREATE method getCourseId () with return type String
         DO
                        RETURN courseID
         ENDDO

         CREATE method getCourseName () with return type String
         DO
                       RETURN courseName
         ENDDO

        CREATE method getCourseLeader () with return type String
```

```
 DO
                RETURN courseLeader
ENDDO

CREATE method getDuration () with return type int
DO
               RETURN duration
ENDDO

        CREATE method setCourseLeader (PASS parameters courseLeader of
String type)
 DO
           ASSIGN parameter courseLeader to instance variable courseLeader
 ENDDO

CREATE method display ()
 DO
             PRINT courseID
             PRINT coursename
              PRINT duration
              IF (courseLeader! = "")
                          PRINT courseLeader
                 ENDIF



ENDDO
```

2. Pseudo code for AcademicCourse class

CREATE subclass AcademicCourse for Course Main class

DECLARE instance variable of private modifier LecturerName with String type
DECLARE instance variable of private modifier level with String type
DECLARE instance variable of private modifier credit with String type
DECLARE instance variable of private modifier StartingDate with String type
DECLARE instance variable of private modifier completionDate with String type
DECLARE instance variable of private modifier NumberOfAssessment with int type
DECLARE instance variable of private modifier isRegistered with boolean type

CREATE constructor AcademicCourse (PASS parameters: course_ID of String type, course_name of String type, duration of int type, level of String type, credit of String type, NumberOfAssessment of double type)
DO
CALL of parameters: (course_ID, course_name, duration) from parent class from course constructor
ASSIGN parameter “” to instance variable LecturerName
ASSIGN parameter level to instance variable level
ASSIGN parameter credit to instance variable credit
ASSIGN parameter “” to instance variable StartingDate
ASSIGN parameter “” to instance variable completionDate
ASSIGN parameter false to instance variable isRegisterer
ENDDO
ASSIGN parameter NumberOfAssessment to instance variable NumberOfAssessment
CREATE method getLecturerName () with return type String
DO
RETURN lecturerName
ENDDO

```
CREATE method getLevel () with return type String
DO
        RETURN level
ENDDO

CREATE method getCredit () with return type String
DO
        RETURN credit
ENDDO


CREATE method getStartingDate () with return type String
DO
        RETURN startingDate
ENDDO

CREATE method getCompletionDate () with return type String
DO
        RETURN completionDate
ENDDO

CREATE method getIsRegistered () with return type Boolean
DO
        RETURN isRegistered
ENDDO

CREATE method getNumberOfAssessment () with return type int
DO
        RETURN NumberOfAssessment
ENDDO
```

```
CREATE method setLecturerNAme (PASS parameter lecturerName of String type)
DO
        ASSIGN parameter lecturerName to instance variable lecturerName
ENDDO

CREATE method setNumberOfAssessment (PASS parameter NumberOfAssessment of int type)
DO
        ASSIGN parameter NumberOfAssessment to instance variable NumberOfAssessment
ENDDO

CREATE method registered (PASS parameters: courseLeader with String type, lecturerName with String type, StartingDate with String type, completionDate with String type)
DO
    IF (isRegistered==true)
     PRINT ("Course Leader" calling method getCourseLeader () by super "has started at" StartingDate and completed at" completionDate)
   ELSE
        Call method getCourseLeader () by super
        PASS instance variable LecturerName =local variable LecturerName
        PASS instance variable StartingDate = local variable StartingDate
        PASS instance variable completionDate = local variable completionDate
        PASS instance variable isRegistered = local variable true
    PRINT "New Academic course is Registered"
    ENDIF
 ENDDO

CREATE method display ()
```

```
        super CALL of display () method from parent class
       DO

       IF (isRegistered==true)
         PRINT LecturerName
         PRINT level
         PRINT credit
         PRINT StartingDate
         PRINT completionDate
       ENDIF
     ENDDO
```

3. Pseudo code for NonAcademicCourse class

```
    CREATE subclass NonAcademicCourse for Course main class
        DECLARE instance variable instructorName with String type using private
modifier
        DECLARE instance variable startDate with String type using private modifier
        DECLARE instance variable completionDate with String type using private
modifier
        DECLARE instance variable examDate with String type using private modifier
        DECLARE instance variable prerequisite with String type using private modifier
        DECLARE instance variable isRegistered with Boolean type using private
modifier
        DECLARE instance variable isRemoved with Boolean type using private
modifier

    CREATE constructor NonAcademicCourse (PASS parameters: courseID of String
    type, courseName of String type, duration of float type, prerequisite of String type)
        CALL of parameters: (courseID, courseName, duration) by super
```

```
DO
    ASSIGN parameter Duration to instance variable Duration
    ASSIGN parameter "" to instance variable startDate
    ASSIGN parameter "" to instance variable completionDate
    ASSIGN parameter “” to instance variable examDate
    ASSIGN parameter prerequisite to instance variable prerequisite
    ASSIGN parameter false to instance variable isRegistered
    ASSIGN parameter false to instance variable isRemoved
ENDDO

 CREATE method getInstructorName () with return type String
            DO
                RETURN instructorName
            ENDDO
    CREATE method getDuration () with return type float
            DO
                RETURN duration
            ENDDO
   CREATE method getStartDate () with return type String
            DO
                RETURN startDate
            ENDDO
   CREATE method getCompletionDate () with return type String
            DO
                RETURN completionDate
            ENDDO
    CREATE method getExamDate () with return type String
            DO
                RETURN examDate
            ENDDO
   CREATE method getPrerequisite () with return type Boolean
```

```
            DO
                RETURN prerequisite
            ENDDO
CREATE method getIsRegistered () with return type Boolean
            DO
                RETURN isRegistered
            ENDDO
CREATE method getIsRemoved () with return type Boolean
            DO
                RETURN isRemoved
            ENDDO

CREATE method setInstructorName () (PASS parameter instructorName of String type)
        DO
            IF (isRegistered==false)
                SET instructorName as a new value
                PRINT "New Instructor name is obtained"
            ELSE
                PRINT "Instructor's name is already registered and cannot be changed"
            ENDIF
        ENDDO
CREATE method register (PASS parameters: courseLeader with String type,
instructorName with String type, startDate with String type, completionDate with String
type)
        DO
            IF (isRegistered==false)
                Assign instructorName as a new value passing parameter
                Assign startDate as new value
                Assign completionDate as new value
                Assign examDate as new value
                Assign isRegistered as true
```

```
            ELSE
                PRINT "Non-Academic Course is already registered"
            ENDIF
        ENDDO

CREATE method remove ()
        DO
            IF (isRemoved==true)
                PRINT "Non-Academic Course is removed"
            ELSE
                Call method getCourseLeader () by super
                    PASS instance variable instructorName to local variable “”
                    PASS instance variable startDate to local variable ""
                    PASS instance variable completionDate to local variable ""
                    PASS instance variable isRegistered to local variable false
                    PASS instance variable isRemoved to local variable true
            ENDIF
        ENDDO

CREATE method display ()
  super CALL of display ()
        DO
      IF (isRegistered==true)
            PRINT instructorName
            PRINT startDate
            PRINT completionDate
            PRINT examDate
      ENDIF
  ENDDO
```

Method Description

1. Course Class

- Following are the Methods used in Course class:

1. **getcourseID()**

   This is a getter method , this method is used to retrieve the value of the courseID when the value of courseID is assigned . The data type of courseID is string.

2. **getcourseName()**

   This method is known as the getter method or accessor. This method is used to retrieve the value of the courseName. Also, this method returns the value of course name when courseName is called. The data type of coursename is String.

3. **getduration()**

   This method is also known as the getter method. It is used to retrieve the value of the the duration of course. It also returns the value when the method is called and data type for duration is int

4. **getcourseLeader()**

   This method is also known as the getter method for the information about course leader . it is used to return the value of the course leader. This method returns the value when it is called. Data type for courseLeader is string.

5. **setcourseLeader()**

This method is known as setter method or mutator method. This method is used to set or update the value of course leader. When the the value is updated getter method returns the value of course leader.

6. **display()**

   This method is used to display the value of courseID, courseName, duration and courseLeader if it is not the empty string. When this method is called the program displays a suitable output for the user.

2. AcademicCourse Class

- Following are the Methods used in AcademicCourse class:

1. **getLecturerName ()**

   This method is known as the getter method or accessor. This method helps to retrieve the value of the LecturerName. Also, this method returns the value of course name when LecturerName is called. The data type of LecturerName is String.

2. **getLevel ()**

   This is a getter method , this method is used to retrieves the value of the Level when the value of Level is assigned . The data type of Level is string.

3. **getCredit ()**

   This method is also known as the getter method. It is used to retrieve the value of the the Credit of course. It also returns the value when the method is called and data type for Credit is string.

4. **getStartingDate ()**

This getStartingDate() method is also known as the getter method for the information about starting date of course . it is used to return the value of the starting date of academic course. This method is used in returning the value for starting date when it is called. Data type for StartingDate is string.

5. **getCompletionDate ()**

This getCompletionDate() method is also known as the getter method for the information about completion date of course . it is used to return the value of the completion date of academic course. This method is used in returning the value for completion date when it is called. Data type for CompletionDate is string.

6. **getisRegistered ()**

getisRegistered () method is the getter method. This method helps to return the value of isRegistered when it is called. The data type of isRegistered is boolean.

7. **getNumberOfAssessment ()**

This getNumberOfAssessment is an accessor method. It is used for returning the value of NumberOfAssessment, it returns the value when it is called. Its data type is double.

8. **setLecturerName ()**

This setLecturerName() method helps to set or update the value of lecturerName the preferred date type for LecturerName is string. It is a mutator method. After the value is set getter method is used to return the value to LecturerName.

9. **SetNumberOfAssesment()**

   This setNumberOfAssesment() method helps to set or update the value of NumberOfAssesment, the preferred date type for NumberOfAssesment is double. It is a mutator method. After the value is set getter method is used to return the value to NumberOfAssesment.

10. **register()**

    This method is used to register the Academic course if the course is already registered it displays suitable output and if it is not it registers the code and displays the suitable output. It has different parameters.

11. **display()**

    At first this method makes a call to the super class Course and a method display () is called from it to display the courseID, course name, duration, and course leader and If the academic course is already registered then lecturer name, level, credit, starting date, completion date and number of assessments are also displayed.

3. NonAcademicCourse Class

- <u>Following are the Methods used in NonAcademicCourse class:</u>

1. **getInstructorName ()**

   This method is known as the getter method or accessor. This method helps to retrieve the value of the InstructorName. Also, this method returns the value of course name when InstructorName is called. The data type of InstructorName is String.

2. **getPrerequisite ()**

This is a getter method , this method is used to retrieves the value of the Prerequisite when the value of Prerequisite is assigned . The data type of Prerequisite is string.

3. **getStartingDate ()**

This getStartDate() method is also known as the getter method for the information about starting date of course . it is used to return the value of the starting date of non-academic course. This method is used in returning the value for starting date when it is called. Data type for StartDate is string.

4. **getCompletionDate ()**

This getCompletionDate() method is also known as the getter method for the information about completion date of non-academic course . it is used to return the value of the completion date of non-academic course. This method is used in returning the value for completion date when it is called. Data type for CompletionDate is string.

5. **getExamDate ()**

This method is also known as the getter method. It is used to retrieve the value for the ExamDate for non-academic course. It also returns the value when the method is called and data type for ExamDate is string.

6. **getisRegistered ()**

getisRegistered () method is the getter method. This method helps to return the value of isRegistered when it is called to the non-academic course class. The data type of isRegistered is boolean.

7. **getisRemoved ()**

This getisRemoved is an accessor method. It is used for returning the value of isRemoved which helps us to know if the course is removed or not, it returns the value when it is called. Its data type is boolean.

8. **setInstructorName ()**

This setInstructorName() method helps to set or update the value of InstructorName the preferred date type for InstructorName is string. It is a mutator method. This method sets the instructor's name if the value of is registered is false but if it's true a message is displayed saying it is already registered.

9. **register ()**

This method is used to register the Non-Academic course if the course is already registered it displays suitable output and if it is not it registers the code and displays the suitable output. It has different parameters.

10. **remove ()**

This method is used to know if the course is registered or not. It is a getter method. This method tells if the course is removed or not by checking if isremoved status is true or false according to the condition suitable output is displayed.

11. **display ()**

At first this method makes a call to the super class Course and a method display () is called from it to display the courseID, course name, duration, and course leader and If the academic course is already registered then instructor name, start date, completion date and exam dates are also displayed are also displayed.

Testing

Test 1:

Inspect AcademicCourse class, register an academic course, and re-inspect the AcademicCourse Class

Objective	• To Inspect AcademicCourse class and register an academic course then re-inspect the AcademicCourse Class.
Action	• AcademicCourse is called with the following agruments: 1. courseID= “PR12345” 2. courseName= “Programming” 3. duration= 27 4. level= “Bachelors” 5. credit= “160 Hours” 6. NumberOfAssessment= 17 • Inspection of AcademicCourse • Void register method is called with following arguments: 1. courseLeader= “Veldora” 2. lecturerName= “Natsu Uchiha” 3. startingDate= “2021-05-09” 4. completionDate= “2022-03-21” • Re-inspection of AcademicCourse
Expected Result	• New Academic Course would be registered as Programming course
Actual Result	• New Academic course is registered
Conclusion	• The test is successful.

Table 1:inspect the Academic course Class, register and reinspect Academic course class

Creating an object for Academic Course

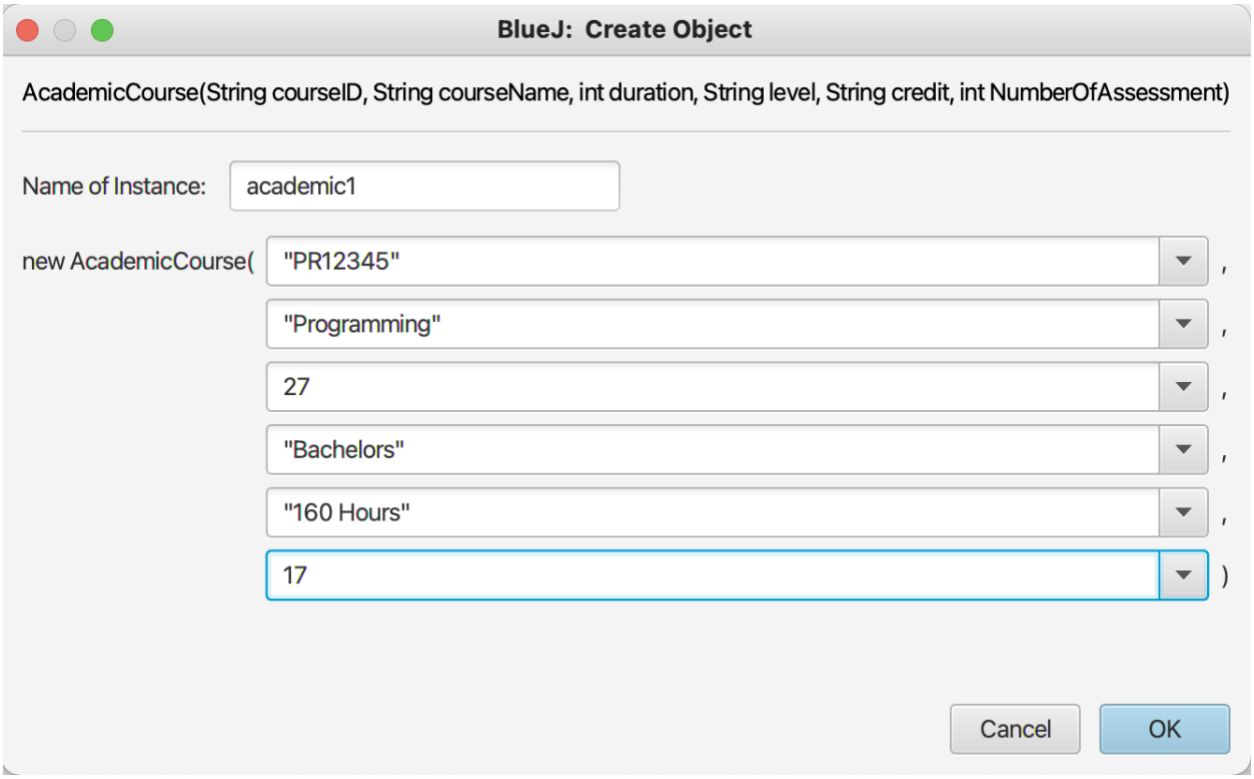

Figure 5:Creating object for Academic Course

Inspecting Academic Course

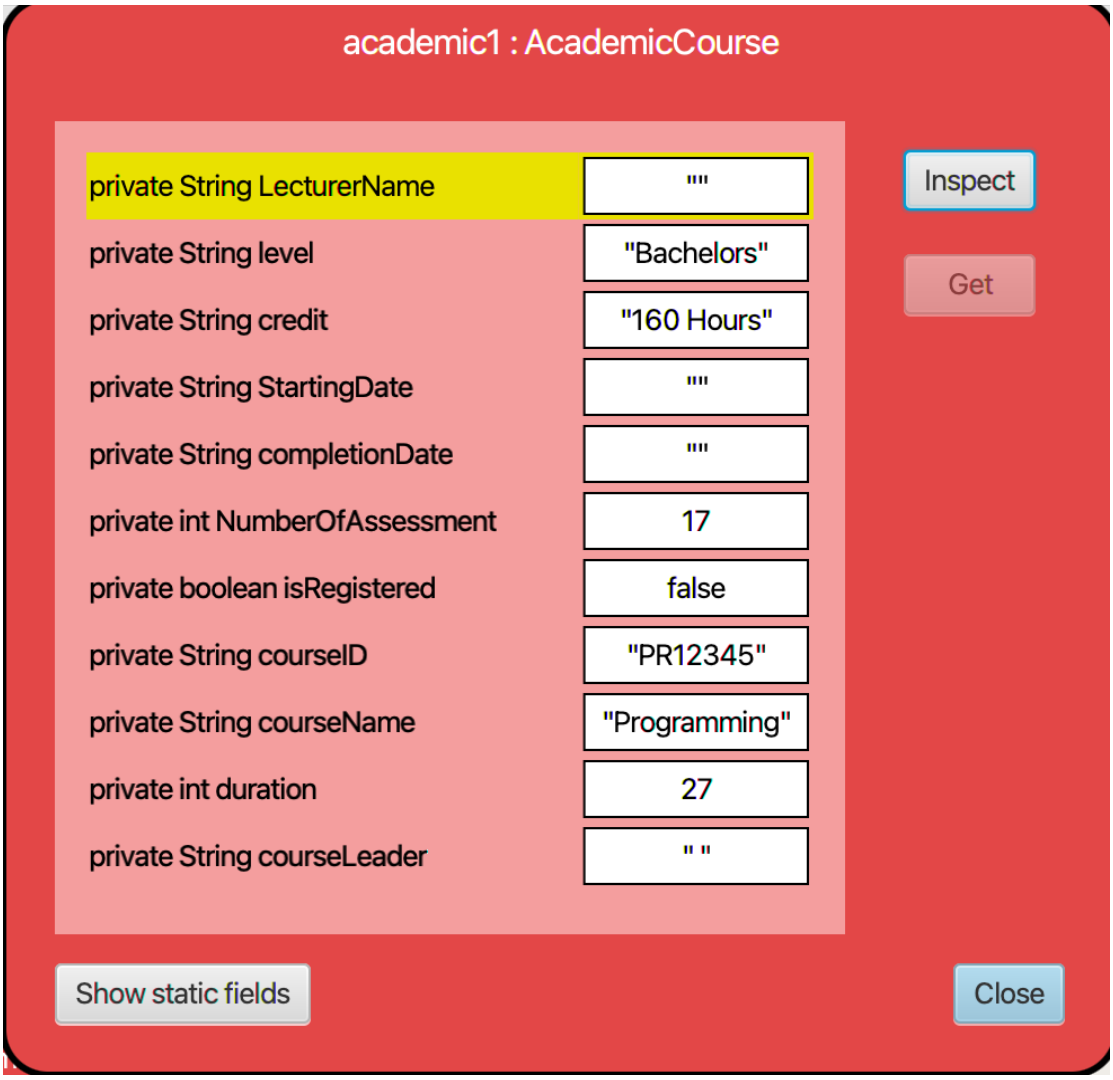

Figure 6: Inspecting Academic Course

Method call to register academic course

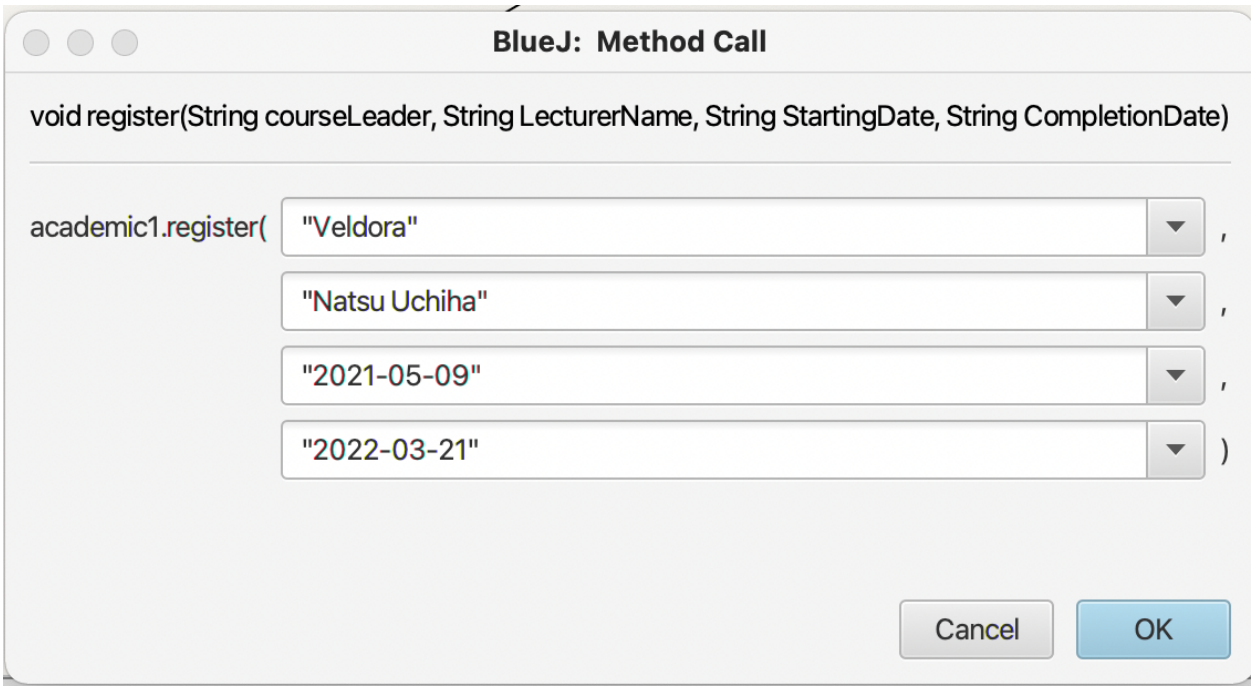

Figure 7: Method call to register academic course

Re-inspecting Academic course object

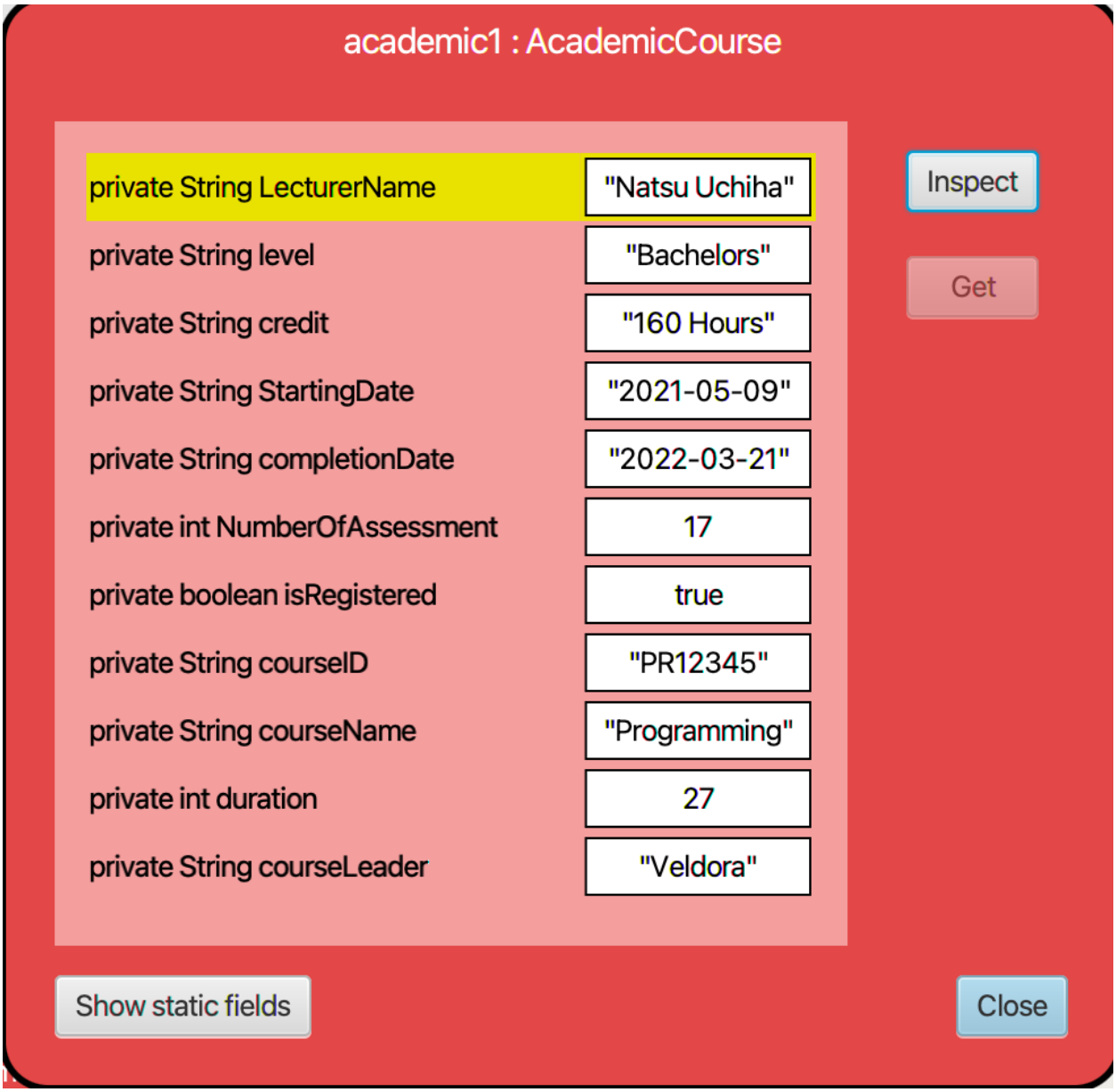

Figure 8: Re-Inspecting academic course

Test 2:

Inspect NonAcademicCourse class, register a non-academic course

and re-inspect the NonAcademicCourse Class

Objective	• To Inspect NonAcademicCourse class and register a non-academic course then to re-inspect the NonAcademicCourse Class
Action	• AcademicCourse is called with the following agruments: 1. courseID= “ADBS12” 2. courseName= “Advance Database” 3. prerequisite= “Fluent English” 4. duration= 15 • Inspection of NonAcademicCourse • Void register method is called with following arguments: 1. courseLeader= “Kakashi Hatake” 2. instructorName= “Tobi” 3. startDate= “2021-05-12” 4. completionDate= “2022-03-13” 5. examDate= “2022-03-05” • Re-inspection of NonAcademicCourse
Expected Result	• New Non-Academic course would be registered with respective variables values
Actual Result	• New Non-Academic course was registered
Conclusion	• The test is successful.

Table 2: Inspect academic course register and reinspect nonacademic course

Creating an object for Non-Academic Course

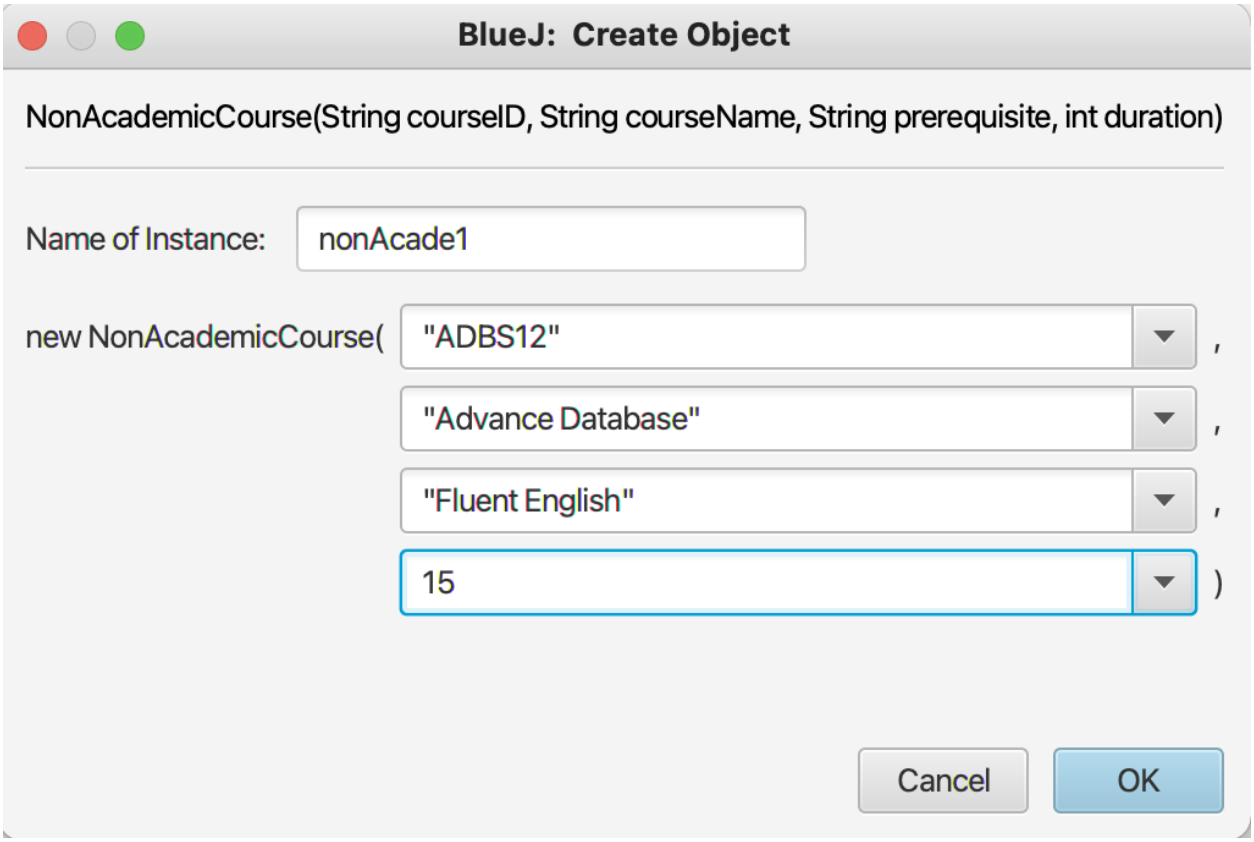

Figure 9: Creating an object for Non-Academic Course

Inspecting Non-Academic Course

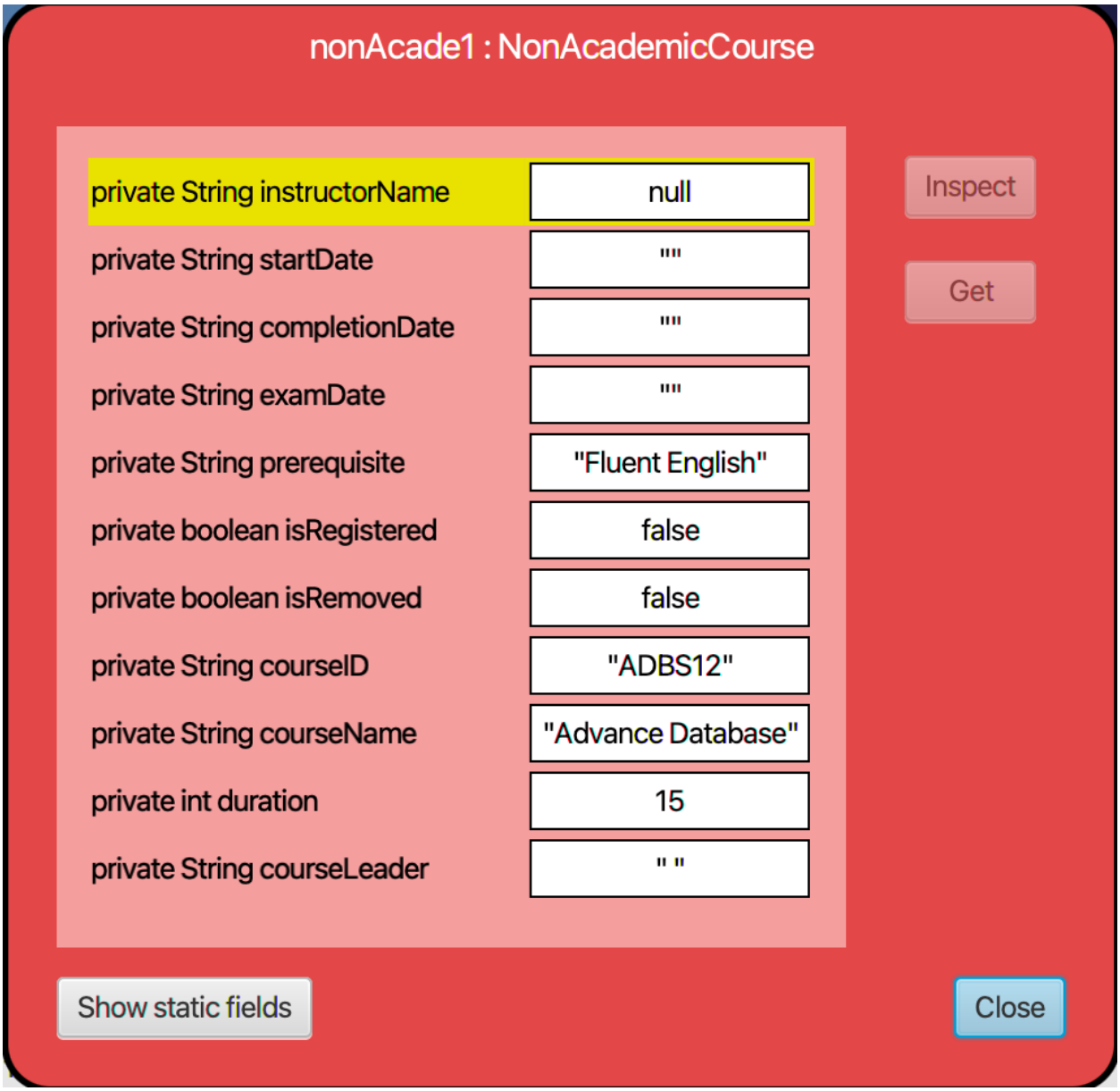

Figure 10: Inspecting Non-Academic Course

Method calls to register Non-Academic Course

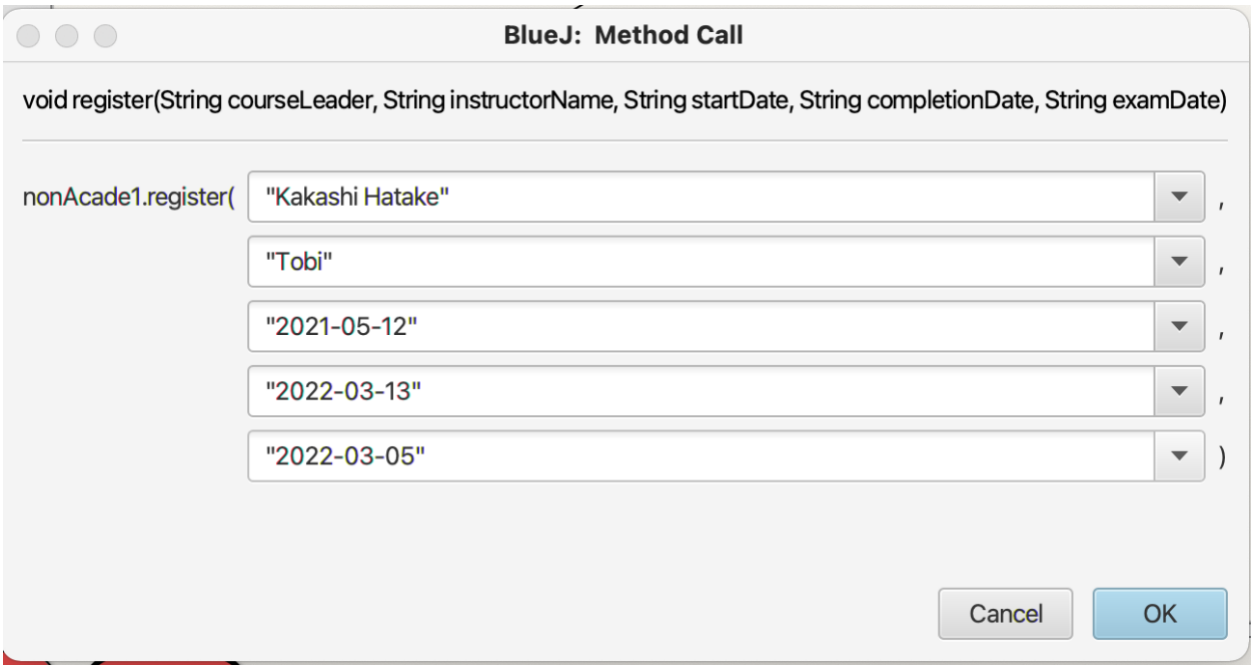

Figure 11:Method calls to register Non-Academic Course

Re-inspecting Non-Academic course object

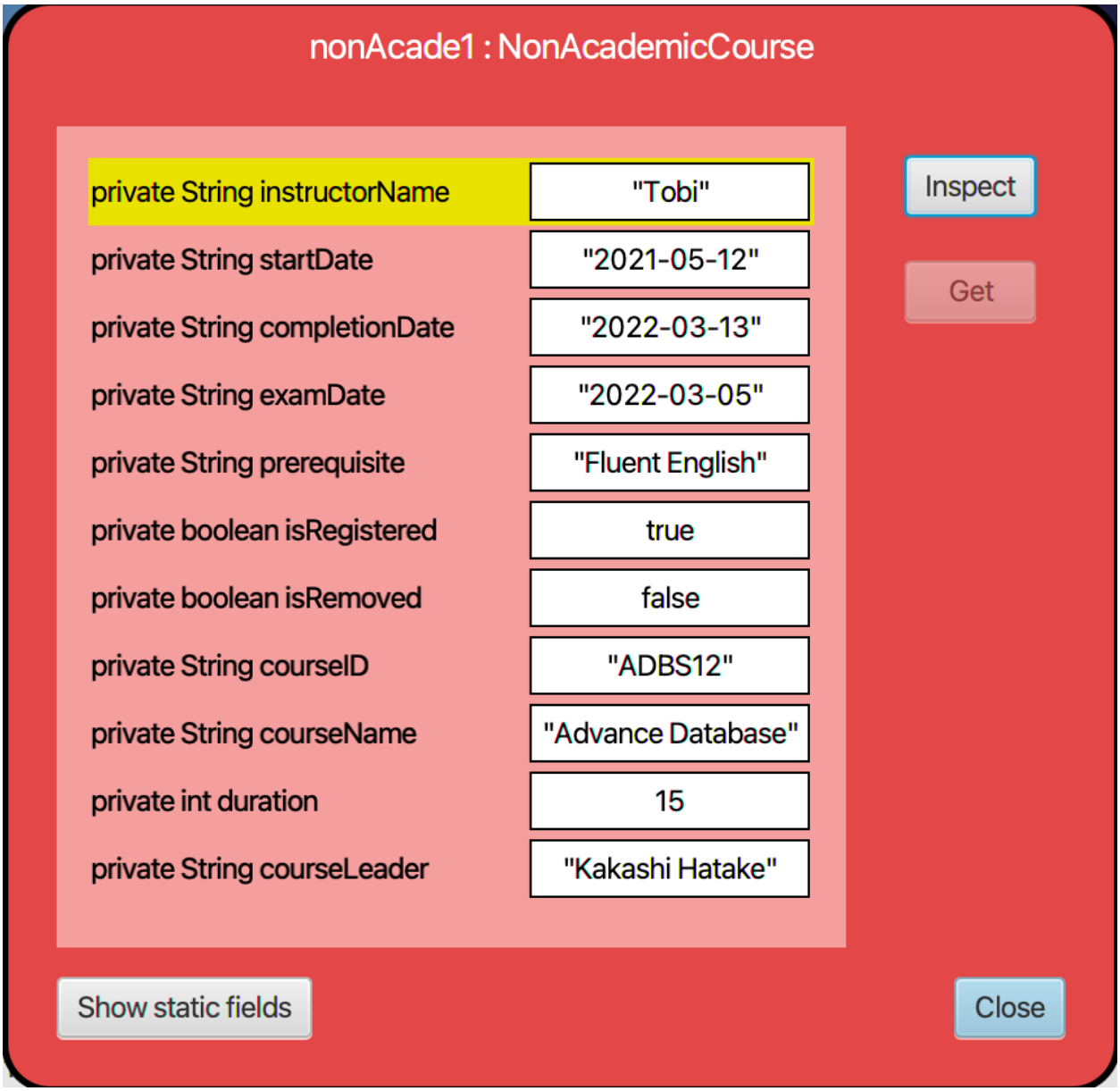

Figure 12: Re-inspecting Non-Academic course

Test 3:

Inspect NonAcademicCourse class again, change the status of

isRemoved to true and re-inspect the NonAcademicCourse class

Objective	• Change the status of isRemoved to true and re-inspect the NonAcademicCourse class
Action	• Inspect Non-Academic Course • Call void remove method Re-Inspection of Non AcademicCourse
Expected Result	• Non-Academic course information would be removed
Actual Result	• Non-Academic Course information was removed
Conclusion	• The test is successful.

Table 3: Change the status of is removed to true and re-inspect Non-Academic Course

Inspecting Non-Academic course object

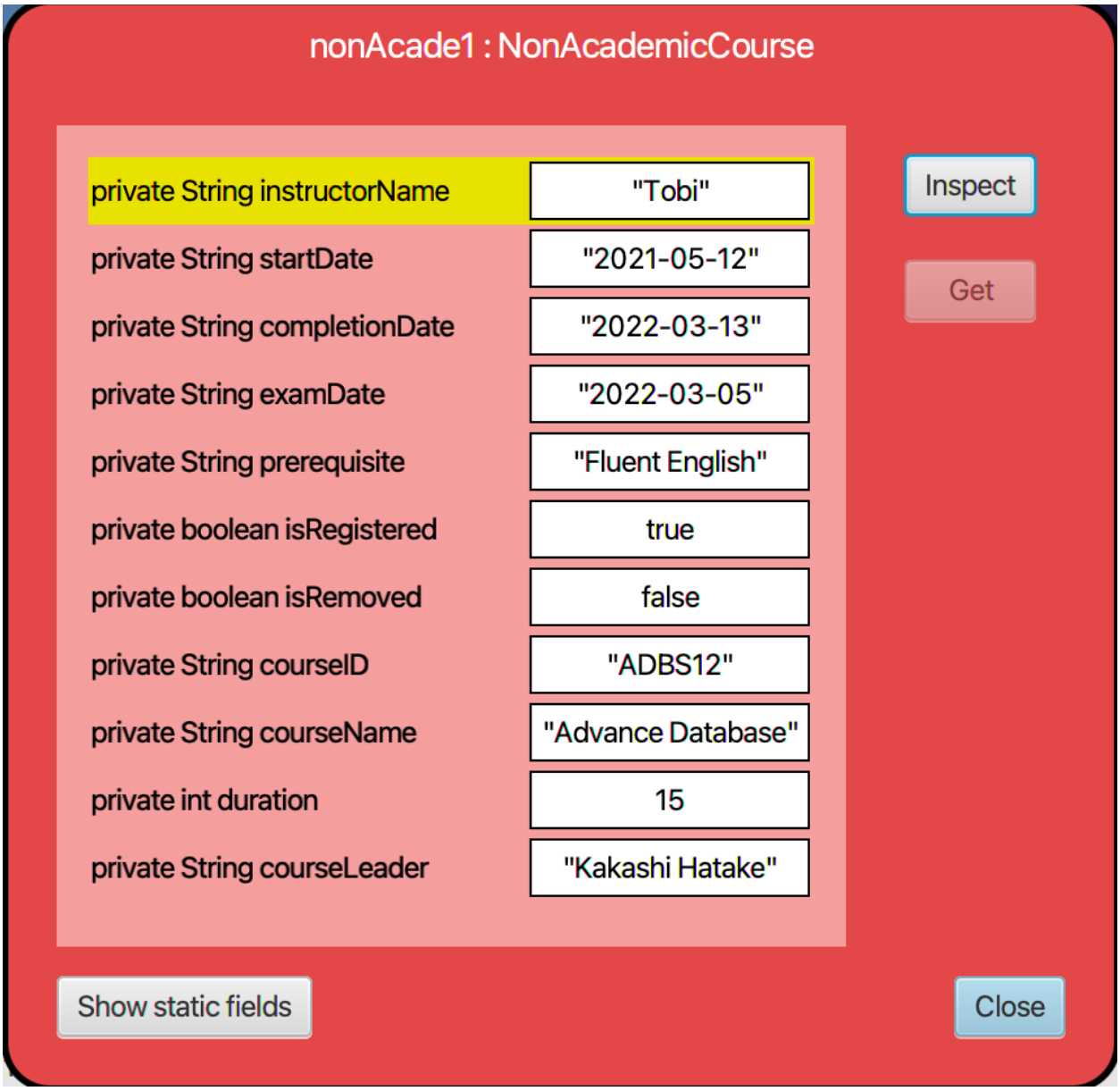

Figure 13: Inspecting Non-Academic Course

Re-inspecting Non-Academic Course Class after changing the status of isRemoved to true

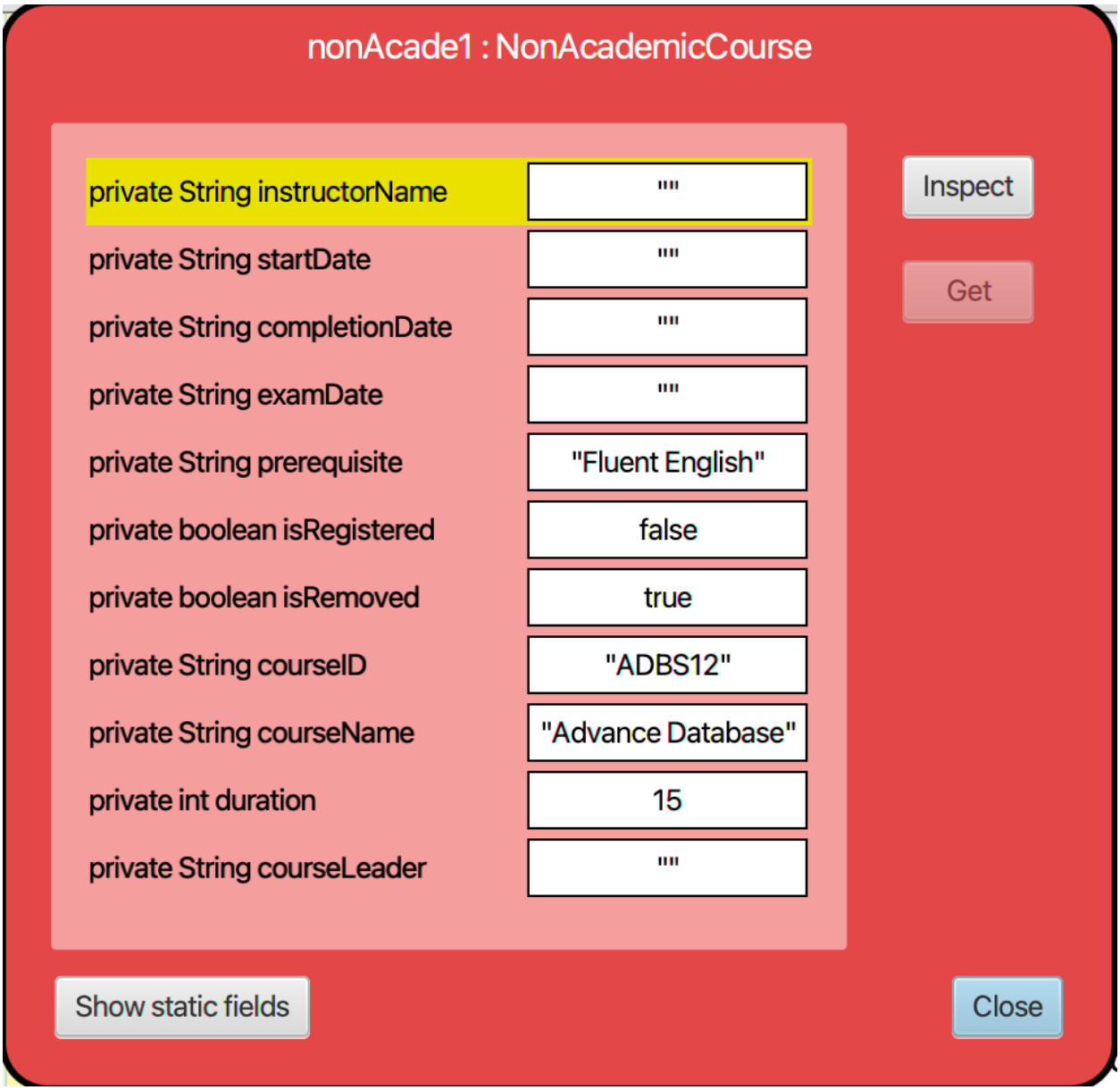

Figure 14:Re-inspecting nonacademic course after changing the status of isRemoved to true

Test 4:

Display the detail of AcademicCourse and NonAcademicCourse

classes.

Details of Academic course

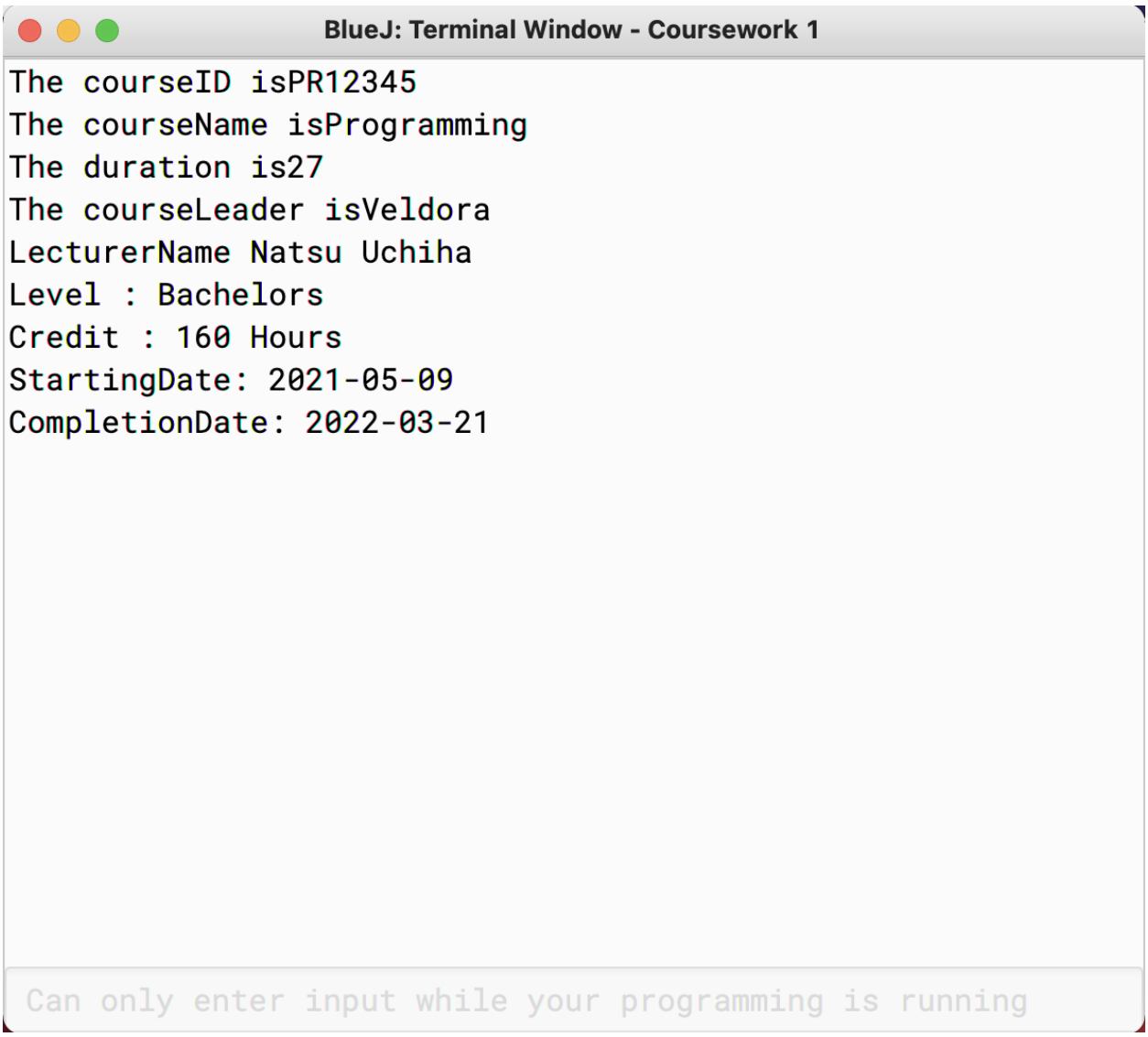

Figure 15: Details Of Academic Course Class

Details of Non-Academic course

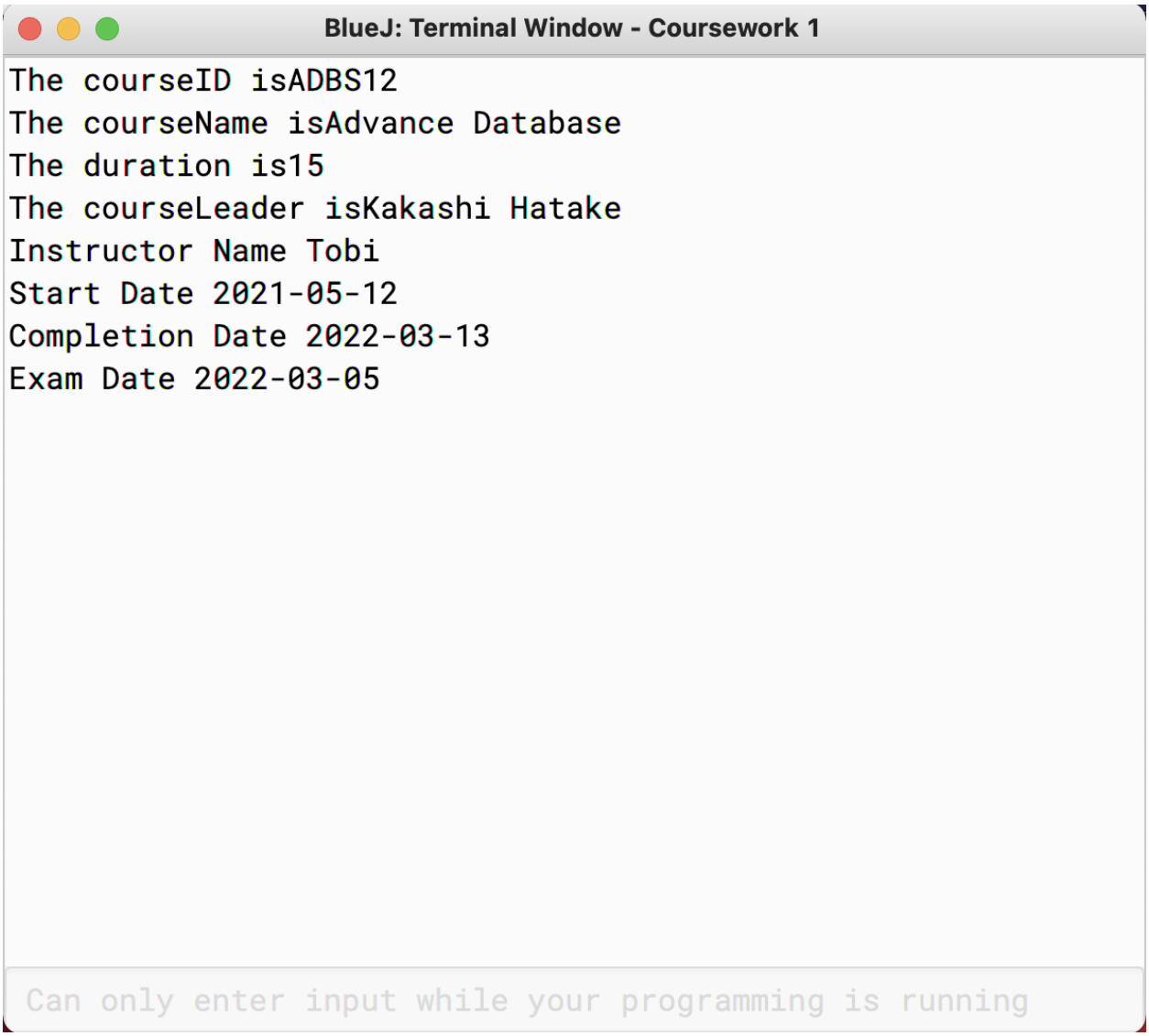

Figure 16: Details of Non-Academic Course

Errors

1. Syntax Error

Syntax error can be referred as the mistake made by the programmer while writing the java program, we can also say that the syntax error is the grammatical error of java programming language. This kind of error is easy to spot and solve because almost all the errors are found by compilers we don't need to look after each and every code to find the mistake.

```
public class Course {
    //declaring instance variables for Course class
     private String courseID;
     private string courseName;
     private int
     private Stri
    //creating an
     public Course(String courseID, String courseName, int duration) {
         this.courseID = courseID;
```

Unknown type: string
• Fix: Correct to: String (java.lang package)
• Fix: Correct to: Spring (javax.swing package)

Figure 17: Example of Syntax Error

This is one of the examples of syntax error in the image above we can see that the error and the line of error is highlighted. Java is a case sensitive language so we must follow the rules while writing the program. First letter of String should always be capital whereas in the image above

Correction of Syntax Error

```
public class Course {
    //declaring instance variables for Course class
     private String courseID;
     private String courseName;
     private int duration;
     private String courseLeader;
    //creating and defining paramaterized constructor
     public Course(String courseID, String courseName, int duration) {
         this.courseID = courseID;
```

Figure 18: Correction of Syntax Error

2. Logical Error

Logical error are the errors that the programmers make while writing the program. It causes the program to malfunction or function improperly. In such case of logical error program compiles without showing any error but during the result we get the wrong result or output. If such cases were to happen, we can know that it is a logical error. it is difficult to find where the error is in the program.

This is one of the examples of logical error

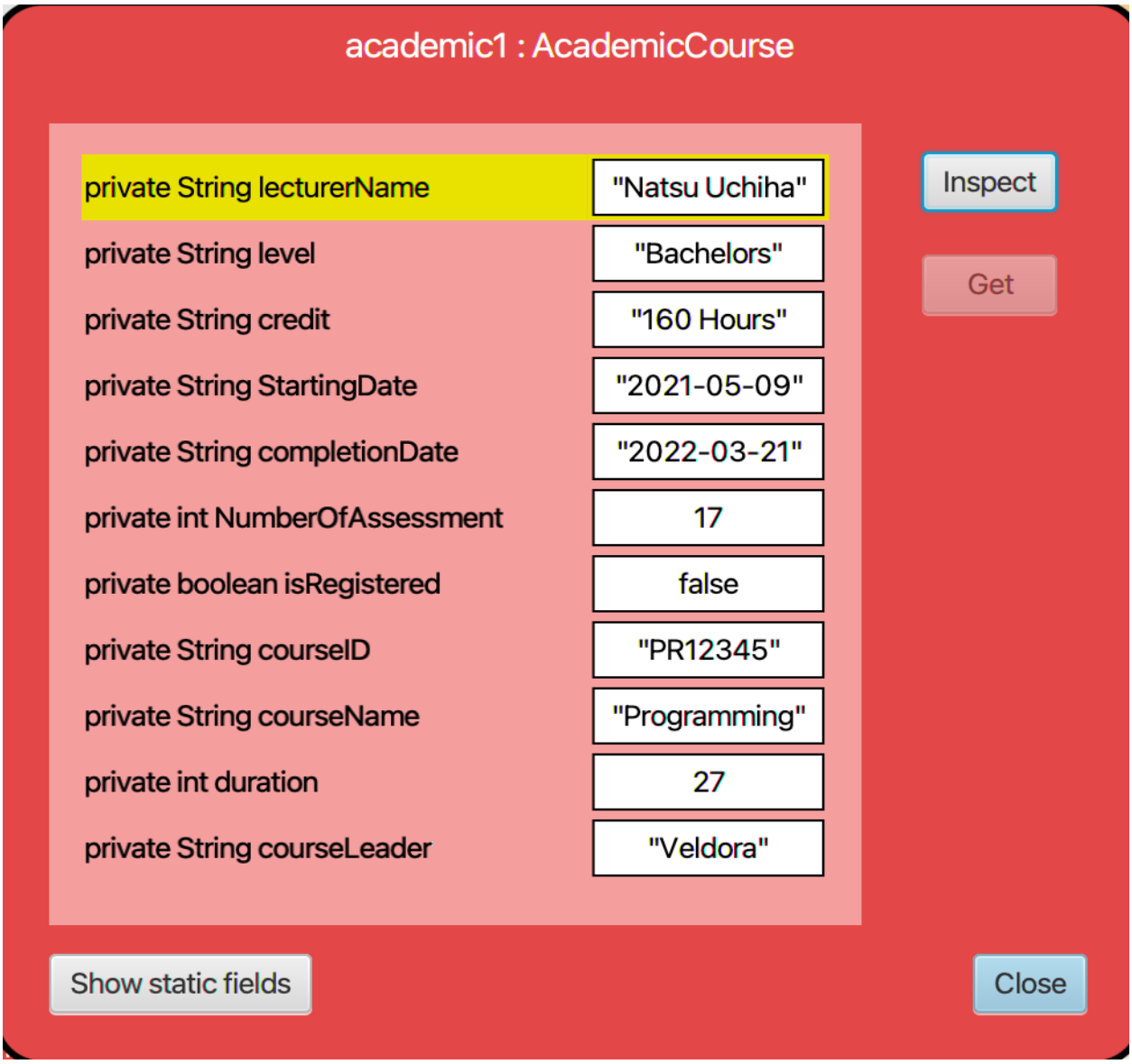

Figure 19: Logical error example

Here, the value of IsRegistered is false which may be due to the logical error in code because if the value of Is registered is always false while calling the method then the same course can be registered multiple times, which destroys the programs one of the sole purposes. The below mentioned value was inserted into the register method call when we got this error.

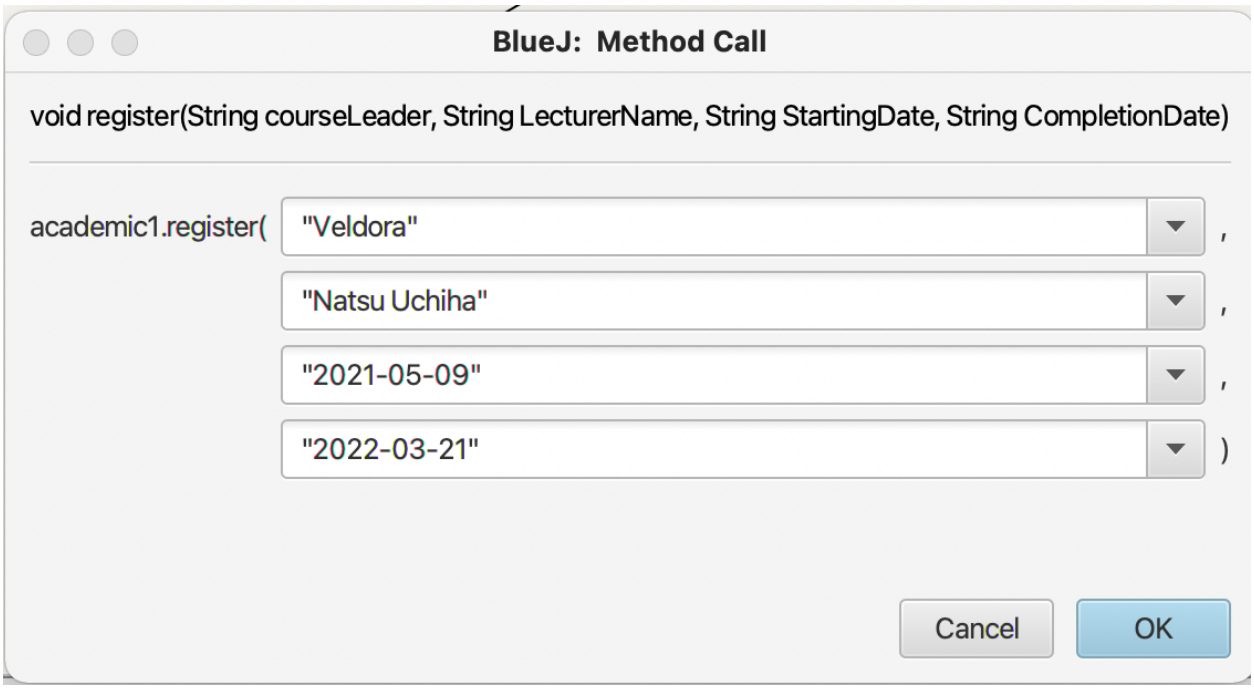

Figure 20: Method call During the error

However, this error was caused because of the error while writing the code in the code I forgot to assign the value for isRegistered parameter inside the void is Registered Method which is the root cause for this error

```
//creating a method register with with four parameters to register the Academic course
public void register(String courseLeader, String lecturerName, String StartingDate, String CompletionDate){
    if(isRegistered== true){
            System.out.println("Corser Leader "+super.getCourseLeader()+ "has started at "+StartingDate +"and completed at" +CompletionDate);
            }
    else{
        super.setCourseLeader(courseLeader);
        this.lecturerName = lecturerName;
        this.StartingDate = StartingDate;
        this.completionDate =CompletionDate;

        System.out.println("New Academic Course is Registered");
        }
}
```

Figure 21: Cause of Logical error

This error can be corrected by assigning the value for isRegistered parameter inside void registered method for condition if is registered is false, which is shown in the below image

```
//creating a method register with with four parameters to register the Academic course
public void register(String courseLeader, String lecturerName, String StartingDate, String CompletionDate){
    if(isRegistered== true){
            System.out.println("Corser Leader "+super.getCourseLeader()+ "has started at "+StartingDate +"and completed at" +CompletionDate);
            }
    else{
        super.setCourseLeader(courseLeader);
        this.lecturerName = lecturerName;
        this.StartingDate = StartingDate;
        this.completionDate =CompletionDate;
        this.isRegistered = true;
        System.out.println("New Academic Course is Registered");
        }
}
```

Figure 22: Correction of Logical error in code

Now we get the following result after correcting the error and using same method call as before in the code:

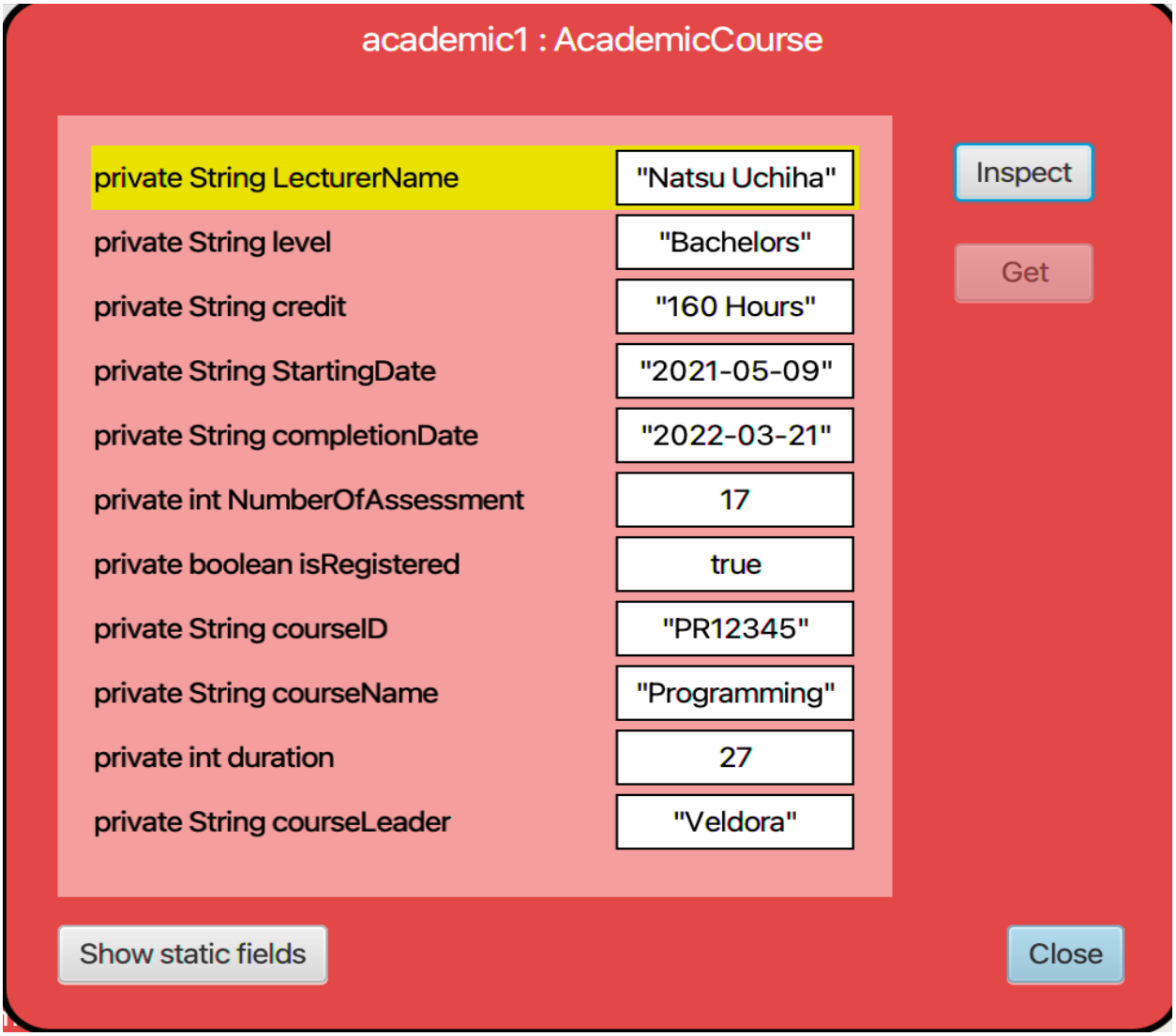

Figure 23: Correction of Logical error

3. Run-Time Error

Runtime error occurs during the execution of program after compiling. The main cause of this type of error is that the code written by the user asks the computer to perform certain task which is impossible for the system to do which might even cause the program to crash.

The example of this error is shown below;

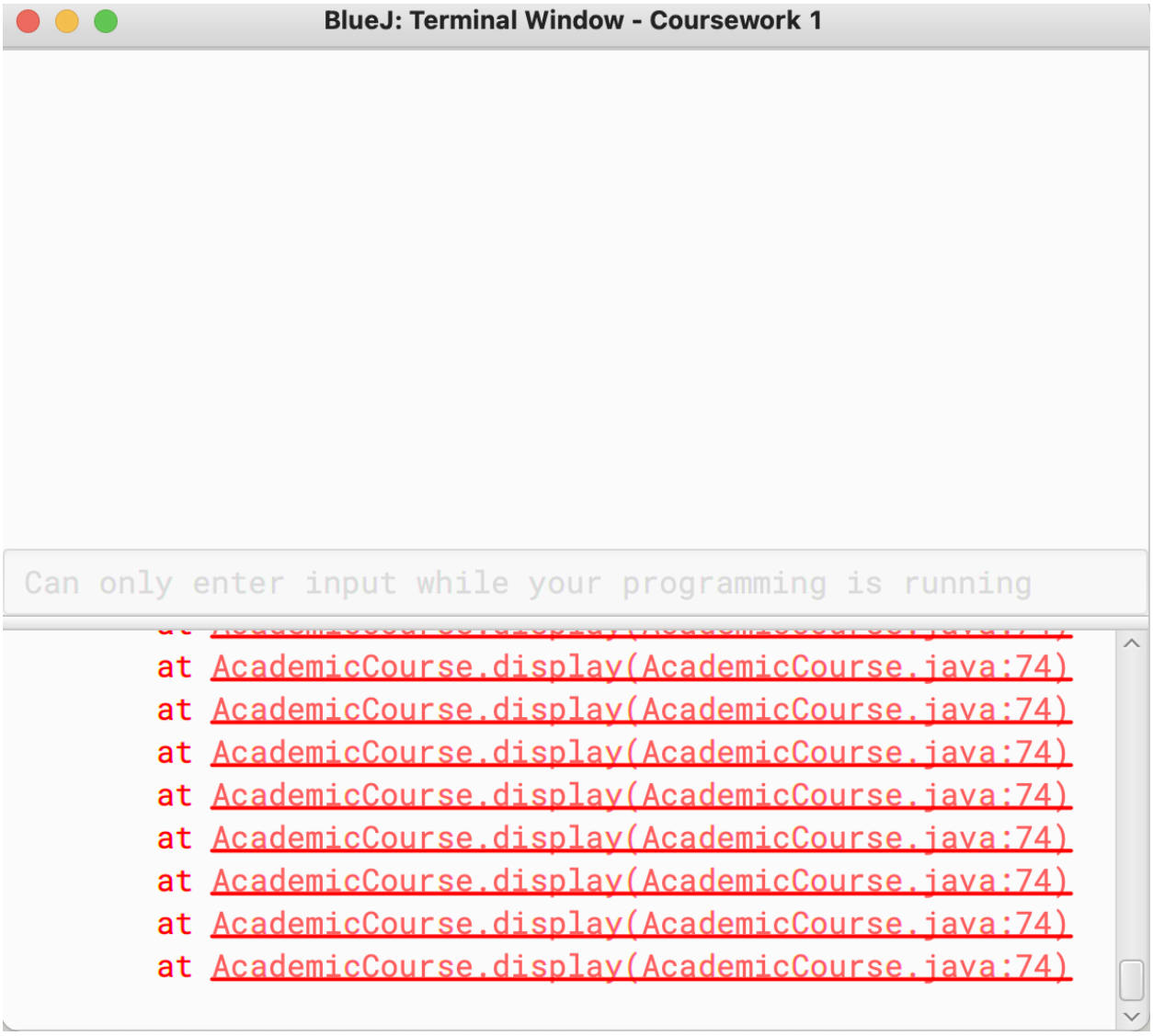

Figure 24: Runtime error example

it happened due to the presence or due to the call of display method of academic course inside the display method of academic course. The cause is shown below:

```
//to display the required output or result
public void display(){
    display();
    super.display();
    if(isRegistered==true){
        System.out.println("lecturerName "+this.lecturerName);
        System.out.println("Level : "+this.level);
        System.out.println("Credit : "+this.credit);
        System.out.println("StartingDate: "+this.StartingDate);
        System.out.println("CompletionDate: "+this.completionDate);
    }
}
```

Figure 25: Cause of error

It can be solved by simply removing the call of display method inside display method if Academic course

```
//to display the required output or result
public void display(){
    super.display();
    if(isRegistered==true){
        System.out.println("lecturerName "+this.lecturerName);
        System.out.println("Level : "+this.level);
        System.out.println("Credit : "+this.credit);
        System.out.println("StartingDate: "+this.StartingDate);
        System.out.println("CompletionDate: "+this.completionDate);
    }
}
```

Figure 26: Solution of the error

Conclusion

This concludes the report for the program I wrote in blue j with the help for java development kit. The program is running smoothly as of now. I have not encountered a huge bug in the code as of now. These programs provide the information about the courses and also helps to register the new one. After doing this work I have the clear idea of how is my knowledge regarding to java programming language. I also got to know my own flaws while doing this project. This project helped me to get the better understanding of java and how it is used in real life scenario.

During my time writing code I encountered various problems but I was able to tackle them with the help of my teacher and the lecture slides that are present in the google class room. I also tried various ways to solve my problem my re writing the code multiple times. And, Now the programming is fully operational it can be used in real life to register and maintain information about the different course, their leaders, lectures, credit hour, start and completion date, etc.

Appendix

1. **Course Class**

```
//creating a class name course
public class Course {
    //declaring instance variables for Course class
     private String courseID;
     private String courseName;
     private int duration;
     private String courseLeader;
    //creating and defining paramaterized constructor
     public Course(String courseID, String courseName, int duration) {
        this.courseID = courseID;
        this.courseName = courseName;
        this.duration = duration;
        this.courseLeader = " ";
```

```
 }
//getter or accesssor method for courseID
 public String getCourseID() {
    return courseID;
 }
//getter or acccessor method for courseName
 public String getCourseName() {
    return courseName;
 }
//getter ir accessor method for duration
 public int getDuration() {
    return duration;
 }
//getter or accessor method for courseLeader
 public String getCourseLeader() {
    return courseLeader;
 }
//to set courseLeader/ Setter method for courseLeader
 public void setCourseLeader(String courseLeader) {
    this.courseLeader = courseLeader;
 }
//To display the desired result or output
 public void display() {
    System.out.println("The courseID is" + courseID);
    System.out.println("The courseName is" + courseName);
    System.out.println("The duration is" + duration);
    if (courseLeader != "") {
        System.out.println("The courseLeader is" + courseLeader);
    }

 }
```

```
}
```

2. AcademicCourse Class

```
//creating a sub class AcademicCourse of the main class course
public class AcademicCourse extends Course{
   private String lecturerName;//declearing all the variables of AcademicCourse
   private String level;
   private String credit;
   private String StartingDate;
   private String completionDate;
   private int NumberOfAssessment;
   private boolean isRegistered;
   //creating parameterized constructor names as AcademicCourse and instance
variables are initialized
   public AcademicCourse(String courseID,String courseName,int duration,
String level, String credit,
   int NumberOfAssessment){
         super(courseID,courseName,duration);
         this.lecturerName="";
         this.level = level;
         this.credit = credit;
         this.StartingDate = "";
         this.completionDate= "";
         this.isRegistered = false;
         this.NumberOfAssessment=NumberOfAssessment;
   }
   //creating a getter or accessor method for lecturerName
   public String getLecturerName(){
            return lecturerName;
   }
```

```
//creating a getter or accessor method for level
public String getLevel(){
        return level;
}
//creating a getter or accessor method for credit
public String getCredit(){
        return credit;
}
//creating a getter or accessor method for startingDate
public String getStartingDate(){
        return StartingDate;
}
//creating a getter or accessor method for completionDate
public String  getCompletionDate(){
        return completionDate;
}
//creating a getter or accessor method for isRegistered
public boolean getIsRegistered(){
        return isRegistered;
}
//creating a getter or accessor method for NumberOfAssessment
public int getNumberOfAssessment(){
        return NumberOfAssessment;
}
//setter or mutator method for lecturerName
public void  setLecturerName (String lecturerName){
        this.lecturerName = lecturerName;
}
//setter or mutator method for NumberOfAssessment
public void  setNumberOfAssessment( int NumberOfAssessment){
        this.NumberOfAssessment = NumberOfAssessment;
```

```
    }
    //creating a method register with with four parameters to register the Academic
course
    public void register(String courseLeader, String lecturerName, String
StartingDate, String CompletionDate){
        if(isRegistered== true){
                System.out.println("Corser Leader "+super.getCourseLeader()+ "has
started at "+StartingDate +"and completed at" +CompletionDate);
                }
        else{
            super.setCourseLeader(courseLeader);
            this.lecturerName = lecturerName;
            this.StartingDate = StartingDate;
            this.completionDate =CompletionDate;
            this.isRegistered = true;
            System.out.println("New Academic Course is Registered");
            }
    }
    //to display the required output or result
    public void display(){
        super.display();
        if(isRegistered==true){
            System.out.println("lecturerName "+this.lecturerName);
            System.out.println("Level : "+this.level);
            System.out.println("Credit : "+this.credit);
            System.out.println("StartingDate: "+this.StartingDate);
            System.out.println("CompletionDate: "+this.completionDate);
        }
    }
}
```

3. **NonAcademicCourse Class**

```
//creating a subclass NonAcademicCourse of main class Course
public class NonAcademicCourse extends Course{
    private String instructorName;//declaring all the variables
    private String startDate;
    private String completionDate;
    private String examDate;
    private String prerequisite;
    private boolean isRegistered;
    private boolean isRemoved;
    /*creating a parameterized constructor NonAcademicCourse and Initializing
      the variables */
    public NonAcademicCourse(String courseID, String courseName, String
prerequisite, int duration){
        super(courseID, courseName, duration);
        this.instructorName = instructorName;
        this.startDate ="";
        this.completionDate="";
        this.examDate = "";
        this.prerequisite = prerequisite;
        this.isRegistered = false;
        this.isRemoved = false;
    }
    //creating a getter or accessor method for instructorName
     public String getInstructorName(){
        return instructorName;
    }
    //creating a getter or accessor method for instructorName
    public String getPrerequisite(){
        return prerequisite;
    }
```

```
    //creating a getter or accessor method for startDate
    public String getStartDate(){
        return startDate;
    }
    //creating a getter or accessor method for completionDate
    public String getCompletionDate(){
        return completionDate;
    }
    //creating a getter or accessor method for examDate
    public String getExamDate(){
        return examDate;
    }
    //creating a getter or accessor method for isRegistered
    public boolean getIsRegistered(){
        return isRegistered;
    }
    //creating a getter or accessor method for isRemoved
    public boolean getIsRemoved(){
        return isRemoved;
    }
    /*using a setter methof for instructorName that accepts instructorName as
      the parameter*/
    public void setInstructorName(String instructorName){
        this.instructorName = instructorName;
        if(isRegistered ==false){
            this.instructorName = instructorName;
            System.out.println("New instructor name is obtain");//disply the msg
        }
        else{
            System.out.println("The instructor is already registered and cannot be
changed");//display the msg
```

```
        }
    }
    /* creating a method register with five parameters to register the
       Non academic course if it is not already registered */
    public void register(String courseLeader, String instructorName, String
startDate, String completionDate, String examDate){
        if(isRegistered == false){
        super.setCourseLeader(courseLeader);
        setInstructorName(instructorName);
        this.startDate=startDate;
        this.completionDate=completionDate;
        this.examDate=examDate;
        this.isRegistered = true;
        }
        else{
            System.out.println("Non Academic Course is already registered");
        }
    }
    //creating a method remove to remove the registered course if any condition
arises
    public void remove(){
             if(isRemoved==true){
            System.out.println("Non-Academic Course is removed");
        }
         else{
            super.setCourseLeader("");
            this.instructorName="";
            this.startDate="";
            this.completionDate="";
            this.examDate="";
            this.isRegistered=false;
```

```
            this.isRemoved=true;
        }
    }
    //to display the desired result or output
    public void display(){
        super.display();
        if(isRegistered==true){
            System.out.println("Instructor Name "+this.instructorName);
            System.out.println("Start Date "+this.startDate);
            System.out.println("Completion Date "+this.completionDate);
            System.out.println("Exam Date "+this.examDate);
        }
    }
}
```