UECS2344 Software Design: Practical 6

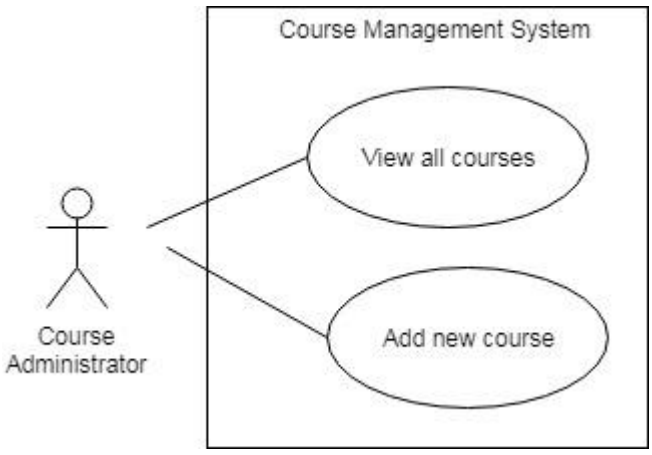
A course management system is to be developed for a course administrator to manage courses. The system is to be developed in 3 iterations.

ITERATION 1

The Analysis Class Diagram, Use Case Diagram, and Prototype Screen, Use Case Description, System Sequence Diagram, Sequence Diagram for each use case and Design Class Diagram (overview) for the first iteration of the system development are given below.

Analysis Class Diagram

Use Case Diagram



Prototype Screens

Use Case: Add new course

Enter course title: xxxxxxxxxxxx
Enter course code: xxxxxx
New course added

Course title: xxxxxxxxxxxxxxx
Course code: xxxxx
Add Course
New course added

Use Case: View all courses

Code: xxxxx Title: xxxxxxxxx
Code: xxxxx Title: xxxxxxxxx
. . .

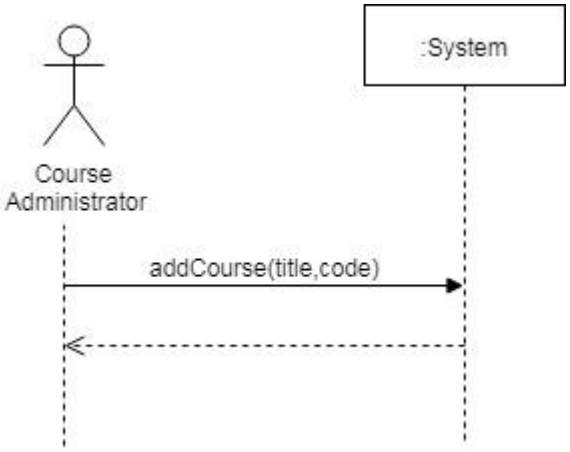
Use Case Descriptions

Use Case Name	Add new course
Actor	Course Administrator
Flow of Events: 1. Course Administrator enters title and code of a new course. 2. System creates and adds a new course record.	

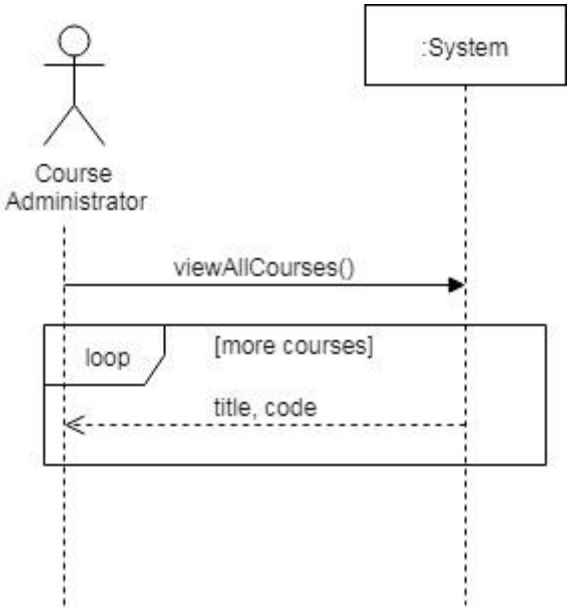
Use Case Name	View all courses
Actor	Course Administrator
Flow of Events: 1. System retrieves all course records and displays title and code of each course.	

System Sequence Diagrams

Use Case: Add new course



Use Case: View all courses



Design

The classes / interface designed for Iteration 1 are as follows:

- Course class: represents a Course (*Entity class*)
- IDataStore interface: contains the methods for managing list of objects
- DataLists class: implements the IDataStore interface and manages a list of Courses
- Controller class: coordinates use case functionality (*Control class*)
- ConsoleUI class: handles all input and output (*Boundary class*)
- CoursesApp class: contains the main() method

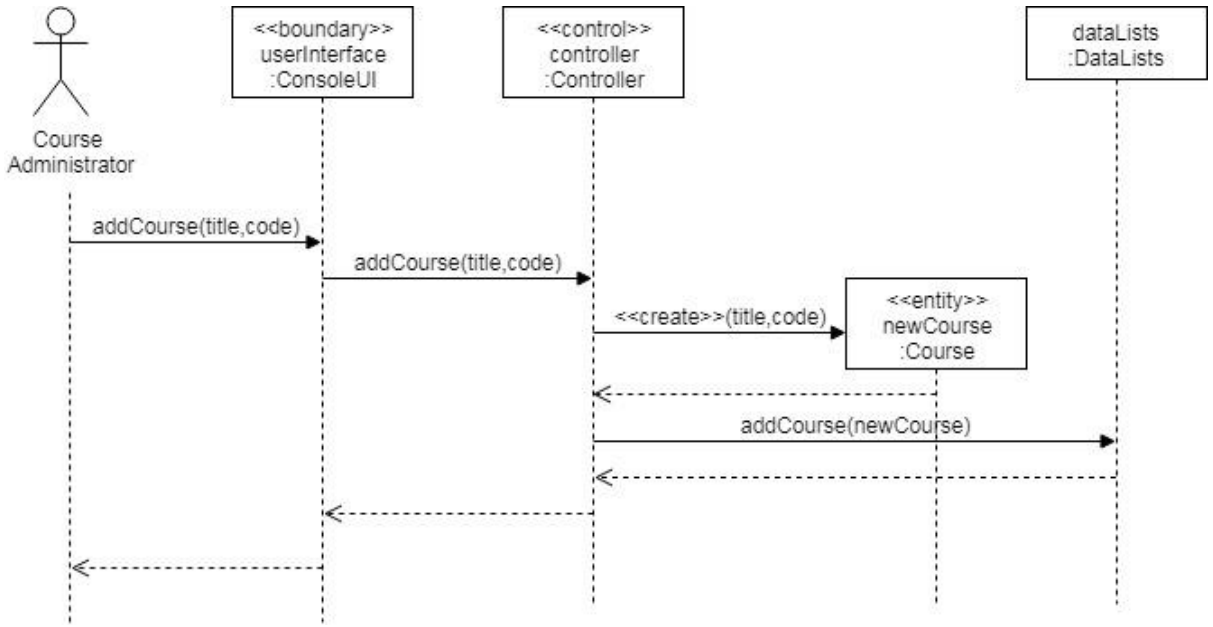
The classes / interface are organised as follows:

- courses.app package
    - ConsoleUI
    - CoursesApp
- courses.domain package
    - Course
    - IDataStore
    - DataLists
    - Controller

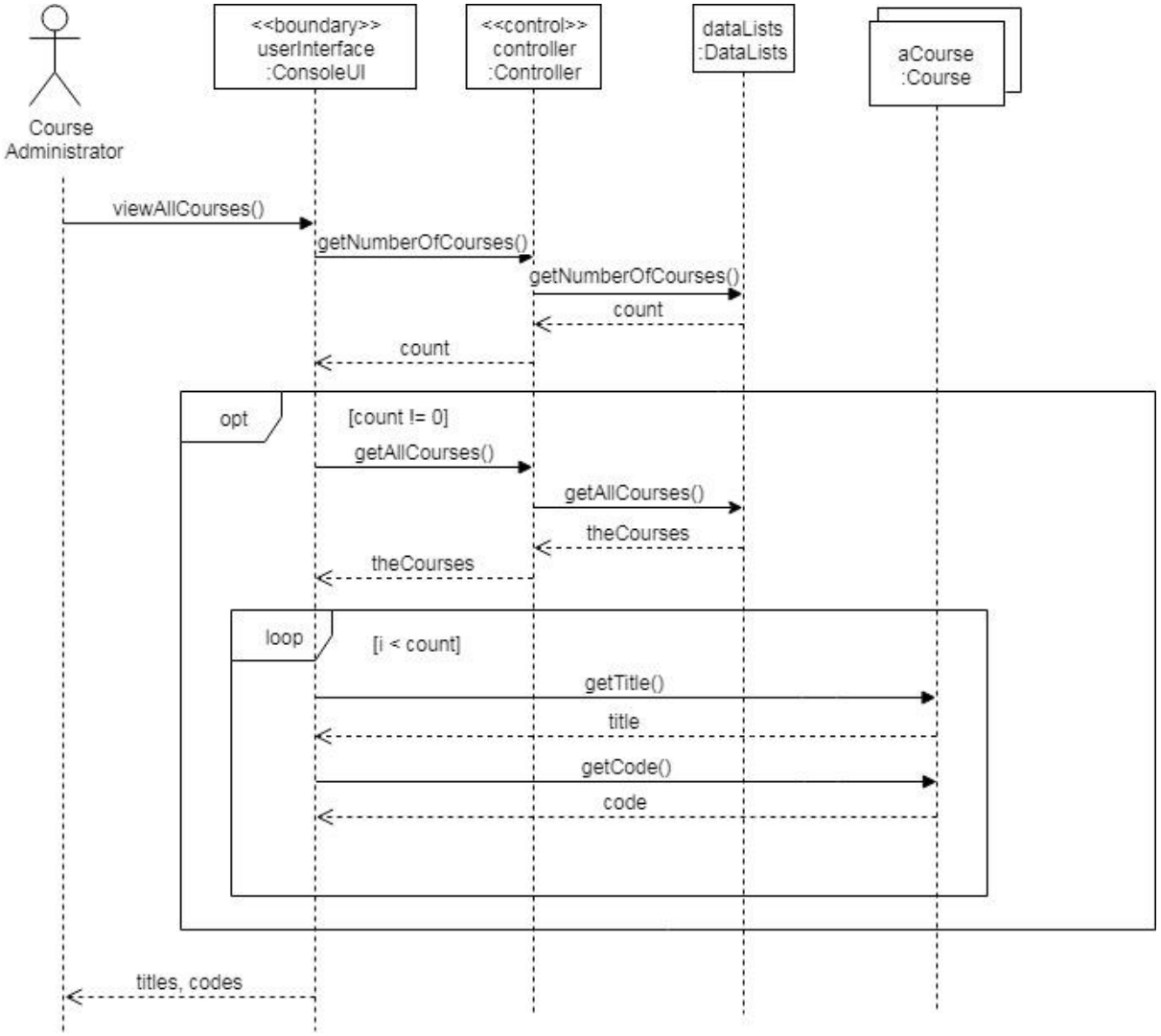
Note: The Scanner and ArrayList classes and List interface are used in implementation (code) but not mentioned in the design.

Sequence Diagrams

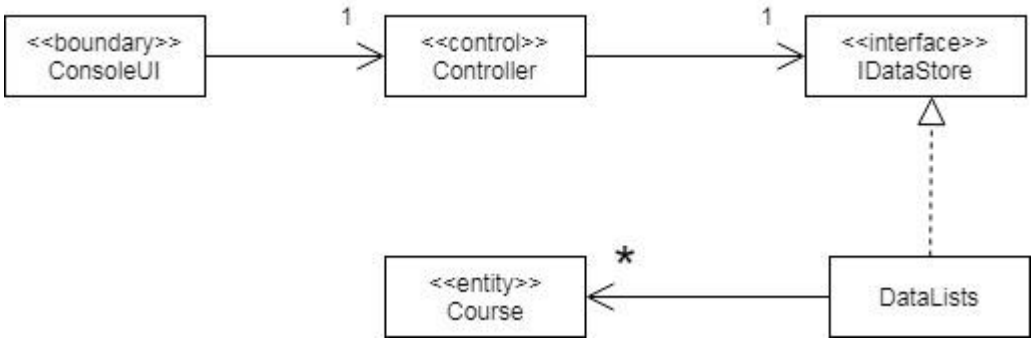
Use Case: Add new course



Use Case: View all courses



Design Class Diagram (overview)



1. Based on the information given above:

   (a) Draw a Package Diagram.

   (b) Draw the Design Class Diagram showing the attributes and operations.

(c) The code for the ConsoleUI and CoursesApp classes is given below. Write the remaining code for the system.

```
package courses.app;

import java.util.List;
import java.util.Scanner;

import courses.domain.*;

public class ConsoleUI {

	private Scanner scanner;

	private Controller controller;

	public ConsoleUI(Controller controller) {
		scanner = new Scanner(System.in);
		this.controller = controller;
	}

	public void start() {

		int choice;

		do {
			System.out.println("Do you want to:");
			System.out.println("1. View all courses");
			System.out.println("2. Add new course");
			System.out.println("3. Exit");

			System.out.print("Enter your choice (1-3): ");
			choice = scanner.nextInt();
			// Clear ENTER key after integer input
			String skip = scanner.next();

			while (choice < 1 || choice > 3) {
				System.out.println("Invalid choice.");
				System.out.print("Enter your choice (1-3): ");
				choice = scanner.nextInt();
				// Clear ENTER key after integer input
				skip = scanner.next();
			}

			switch(choice) {
			case 1: viewAllCourses(); break;
			case 2: addCourse(); break;
			case 3: break;
			}
			System.out.println();
		} while (choice != 3);
```

```
	}

	public void viewAllCourses() {

		int count = controller.getNumberOfCourses();

		if (count == 0)
			System.out.println("No courses to display");
		else {

			List<Course> theCourses = controller.getAllCourses();
			Course aCourse;
			for (int i=0; i<count; i++) {
				aCourse = theCourses.get(i);
				System.out.println("Code: " + aCourse.getCode()
						+ "\tTitle: " + aCourse.getTitle());
			}
		}
	}

	public void addCourse() {
		System.out.print("Enter course title: ");
		String title = scanner.nextLine();

		System.out.print("Enter course code: ");
		String code = scanner.nextLine();

		controller.addCourse(title, code);
		System.out.println("New course created");
		System.out.println();
	}
}
```

```
package courses.app;

import courses.domain.*;

public class CoursesApp {

	public static void main(String[] args) {

		IDataStore dataLists = new DataLists();

		Controller controller = new Controller(dataLists);

		ConsoleUI userInterface = new ConsoleUI(controller);

		userInterface.start();
    }
}
```

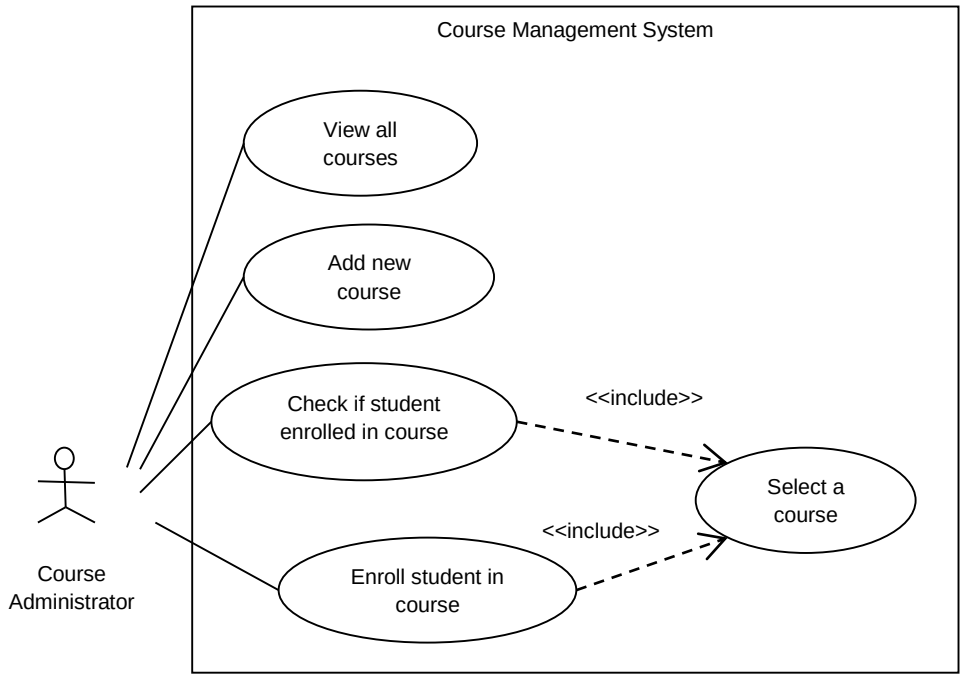
ITERATION 2

The modified Analysis Class Diagram, modified Use Case Diagram, and Prototype Screen, Use Case Description, System Sequence Diagram, and Sequence Diagram for the additional use cases for the second iteration are given below.

Modified Analysis Class Diagram

A course may have many students.
A student takes only 1 course.

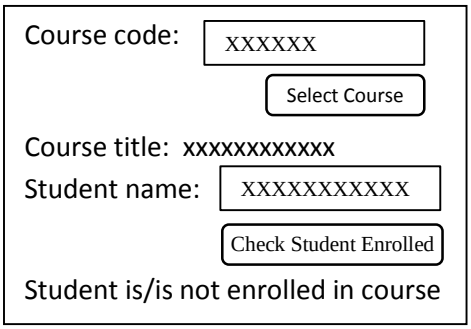
Modified Use Case Diagram



Additional Prototype Screens

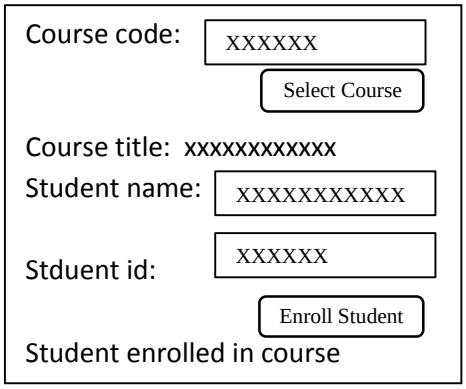
Use Case: Check if student enrolled in course

Enter course code: xxxxxxx
Course title: xxxxxxxxxx
Enter student name: xxxxxxxxxxxx
Student is/is not enrolled in course



Use Case: Enroll student in course

Enter course code: xxxxxxx
Course title: xxxxxxxxxxx
Enter student name: xxxxxxxxx
Enter student id: xxxx
Student enrolled in course



Additional Use Case Descriptions

Use Case Name	Select a course
Actor	Course Administrator
Flow of Events: 1. Course Administrator enters code of the course to select. 2. System searches for the course record and displays title of the course.	
Alternative Flow of Events: 2.1 Course is not found. 2.1.1 System displays “course not found” message. 2.1.2 Use case terminates.	

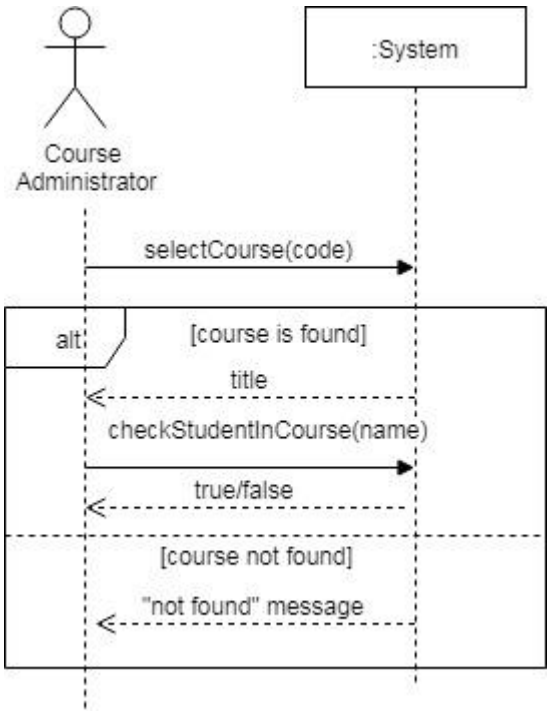
Use Case Name	Check if student enrolled in course
Actor	Course Administrator
Flow of Events: 1. Perform use case Select a course. 2. Course Administrator enters name of student to check. 3. System searches the name in the list of students enrolled for the course. 4. System displays message whether student is enrolled or not.	
Alternative Flow of Events: 1.1 Course is not found. 1.1.1 Use case terminates.	

Use Case Name	Enroll student in course
Actor	Course Administrator

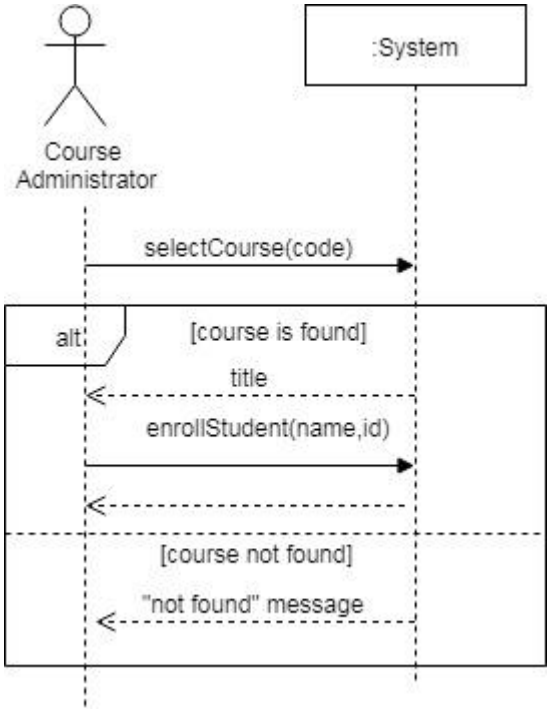
Flow of Events: 1. Perform use case Select a course. 2. Course Administrator enters name and id of student to enroll. 3. System adds the student to the list of students enrolled for the course.
Alternative Flow of Events: 1.1 Course is not found. 1.1.1 Use case terminates.

System Sequence Diagrams

Use Case: Check if student enrolled in course

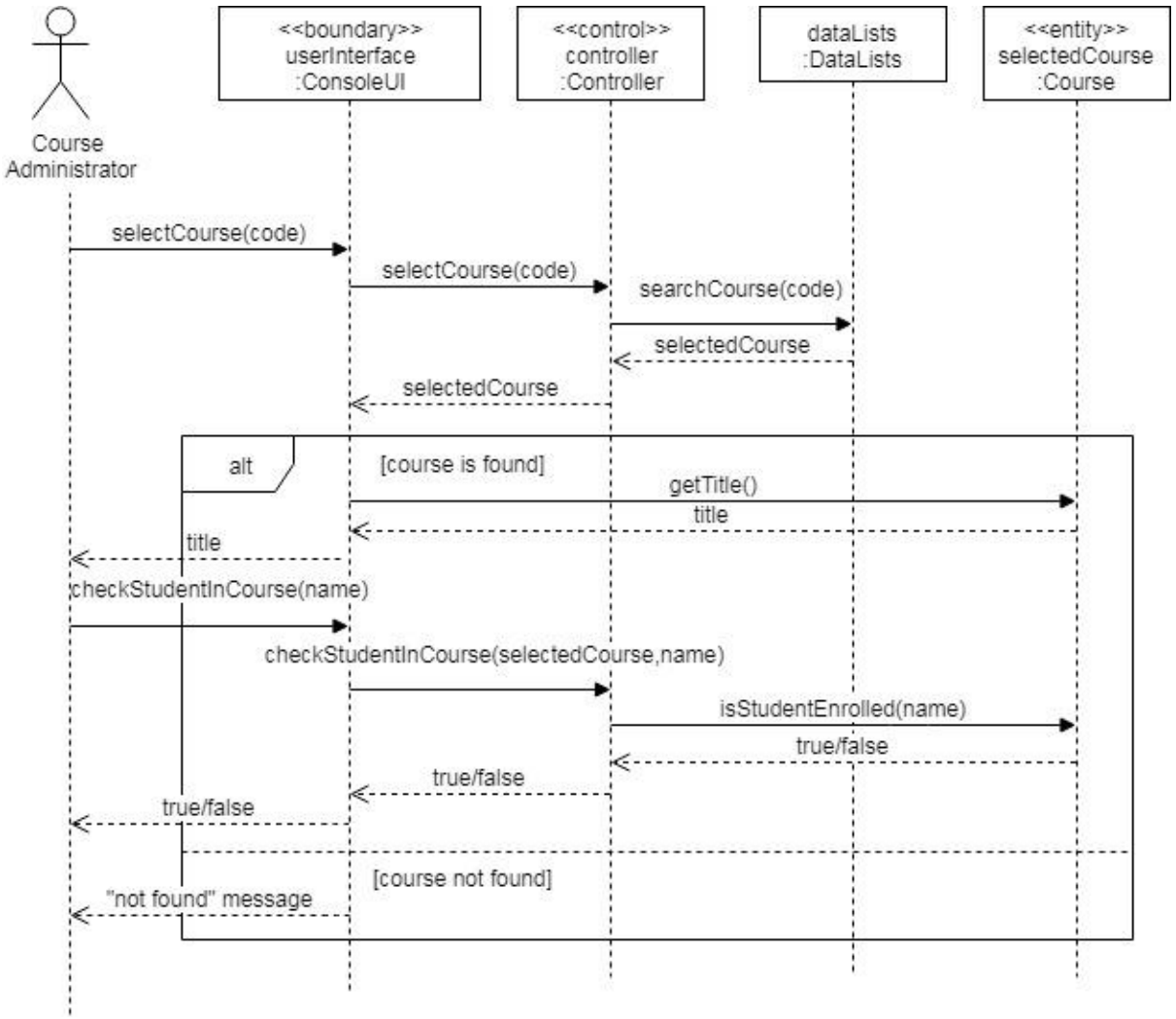


Use Case: Enroll student in course

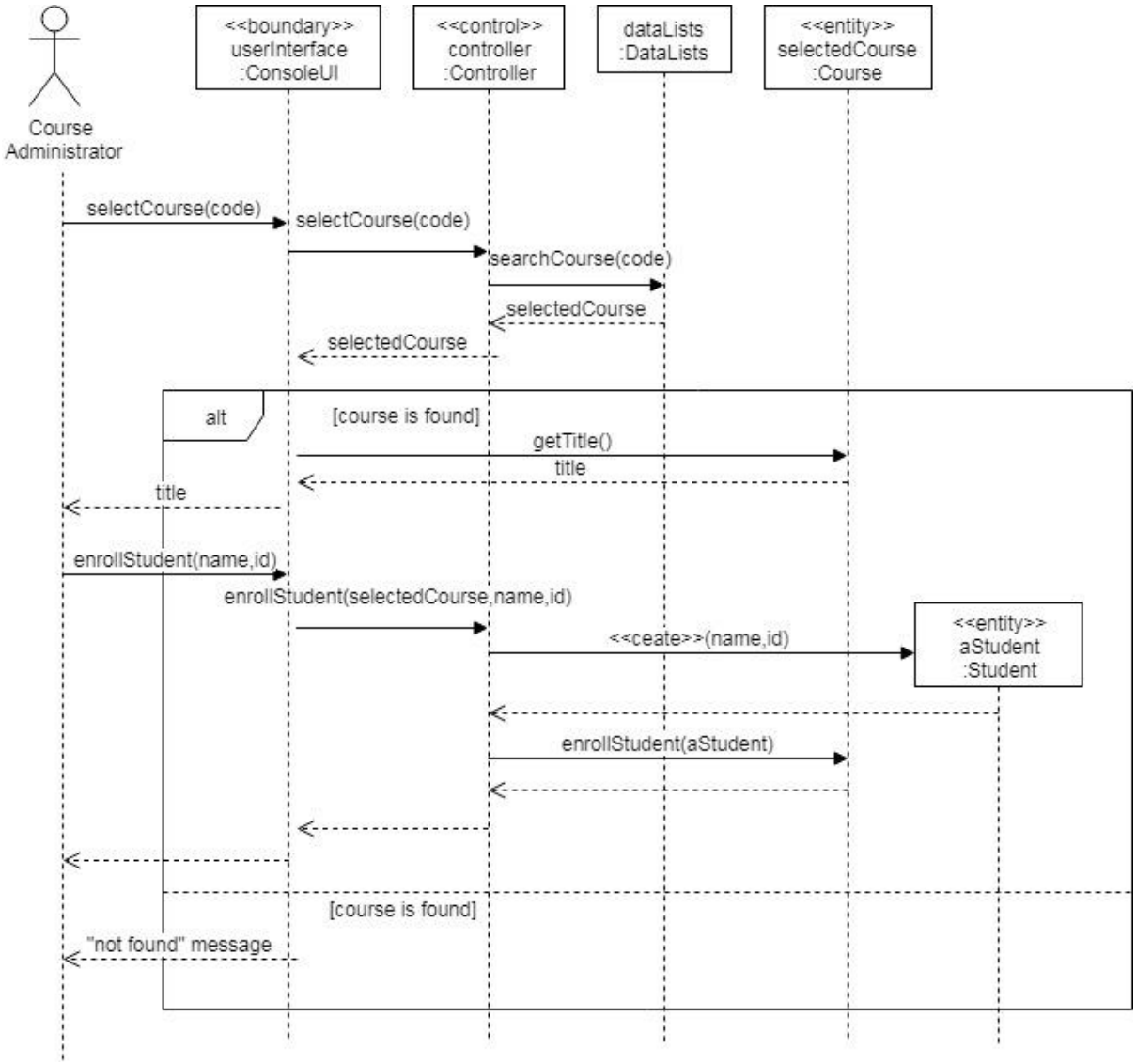


Sequence Diagrams

Use Case: Check if student enrolled in course


Course Administrator
<<boundary>> userInterface :ConsoleUI
<<control>> controller :Controller
dataLists :DataLists
<<entity>> selectedCourse :Course
selectCourse(code)
selectCourse(code)
searchCourse(code)
selectedCourse
selectedCourse
alt
[course is found]
getTitle()
title
title
checkStudentInCourse(name)
checkStudentInCourse(selectedCourse,name)
isStudentEnrolled(name)
true/false
true/false
true/false
[course not found]
"not found" message

Use Case: Enroll student in course



2. Based on the information given above:

   (a) Draw the modified Design Class Diagram showing the attributes and operations.

   (b) Write the modified code for the system.