Assignment 4 Requirement

Late homework assignments will not be accepted, unless you have a valid written excuse (medical, etc.). You must do this assignment alone. No team work or "talking with your friends" will be accepted. No copying from the Internet. Cheating means zero.

Create a new Eclipse workspace named "**`Assignment4_1234567890`**" on the desktop of your computer (replace `1234567890` with your student ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**`Question1`**", "**`Question2`**", etc. Answer all the questions below. At the end of the assignment, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**`Assignment4_1234567890.zip`**" (replace `1234567890` with your student ID number). Upload the ZIP file on iSpace.

Here are a few extra instructions:

- Do not forget to write tests for *all* the code of *all* the classes.
- Give meaningful names to your variables so we can easily know what each variable is used for in your program.
- Put comments in your code (in English!) to explain WHAT your code is doing and also to explain HOW your program is doing it.
- Make sure all your code is properly indented (formatted). Your code should be beautiful to read.

Failure to follow these instructions will result in you losing points.

Question 1

Create a course class with the following UML specification:

```
+-----------------------------------------------------------+
|                          Course                           |
+-----------------------------------------------------------+
| - code: String                                            |
| - title: String                                           |
| - preRequisite: Course                                    |
+-----------------------------------------------------------+
| + Course(String code, String title, Course preRequisite)|
| + getCode(): String                                       |
| + getTitle(): String                                      |
| + getPreRequisite(): Course                               |
| + isRequired(): boolean                                   |
| + testCourse(): void                                      |
+-----------------------------------------------------------+
```

where the **`code`** is an instance variable storing the code of the course, the **`title`** is an instance variable storing the title of the course, the **`preRequisite`** is an instance variable storing the pre-requisite course of the course and a **`preRequisite`** is another course (in our homework setting, a course always has only one pre-requisite course and the pre-requisite course cannot be null. If the pre-requisite course is null, the constructor should print a message of “pre-requisite course cannot be null!” and return directly). The **`testCourse`** method is static.

The **`isRequired`** is a method that returns a boolean value to indicate whether a course is required or not. One course could either be a required course or an elective course.

Should the **isRequired** method be abstract? Why or why not?

Should the **Course** class be abstract? Why or why not?

What kinds of tests can you write inside the **testCourse** method?

Add the following code to your program to test the **Course** class:

```
public class Start {
      public static void main(String[] args) {
            Course.testCourse();
      }
}
```

Question 2

Add a class **MajorRequired** to your program. **MajorRequired** are courses that is required for every student.

The constructor for the **MajorRequired** class takes the code, string and pre-requisite of the course as arguments. The pre-requisite course of a major required course should also be a major required course (type should be **MajorRequired** and pre-requisite course cannot be null).

MajorRequired class should also contain a static method called **testMajorRequired()** that contains the unit tests of **MajorRequired** class.

What kinds of tests can you write inside the **testMajorRequired** method?

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **MajorRequired** class.

Question 3

Add a class **MajorElective** to your program. **MajorElective** are courses that are elective for every student.

The constructor for the **MajorElective** class takes the code, string and pre-requisite of the course as arguments. The pre-requisite course of a major elective course should be a major required course (type should be **MajorRequired** and pre-requisite course cannot be null).

Your **MajorElective** class should also contain a static method called **testMajorElective()** that contains the unit tests of **MajorElective** class.

What kinds of tests can you write inside the **testMajorElective** method?

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **MajorElective** class.

Question 4

The problem with the **MajorRequired** class above is that the constructor takes a **MajorRequired** object as the third argument and it cannot be null. This means that, to create a major required course, you need to already have another major required course. In turn this means that you cannot create any major required course at all, because you cannot create the first one! To solve this problem, add to your program a `Base` class with the following UML specification:

```
+-----------------------------------+
|               Base                |
+-----------------------------------+
| - code: String                    |
| - title: String                   |
+-----------------------------------+
| + Base(String code, String title) |
| + getCode(): String               |
| + getTitle(): String              |
| + getPreRequisite(): Base         |
| + testBase(): void                |
+-----------------------------------+
```

Base courses do not necessarily have any pre-requisite, so the `getPreRequisite` method will return itself.

Your `Base` class should also contain a static method called `testBase()` that contains the unit tests of `Base` class. Do not forget to change the `main` method of the `Start` class to run the unit tests of the new `Base` class.

Question 5

We now want to modify the code so that a base course can be the pre-requisite of other courses. To do this, add to your program a `Learnable` interface, which can be the pre-requisite of any other courses, with the following UML specification:

```
+----------------------------------+
|          <<interface>>           |
|            Learnable             |
+----------------------------------+
| + getCode(): String              |
| + getTitle(): String             |
| + getPreRequisite(): Learnable   |
+----------------------------------+
```

Modify the `Course` and `Base` classes so that they both implement the `Learnable` interface, and modify the `Course`, `MajorRequired`, and `MajorElective` so that a major required/elective course can now have a pre-requisite who is either a major required course or a base course.

Once you have done this, add tests to:

- `testMajorRequired` method of the `MajorRequired` class: create a `Base` object and use that `Base` object to create a first `MajorRequired` object who has the base course as pre-requisite; then use that first `MajorRequired` object to create a second `MajorRequired` object who has the first major required course as pre-requisite. Test all the methods of the two `MajorRequired` objects.

- `testMajorElective` method of the `MajorElective` class: create a `Base` object and use that `Base` object to create a first `MajorRequired` object who has the base course as pre-requisite; then use that `MajorRequired` object to create a `MajorElective` object who has the major required course as pre-requisite. Test all the methods of the `MajorElective` object.

Question 6

We now want to be able to manipulate many courses together, not just one courses at a time. So add a **`ManyCourses`** class to your program with the following UML diagram:

```
+--------------------------------+
|            ManyCourses         |
+--------------------------------|
| - Courses: ArrayList           |
+--------------------------------+
| + ManyCourses()                |
| + addCourse(Learnable c): void |
| + listCourses(): void          |
| + testManyCourses(): void      |
+--------------------------------+
```

The **`Courses`** instance variable is of type **`ArrayList`** which is a class provided to you by Java that you need to import into your program using: `import java.util.ArrayList;`
You can then define the **`Courses`** instance variable like this: `private ArrayList Courses;`

In the **`ManyCourses`** constructor you need to create a new **`ArrayList`** object and store it in the instance variable **`Courses`**, like this: `this.courses = new ArrayList();`
(if you forget to do this then the instance variable **`Courses`** will point at nothing and you will get an error when you run your program and you try to call a method of the nonexistent arraylist object).

The **`addCourse`** method takes a course as argument and adds it to the arraylist.

The **`listCourses`** method prints on the screen the course code and title, one by one, using a loop.

Here is the code of the **`testManyCourses`** method:

```
public static void testManyCourses() {
    Base b = new Base("DS1001","EntryCourse");
    MajorRequired mr1 = new MajorRequired("DS200X","OOP",b);
    MajorElective me1 = new MajorElective("DS300X","Data Mining",mr1);
    ManyCourses mc = new ManyCourses();
    mc.addCourse(b);
    mc.addCourse(mr1);
    mc.addCourse(me1);
    mc.listCourses();
}
```

Then the **`listCourses`** method should print:

```
DS1001,EntryCourse
DS200X,OOP
DS300X,Data Mining
```

Do not forget to change the **`main`** method of the **`Start`** class to run the unit tests of the new **`ManyCourses`** class.