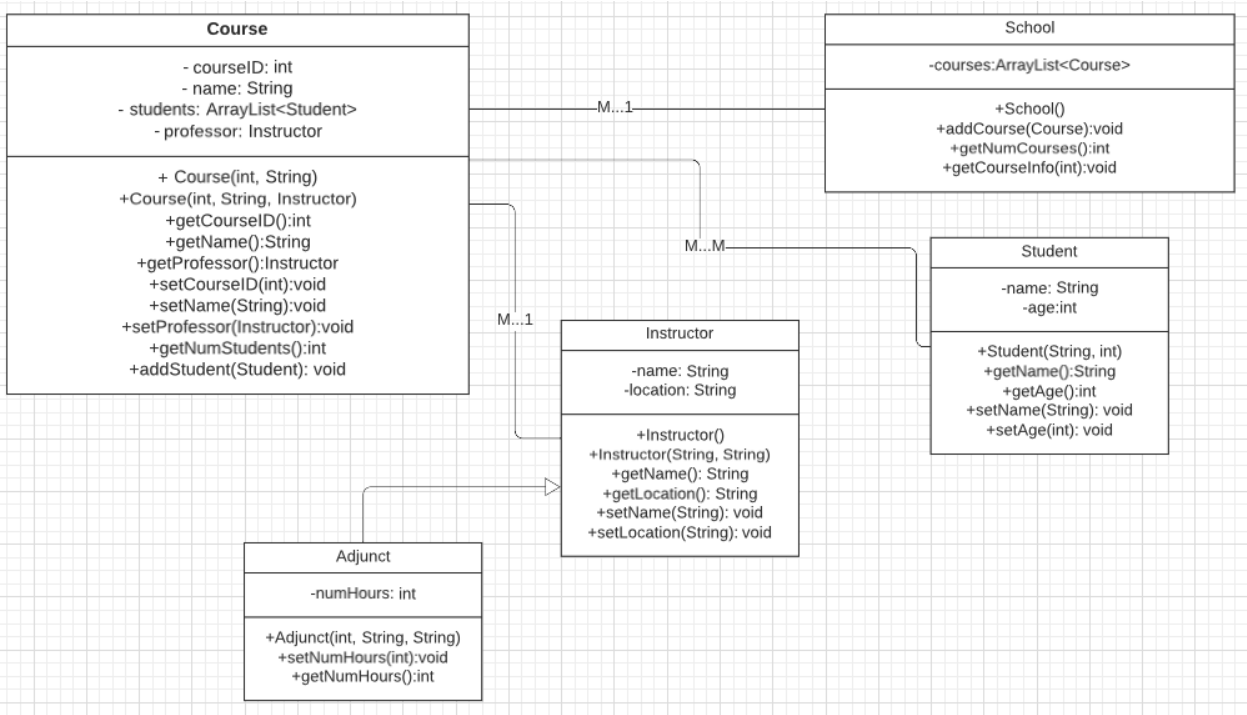
CS234 Computer Science II
Lab 7
Total points: 100

Read the instructions carefully.

In this Lab you will practice how to use inheritance and multiple classes in a Java program.
Moreover, you will practice how to read UML Class diagrams containing inheritance.



Implement the **classes** (i.e., a .java file ***without the main method***) shown in the above UML Class diagram.

No extra instance variables or extra methods are allowed. Read the diagram, it contains all the information for this lab.

Class School

This class contains a single instance variable for a list of Courses (objects for each course).
The **constructor initializes** the list of courses.
The method **addCourse()**, receives as explicit parameter an **object** of class **Course** and adds it to the list of courses.
The method **getNumCourses()**, returns the total number of courses stored in the list of courses.
The method **getCourseInfo()**, receives as explicit parameter a course id. If the course id exists, then it prints out the information of that course. Otherwise, it should send a message to the user saying that the course does not exists (See the example below. Your output should be the **same**).

Class Course

This class contains four instance variables. One for the course id, one for the course name, one for the list of Students (objects for each student), and one for the instructor (the object for the Instructor).
The **first constructor**, initializes the list of students, assigns the course id, and assign the course name.
The **second constructor**, initializes the list of students, assigns the course id, assigns the course name, and assigns the instructor.
The method **getCourseID()**, returns the course id.
The method **getName()**, returns the course name.
The method **getProfessor()**, returns the Instructor
The method **setCourseID()**, receives as explicit parameter a course id and assigns it to the course.
The method **setName()**, receives as explicit parameter the name of the course and assigns it to the course.
The method **setProfessor()**, receives as explicit parameter the object of an instructor and assigns it to the course.
The method **getNumStudents()**, returns the total number of students in the students' list.
The method **addStudent()**, receives as explicit parameter the object of a Student and stores it in the list of students.

Class Instructor

This class contains two instance variables. One for the name of the instructor and another for the location.
The **first constructor**, assigns "STAFF" for the name and "Portales" for the location.
The **second constructor** receives as explicit parameters the name and location and assigns them to the instance variables.
The method **getName()**, returns the name of the instructor
The method **getLocation()**, returns the location of the instructor
The method **setName()**, receives as explicit parameter the name of the instructor and assigns it to the instance variable.
The method **setLocation()**, receives as explicit parameter the location of the instructor and assigns it to the instance variable.

Class Adjunct

This is a subclass of Instructor. It contains one instance variable for the number of hours that this person works.
Its constructor receives the number of hours, the adjunct's name, and the location. Therefore, it needs to initialize the parent class' variables.
The method **setNumHours()**, receives as parameter the number of hours and assigns this number to the corresponding variable.
The method **getNumHours()**, returns the number of hours worked.

Class Student

This class contains two instance variables. One for the name of the student and another for the age of the student.
The **constructor** receives as explicit parameters the name and age and assigns them to the instance variables.
The method **getName()**, returns the name of the student
The method **getAge()**, returns the age of the student

The method **setName()**, receives as explicit parameter the name of the student and assigns it to the instance variable.
The method **setAge()**, receives as explicit parameter the age of the student and assigns it to the instance variable.

I will **use** a separate **tester program** (separate .java file with main() method) to test your class definition.
The main method for **my** tester program is the following:

```
    public static void main(String[] args) {
//        Create a new school
        School enmu = new School();

        System.out.println("Number of Courses: " + enmu.getNumCourses());
//        Create students
        Student s1 = new Student("Emma", 18);
        Student s2 = new Student("John", 21);
        Student s3 = new Student("Julian", 20);
        Student s4 = new Student("Bella", 19);
        Student s5 = new Student("Carlos", 22);
        Student s6 = new Student("Stephen", 18);
//        Create instructors
        Instructor ecv = new Instructor("Eduardo", "Portales");
        Instructor ei = new Instructor("Essa", "Clovis");
        Instructor staff = new Instructor();
        Adjunct big = new Adjunct(4, "Carlos", "Lubock");
//        Create courses
        Course cs123 = new Course(123, "Computer Science I");
        cs123.setProfessor(ei);
        Course cs234 = new Course(234, "Computer Science II");
        cs234.setProfessor(ecv);
        Course csxx = new Course(111, "Directed Studies", staff);
        Course elective = new Course(222, "Big Data", big);
//        Add students to courses
        cs123.addStudent(s6);
        cs123.addStudent(s5);

        cs234.addStudent(s1);
        cs234.addStudent(s6);
        cs234.addStudent(s2);

        csxx.addStudent(s3);
        csxx.addStudent(s4);

        elective.addStudent(s4);
        elective.addStudent(s6);
        elective.addStudent(s2);

//        Add courses to the school
        enmu.addCourse(cs123);
        enmu.addCourse(cs234);
        enmu.addCourse(csxx);
        enmu.addCourse(elective);
        System.out.println("Number of Courses: " + enmu.getNumCourses());

//        This course exists
        enmu.getCourseInfo(234);
//        This course does not exist
        enmu.getCourseInfo(555);
//        This course exists
        enmu.getCourseInfo(222);
    }
```

The output of my tester program is:

```
Number of Courses: 0
Number of Courses: 4
The course 'Computer Science II' has 3 students.
The instructor is Eduardo
The course 555 does not exist
The course 'Big Data' has 3 students.
The instructor is Carlos
```

You can create your own tester program to practicing and test your class definitions.
From my tester program you **must take the names for your methods and classes.** Be careful, I will use this same tester program to test your class definition therefore the **names** for the classes and methods must be **the same** in your files.

Tip: If you are stuck on how to pass one object as parameter of another object's method, think in how you would pass a String (Strings are objects. They have their own methods.) as a parameter. Think about how you would use the methods of that String once it has been passed.

Please manually trace my tester program to **understand** the output.

Submission details:

Upload a **single ZIP** file.
Name your file as follows: **Lab7_Lastname_Firstname.zip**

Your **.zip** file must contain the following:

1. Your **.java** source file for your **class definition** (.java file without the **main** method. No .class file). **Do not send the tester program.**
2. A **SINGLE PDF** with screenshots from your program running. Do not send .jpg files.

For this lab, you **do not need to submit the .txt** file with your instructions*****. **Why?** Because I will use my tester programs to use your classes. Therefore, it is **extremely important** that your **class and method names** are **the same** as they are shown in the class diagram and in my tester program.

***Note:** If you want to use packages, you need to submit the .txt file indicating how to use your package (i.e., how the path should be created)
In each .java file, **write** as a multiline comment at the beginning of the file the following: Your name

The **zip** file must be uploaded to Canvas. I do not accept answers via email or as comments.
I do not accept image files; it must be a PDF file.

Make sure to check the **due date** for this activity on Canvas. Try to submit it before the due date so you can have time to check for improvements. **No late submissions.**

Make sure you are **submitting the correct files**. I will grade the files uploaded to Canvas.

Make sure you **test** your **classes** with a similar **tester program** as the ones I am showing in this lab (i.e., a .java program with a main method where you create objects from your class).

Use the *javac* and *java* commands **before** submitting your solution. Remember that you can compile your classes to check for error but they are only executed using a program.

Make sure to review the **grading rubric.**

Read all the instructions carefully.

If you have questions, contact me before making assumptions about what you need to do for solving this assignment.
This is the basis for your project, it is important to understand these concepts!

Ensure that your code is original and developed by you.