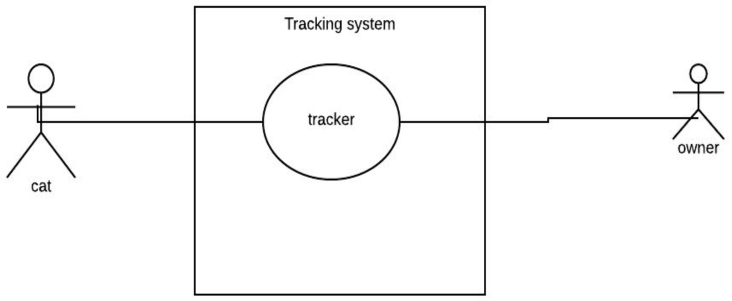
Class Exercise 2:

OO design:

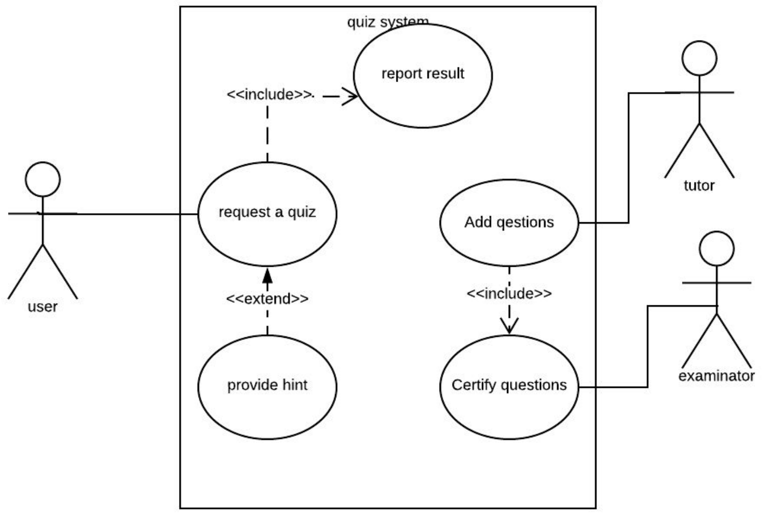
[Use case Diagram]

Many people have cats. Suppose you want to develop software enabling them to keep track the location of their cats; give a use case diagram modeling this software. In this example, just decide by yourself what you want from this software.

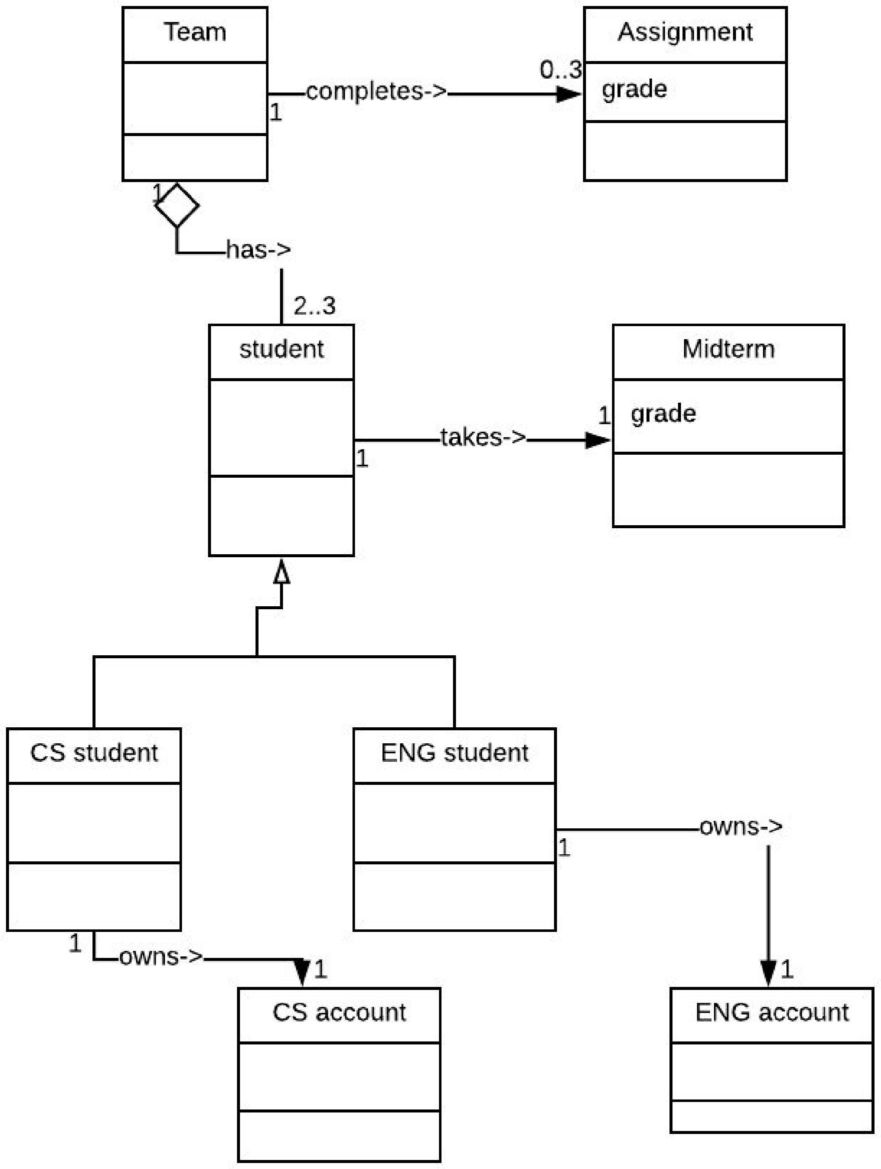
a) What is the main purpose of use case modeling?
   To show the functional requirement.
b) Give few examples of non-functional requirements that could be sensical for this software. How do you position use-case modeling with respect to such requirements?
   For example, platform independent that can run on Mac, windows, Unix, etc…) Such requirement can not be shown on use-case diagram and has to be written in complementary document.



Let's imagine we will develop a browser-based training system to help people prepare for a certification exam such as real estate certificate or java programming certificate. A **user** can request a quiz for the system. The system picks a set of questions from its database, and compose them together to make a quiz. It rates the user's answers, and gives hints if the user requests it. 2 In addition to users, we also have **tutors** who provide questions and hints. And also **examinators** who must certify questions to make sure they are not too trivial, and that they are sensical. Make a use case diagram to model this system. Work out some of your use cases. Since we don't have real stake holders here, you are free to fill in details you think is sensical for this example.



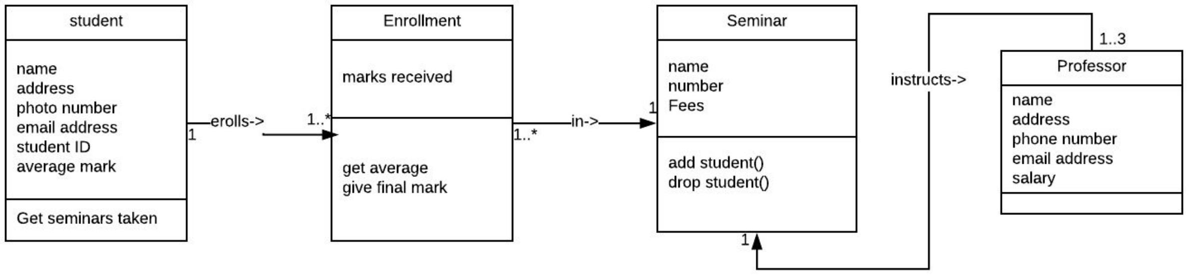
[Class Diagram]
Draw a class diagram (with attributes, where appropriate) that describes entities and relationship relevant to the below scenario. Students **are members of teams**. Each team has 2 or 3 members. Each team completes 0 to 3 **assignments**. Each student takes exactly one midterm test. Computer Science students have a single account on NCS facility, while each engineering student has an **account** on the ENGR facility. Each assignment and midterm is assigned a grade.



Consider the following simplified description of a university where professors teach seminars in which students can enroll.

A professors has a name, address, phone number, email address, and salary. A student has also a name, etc., but no salary (sorry). A student, however, has an average mark (of the final marks of his or her seminars). A seminar has a name and a number. When a student is enrolled in a seminar, the marks for this enrollment are recorded and the current average as well as the final mark (if there is one) can be obtained from the enrollment. From a student one can obtain a list of seminars he or she is enrolled in. Professors teach seminars. Each seminar has at least one and at most three teachers. There are two types of seminar: bachelor and master. From a bachelor seminar students can not withdraw. From a master seminar they can.

Draw a class diagram for this university. Add attributes and methods when necessary. You do not have to include getters and setters for attributes. Visibility modifiers (public, private, etc.) are not required.



[Observer Pattern]
In the following Java GUI/Event Handling code fragment:

```
/* Some GUI control object that lays out GUI components */
JButton button1 = new JButton( "Stop" );
button1.addActionListener( new ResizableBall( /* args */ ) );

public class ResizableBall implements ActionListener
{
/* Lots of other stuff associated with a ResizableBall */
public void actionPerformed( ActionEvent evt )
{
/* Do something with evt */
}
}
```

Which object is the Subject? _____**button1___**________________

Which object is the Observer? ______**ResizableBall__**____________

Questions about software architecture

1. What are the fundamental architectural views proposed in Krutchen’s 4+ 1 model?

2.What is an architectural pattern?

3. What is the fundamental characteristic of a repository architecture?

4. What is advantage and disadvantage of layered architecture?

5.What is the most important advantage of a client-server architecture?

6. What are the two very widely used generic application architectures?

Class relationship:

Generalization
Inheritance
Aggregation
Composition