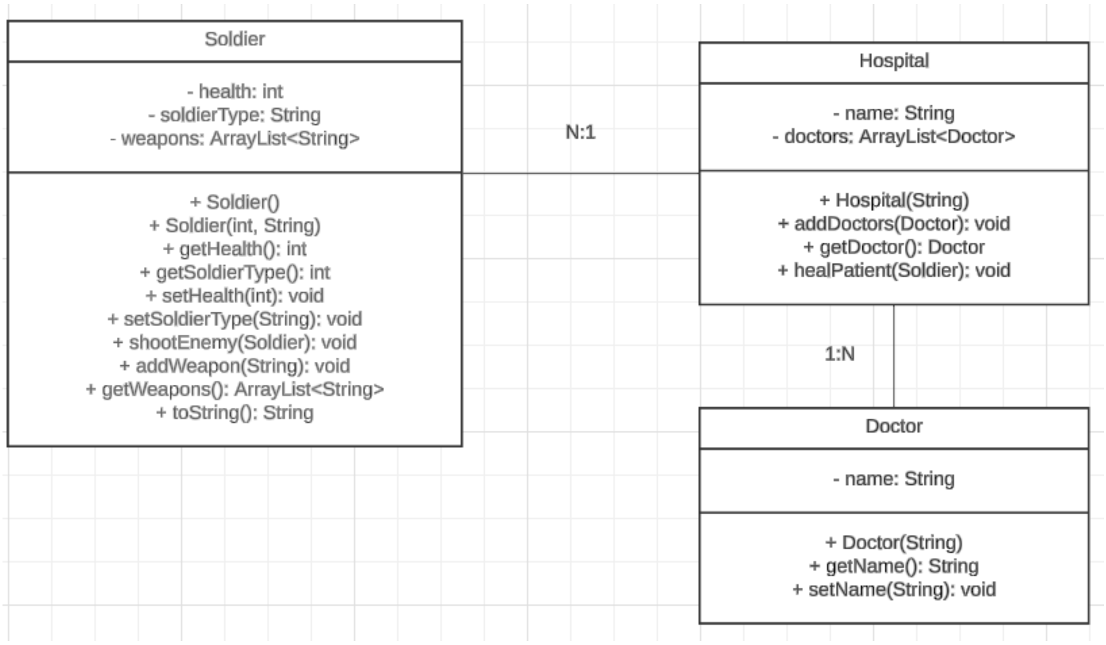
CS234 Computer Science II
Lab 6
Total points: 100

Read the instructions carefully.

In this Lab you will practice how to use more than one class definition in a Java program.
Moreover, you will practice how to read UML Class diagrams.



Tasks:
A)
Write a ***class definition*** for a **Hospital** (i.e., .java file **without the main method**).
You need to **define** a private **instance** variable to store a hospital's **name**.
You need to **define** a private **instance** variable to store a **list** of **doctors**.
No extra instance variables are allowed.

The name for the hospitals and the list of doctors are **initialized** in the **constructor** for that instance.

The **addDoctors** method, **adds** a doctor to the list of doctors.
The **getDoctor** method, **randomly** selects a doctor from the list of doctors and returns the doctor.

The **healPatient** method, receives a patient as a parameter. The method informs that the patient is at the Hospital and gets a doctor to help the patient.
The hospital checks the patient's health. If the health is **0** units of health or less, sadly, the patient is **dead**.
Otherwise, the hospital **increases** the patient health by **1** unit.
This method informs the user that a soldier is at the hospital, the doctor's name who is taking care of the patient, and if the patient is dead or recovering.

No extra methods are needed nor allowed.

B)
Write a ***class definition*** for a **Soldier** (i.e., a .java file **without the main method**).
A given soldier has the definition of private instance variables for **health**, the **type** of soldier, and a **list** of **weapons**.
No extra instance variables are allowed.

There must be **two constructors.** A default constructor and custom constructor.
The **default constructor initializes** the health with a value of **10** units, the soldier type with an **empty** string, and an **empty** list of weapons.
The **custom constructor** receives the health and type and **initializes** an empty list of weapons.

There must be methods to **get** and **set** the soldier's **health** value and the soldier's **type**.
Similarly, there must be a method to **add a weapon** to the list of weapons for each soldier, and a method to **get** that **list** of weapons.

A soldier can shoot enemies. Therefore, you need to create a method for **shooting** at an enemy (which is also a soldier). Every time a soldier shoots at an enemy, the health of that enemy is **reduced** by **1** unit. If the enemy's **health** reaches **0**, the enemy is **dead**.

Finally, a **toString** method is expected to get all the soldier's information. This method is a *built-in method* in Java that you are going to **override** with your own definition to output each soldier's information: type, health, and weapons (see the example below).

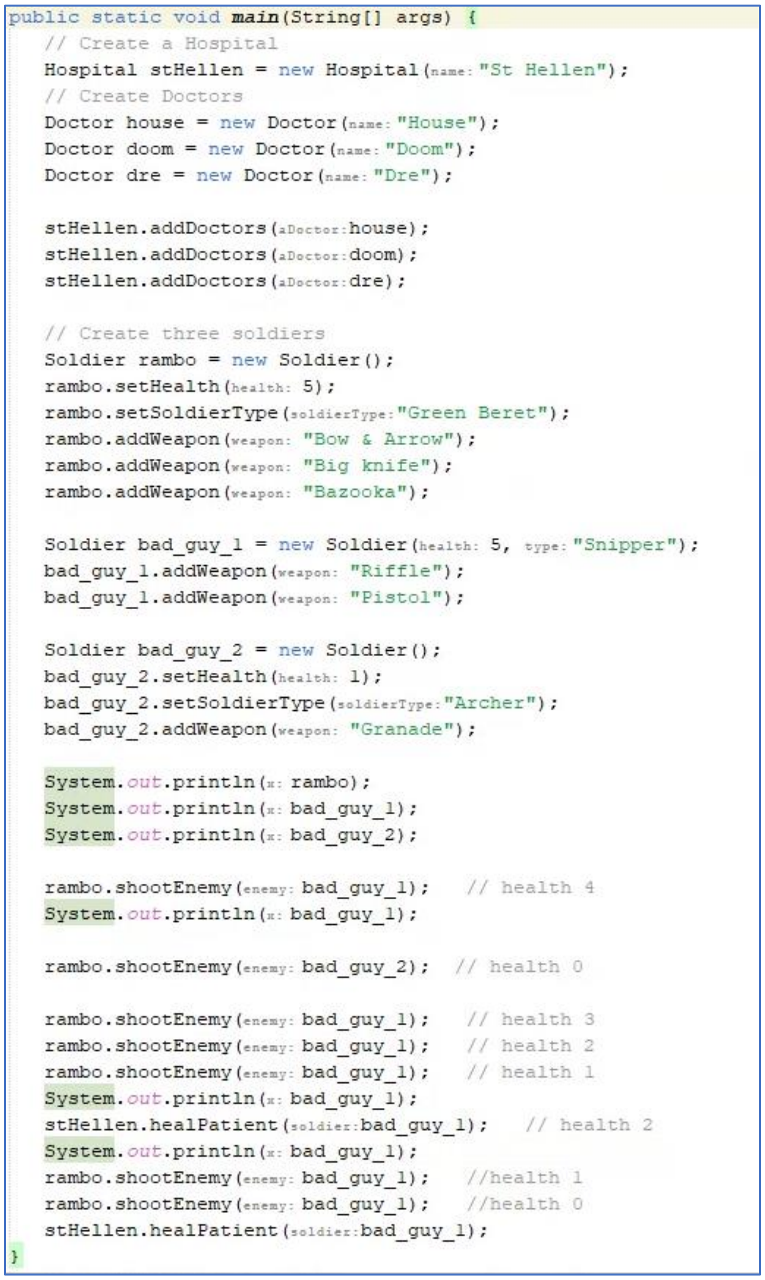
No extra methods are needed nor allowed.

C)
Write a ***class definition*** for a **Doctor** (i.e., a .java file **without the main method**).

For the Doctor class you need to **define** a **private** instance **variable** for the **name** of the doctor.
The **constructor initializes** the name of the doctor with the received parameter.
The **getName** method returns the name of the doctor.
The **setName** method assigns a name to the doctor.

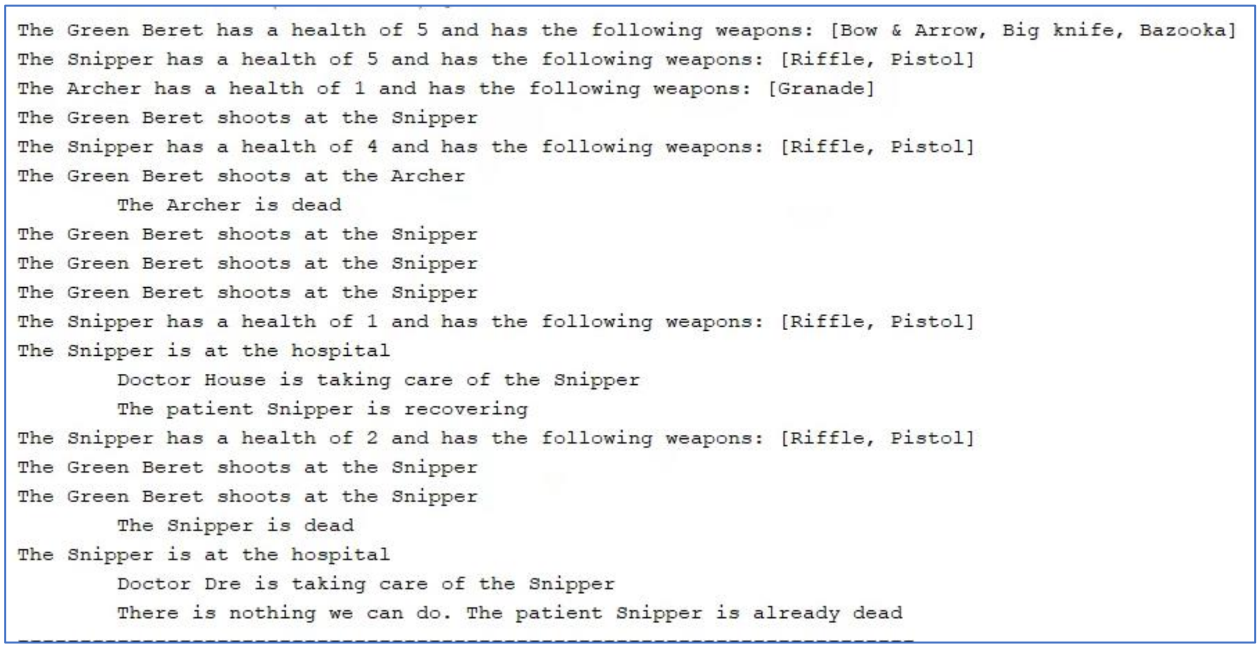
No extra instance variables are allowed.

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



```
public static void main(String[] args) {
    // Create a Hospital
    Hospital stHellen = new Hospital(name: "St Hellen");
    // Create Doctors
    Doctor house = new Doctor(name: "House");
    Doctor doom = new Doctor(name: "Doom");
    Doctor dre = new Doctor(name: "Dre");

    stHellen.addDoctors(aDoctor:house);
    stHellen.addDoctors(aDoctor:doom);
    stHellen.addDoctors(aDoctor:dre);

    // Create three soldiers
    Soldier rambo = new Soldier();
    rambo.setHealth(health: 5);
    rambo.setSoldierType(soldierType:"Green Beret");
    rambo.addWeapon(weapon: "Bow & Arrow");
    rambo.addWeapon(weapon: "Big knife");
    rambo.addWeapon(weapon: "Bazooka");

    Soldier bad_guy_1 = new Soldier(health: 5, type: "Snipper");
    bad_guy_1.addWeapon(weapon: "Riffle");
    bad_guy_1.addWeapon(weapon: "Pistol");

    Soldier bad_guy_2 = new Soldier();
    bad_guy_2.setHealth(health: 1);
    bad_guy_2.setSoldierType(soldierType:"Archer");
    bad_guy_2.addWeapon(weapon: "Granade");

    System.out.println(x: rambo);
    System.out.println(x: bad_guy_1);
    System.out.println(x: bad_guy_2);

    rambo.shootEnemy(enemy: bad_guy_1);   // health 4
    System.out.println(x: bad_guy_1);

    rambo.shootEnemy(enemy: bad_guy_2);  // health 0

    rambo.shootEnemy(enemy: bad_guy_1);   // health 3
    rambo.shootEnemy(enemy: bad_guy_1);   // health 2
    rambo.shootEnemy(enemy: bad_guy_1);   // health 1
    System.out.println(x: bad_guy_1);
    stHellen.healPatient(soldier:bad_guy_1);   // health 2
    System.out.println(x: bad_guy_1);
    rambo.shootEnemy(enemy: bad_guy_1);   //health 1
    rambo.shootEnemy(enemy: bad_guy_1);   //health 0
    stHellen.healPatient(soldier:bad_guy_1);
}
```

The output of my tester program is:



```
The Green Beret has a health of 5 and has the following weapons: [Bow & Arrow, Big knife, Bazooka]
The Snipper has a health of 5 and has the following weapons: [Riffle, Pistol]
The Archer has a health of 1 and has the following weapons: [Granade]
The Green Beret shoots at the Snipper
The Snipper has a health of 4 and has the following weapons: [Riffle, Pistol]
The Green Beret shoots at the Archer
        The Archer is dead
The Green Beret shoots at the Snipper
The Green Beret shoots at the Snipper
The Green Beret shoots at the Snipper
The Snipper has a health of 1 and has the following weapons: [Riffle, Pistol]
The Snipper is at the hospital
        Doctor House is taking care of the Snipper
        The patient Snipper is recovering
The Snipper has a health of 2 and has the following weapons: [Riffle, Pistol]
The Green Beret shoots at the Snipper
The Green Beret shoots at the Snipper
        The Snipper is dead
The Snipper is at the hospital
        Doctor Dre is taking care of the Snipper
        There is nothing we can do. The patient Snipper is already dead
```

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: **Lab6_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!