CS 111, Programming Fundamentals II
Lab 2: Robot Garage

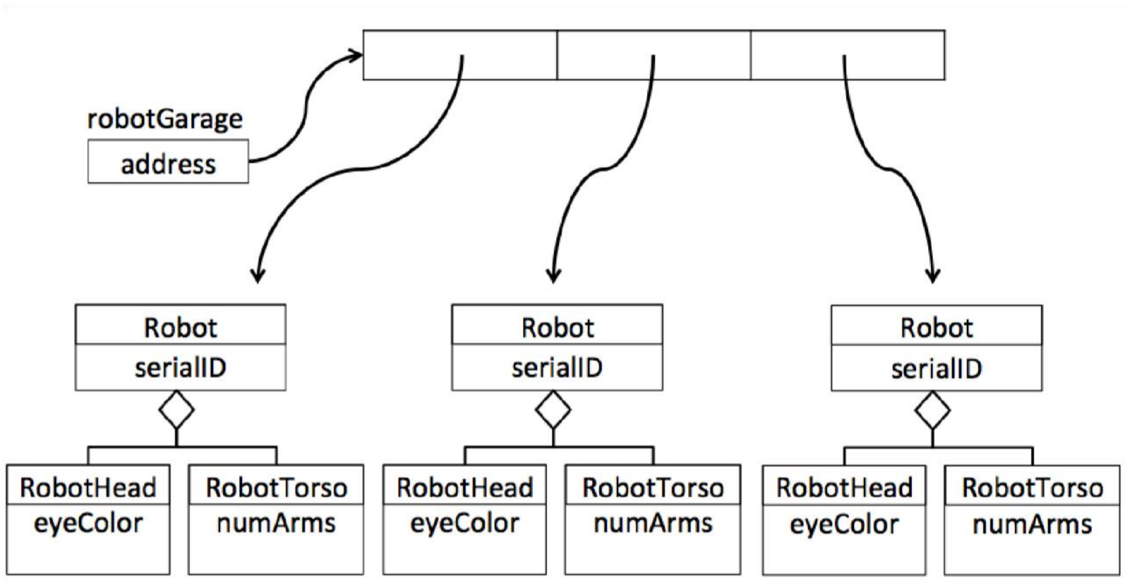
Computer Science

In lecture we've discussed the ***equals*** method, ***copy*** method and the ***toString*** method. You will gain experience writing them in this lab.

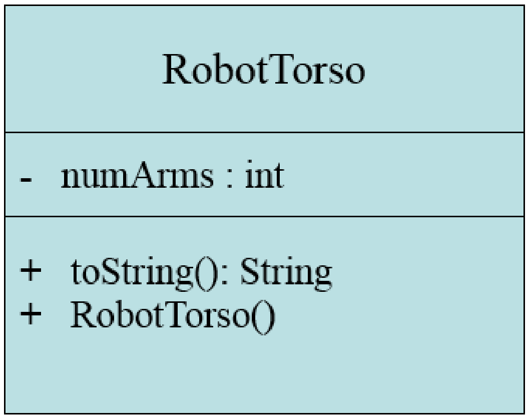
I. Introduction

To motivate this lab, assume that a robot factory assembly line builds futuristic robots, all with the first part of their name being "***ArmyRobot***" and the second part of their name being a random integer value. The combined first and second part of the name for each robot is used to construct a robot's ID number (***serialID***). For example the factory might produce robots with IDs "***ArmyRobot45678"*** or "***ArmyRobot4549008"***. A robot is made up of a head and torso.

Your task is to write a program that simulates a robot garage, which is an array of three **Robot** objects. The **RobotTorso** and **RobotHead** classes are small. The **Robot** class is a bit more complex, because it contains an ***equals()*** methods, a ***copy()*** method, as well as a few getter and setter methods. The third robot in your array must have a ***serialID*** that is identical to the ***serialID*** of one of the other robots (you'll use the ***copy()*** method to create the third robot). The **Robot** class has a static field, ***numRobots*** that keeps count of how many robots there are. The figure below is a schematic representation of your robot garage:



II. RobotTorso Class

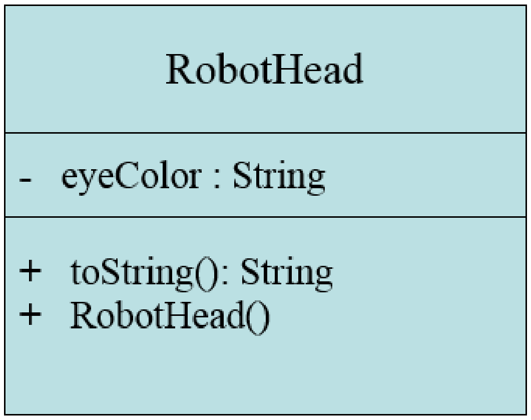


Write the class **RobotTorso** according to the UML diagram below. The ***numArms*** field must be private. The constructor should set the value of ***numArms*** instance field to a random integer value between 0 and 10 (inclusive). As was discussed in lecture, it is a good practice to include a ***toString***() method in each class, which should output a String that contains the information (or state of) the instance **RobotTorso**. In this case, for example if the **RobotTorso** field ***numArms*** is 9, the ***toString*** method should output :

```
Number of arms : 9
```

Write this class and make sure it compiles before moving on to the next one. The entire class can be written 20 lines or fewer.

III. RobotHead Class



The **RobotHead** class is also straight-forward. It is made up of a single instance field, a constructor and a ***toString*** method. The constructor should generate a random number between 0 and 5 (inclusive), and based on that number, assign a unique value to the instance field ***eyeColor***. You select the eye colors. You could use an if-else or a switch on the randomly generator number to assign the color. For example if the number is 0, then the color could be red, if the number is 1 the color could be green and so on. The ***toString*** method is very similar to the one in the class **RobotTorso**, it will print out the value in the instance field. If the ***eyeColor*** is “brown”, then the ***toString*** method should print out:

```
Eye color : brown
```

IV. Robot Class

The **Robot** class is an aggregate class, that is made up of four instance fields (one of which is static), a constructor, and 6 utility methods (one of which is static), as shown in the UML below. Remember that by convention static fields and methods are underlined in an UML. All of the instance fields need to be private.

```
Robot
-------------------------------------------
-   numRobots : int
-   robotHead : RobotHead
-   robotTorso : RobotTorso
-   serialID : String
-------------------------------------------
+   Robot ()
+   setSerialID(serial : String): void
+   getSerialID(): String
+   getRobotCount(): int
+   toString(): String
+   equals(aRobot : Robot): boolean
+   copy(): Robot
```

Robot constructor:

- Increment the static field ***numRobots*** by 1.
- Generate a random number between 0 and 100,000. Append the number to a string "***ArmyRobot***" and assign the string to the field ***serialID***. For example if the number is 9865345, then the serialID should be "***ArmyRobot9865345***".
- Instantiate the fields ***robotHead*** and ***robotTorso***. For example , to create a new **RobotHead** instance field, you'd write:

```
robotHead = new RobotHead();
```

This creates a new instance of **RobotHead** object, and assigns the reference of the new object to the instance filed ***robotHead***.

copy() method:

- Create a new **Robot** object.
- Set the ***serialID*** of the newly created object to be the ***serialID*** of the **Robot** object from which the copy method is called. The full code for this method can be seen below.

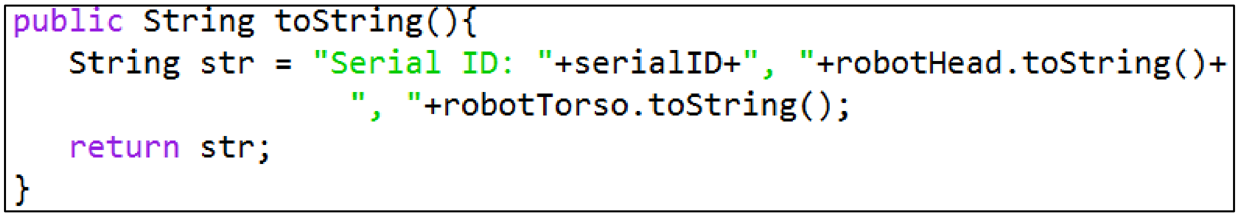
```
public Robot copy(){
    Robot robotCopy = new Robot();
    robotCopy.setSerialID(serialID);
    return robotCopy;
}
```

equals() method:

- Checks if the ***serialID*** in the instance from where the method is being invoked is the same as the ***serialID*** of the instance that is being received as the argument. Return true if the ***serialIDs*** are the same. Refer to the lecture slides if you need some help.

toString() method:

- Creates and returns a string which contains the ***serialID*** and also appends the returning values of the ***robotTorso*** and ***robotHead toString*** methods. The code for this method can be seen below.



```java
public String toString(){
   String str = "Serial ID: "+serialID+", "+robotHead.toString()+
                  ", "+robotTorso.toString();
   return str;
}
```

setSerialID() and **getSerialID()** methods:

- Set and return the field ***serialID***.

getRobotCount() method:

- Returns the field ***numRobots***.

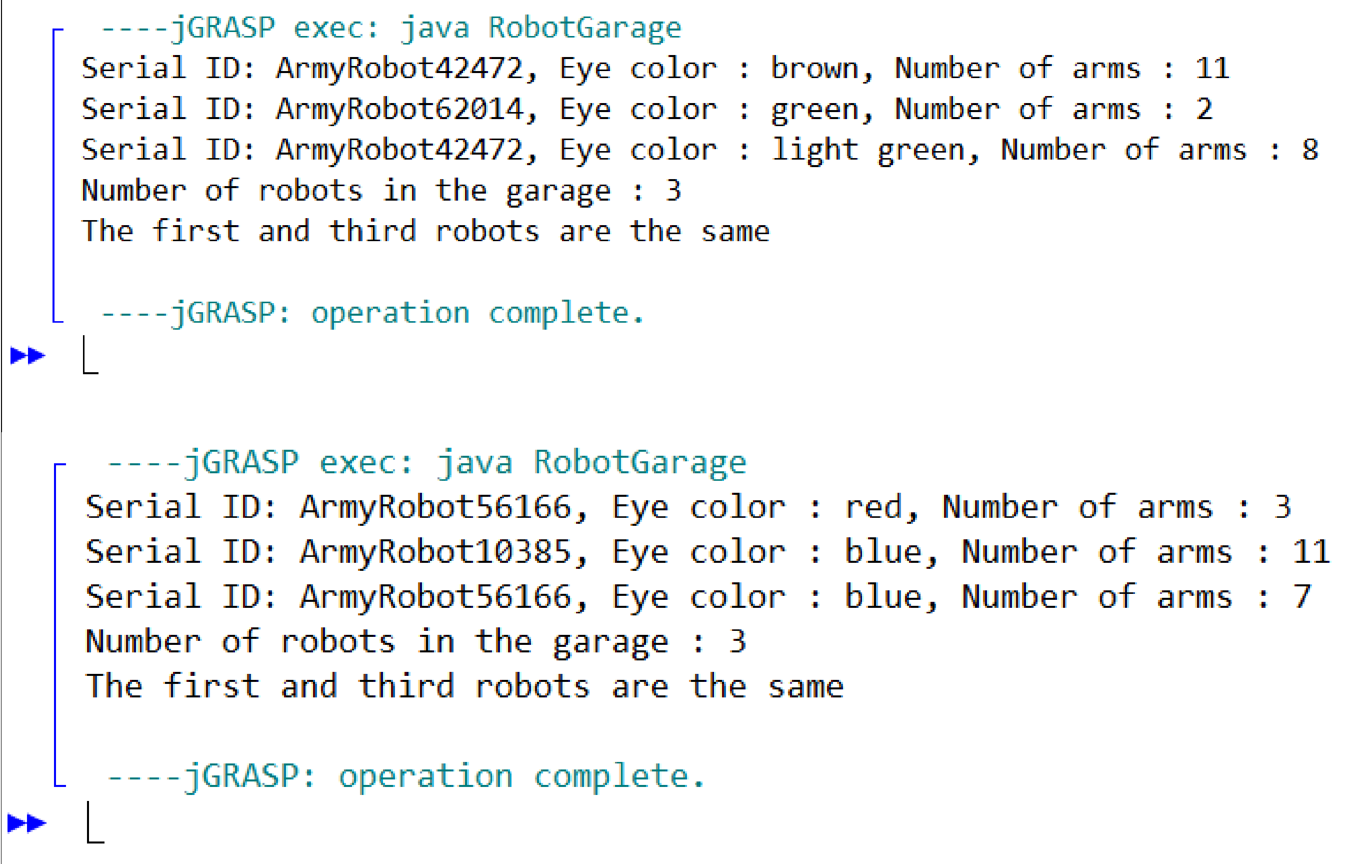
V. RobotGarage Class

Now that you've written the **Robot**, **RobotTorso**, and **RobotHead** classes, you will use them in the class **RobotGarage**. This will be the driver class for this programming assignment, it will have the main method. Figure 1 contains the pseudocode for the ***main*** method of this class.

```
Declare an array of 3 Robot objects

Instantiate new Robot objects into indices 0 and 1 of the array of Robots

Use the copy() method from one of the robots in indices 0 or 1 to create a new robot
object, and place the copy of the robot into index 2 of the // array of Robots

Use the toString method in each robot object to print to the screen details about
that robot

Retrieve the value of the static variable numRobots and print it to the // screen

Use a series of if-then statements to check if robots at indices 0 and 1, 0 and 2, or 1
and 2 are identical using the equals method in the Robot class, and print to the screen
which robots are identical
```

Figure 1: Pseudocode for main method of RobotGarage

VI. Sample outputs



```
 ----jGRASP exec: java RobotGarage
Serial ID: ArmyRobot42472, Eye color : brown, Number of arms : 11
Serial ID: ArmyRobot62014, Eye color : green, Number of arms : 2
Serial ID: ArmyRobot42472, Eye color : light green, Number of arms : 8
Number of robots in the garage : 3
The first and third robots are the same

 ----jGRASP: operation complete.
```

```
 ----jGRASP exec: java RobotGarage
Serial ID: ArmyRobot56166, Eye color : red, Number of arms : 3
Serial ID: ArmyRobot10385, Eye color : blue, Number of arms : 11
Serial ID: ArmyRobot56166, Eye color : blue, Number of arms : 7
Number of robots in the garage : 3
The first and third robots are the same

 ----jGRASP: operation complete.
```

Figure 2: Sample Outputs

VII. Upload your work to Canvas

For this lab, make sure that you upload the following files to the Lab 2 assignment in your Canvas account:

RobotHead.java
RobotTorso.java
Robot.java
RobotGarage.java
Screenshot image containing an output of the program

Using the ***Snipping Tool*** take a screenshot of the output and save it as an image.

There will be additional files (the .class files, that you've generated when compiling your code) in your lab2 folder, but don't upload them to Canvas. Those files are not graded; they are just the byte code files that are used by the Java Virtual Machine.

Rubric

File / Lab	**Points**
Class *RobotHead* written and compiles per UML diagram	15
Class *RobotTorso* written and compiles per UML diagram	15
Class *Robot* written and compiles per UML diagram	30
Class *RobotGarage* written according to the pseudocode	20
Class *RobotGarage* produces output as expected	10
Screenshot of the output, code is commented and formatted	10
Total	100