CS241, Homework 1 – Java and OOP fundamentals

1. Suppose we create a Coin class that is designed to represent a coin that can be tossed to come up with heads or tails, each with a 50% chance. Each coin object should also store a record of all the tosses it has made.

```
public class Coin {
  ______ ArrayList<Boolean> pastTosses;  // stores a record of each toss:
                                         // true=heads, false=tails
  public Coin() {
    pastTosses = new ArrayList<>();
  }

  public ______ toss() {
    if (Math.random() >= 0.5) {
      // do something
    }
    else {
      // do something else
    }
    // maybe more code?
  }

  public boolean getLastToss() { // you will add code here
  }
}
```

   a. Should the pastTosses instance variable be public, private, or protected? Explain your choice.

   b. Should toss() have a return value? Explain your answer.

   c. Explain how you would fill in the two (or perhaps three) blank sections marked with comments in the toss() method.

   d. Should there be a getter for pastTosses? Why or why not?

   e. Should there be a setter for pastTosses? Why or why not?

   f. Fill in the getLastToss() method.

2. Design and write code for a TrickCoin class that allows the user to specify the likelihood that the coin will come up heads vs tails. TrickCoin should be a subclass of Coin (TrickCoin extends Coin), and should have a constructor TrickCoin(double percent) that allows the user to specify the percentage of the time the coin comes up heads. percent will be a number between 0 and 1, with 0=0%, and 1=100%.

   Make sure to override toss() correctly, and add/override any other methods you deem appropriate. You can also choose to make any changes you want to the original Coin class.

   This question is intended to be done on paper, not at a computer. That is, I've purposefully made this simple enough that you should try to do it without running it in an IDE. You will not be graded on syntax. (This is the kind of question that would be appropriate for a test --- a small coding question to be done on paper).

3. Suppose we have the following FruitSalad class:

```
public class FruitSalad {
  private ArrayList<String> fruits;
  private int numServings;

  public FruitSalad(int servings) { }

  public void addFruit(String newFruit) { }

  public void serve() { }
}
```

   a. Fill in the constructor for the FruitSalad class, which should just initialize numServings to the argument specified in the constructor, and fruits to an empty ArrayList.

   b. Fill in the addFruit method, which takes a String argument newFruit (which would be something like "apples", "melon", etc). This function should add this fruit to the end of the fruits list.

   c. Fill in the serve function. This function, when called, should print "Serving fruit" but only up to the number of times specified by the numServings field. After that, when called, the function should print "No fruit left." As an example, if numServings is 3, then the FruitSalad class should allow serve() to be called 3 times and print "Serving fruit" each time. On the 4th call or subsequent calls, it should print "No fruit left."

      In addition, after the first call to serve(), no more fruit should be able to be added to the salad with addFruit().

      To be clear: Each call to serve() only prints "Serving fruit" at most once.

      You may add additional fields to the class, if desired, for question C.

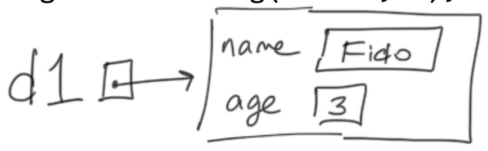
4. Assume you have a Dog class with (private) fields for name and age and (public) methods getName(), getAge(), and setAge(). The constructor for the Dog class takes the dog's name and age and sets them to the values given.

Part A: Determine the output of the code below.

Part B: Draw two diagrams illustrating what the computer's memory looks like for the first half of the code (before the array section starts), and the second half (the array section). Illustrate how memory looks with boxes and arrows as was done in class. The specific way you draw the diagrams is not important, but it should be clear when memory is being shared (two arrows pointing to the same box).

```
Dog d1 = new Dog("Fluffy", 3);
Dog d2 = new Dog("Lila", 4);
Dog d3 = new Dog("Daisy", 5);
Dog d4 = d3;

System.out.println(d1.getAge() + " " +  d2.getAge() + " " + d3.getAge() + " " +
d4.getAge());

d2.setAge(6);
d4.setAge(7);
d3.setAge(8);
d4 = d1;
d1.setAge(9);
d4 = new Dog("Loki", 10);

System.out.println(d1.getAge() + " " +  d2.getAge() + " " + d3.getAge() + " " +
d4.getAge());

// Draw a diagram illustrating the state of memory for everything above here, and a
// second diagram for everything below.

ArrayList<Dog> dogs = new ArrayList<Dog>();
dogs.add(new Dog("Ginger", 3));
dogs.add(new Dog("Buster", 4));
dogs.add(new Dog("Lucky", 5));
dogs.add(new Dog("Rocky", 6));
Dog anotherdog = dogs.get(3);
dogs.set(3, dogs.get(1)); // dogs[3] = dogs[1]
dogs.set(2, anotherdog);  // dogs[2] = anotherdog
dogs.get(2).setAge(7);
anotherdog = dogs.get(0);
anotherdog.setAge(8);
dogs.get(0).setAge(dogs.get(0).getAge() + 1);

System.out.println(
          dogs.get(0).getAge() + " "
        + dogs.get(1).getAge() + " "
        + dogs.get(2).getAge() + " "
        + dogs.get(3).getAge() + " "
        + anotherdog.getAge());
```

Example for how you might illustrate
Dog d1 = new Dog("Fido", 3);


d1
name
Fido
age
3