Exercise 1.

Suppose you are programming a game. You are planning to have a number of game objects that can be collected by players in different ways. Consider the following `GameObject` interface and `PowerUp` class:

```
public interface GameObject {
    void collect();
}

public class PowerUp implements GameObject {
    private int count;
    private boolean collected;
    public PowerUp() {
        count = 0;
        collected = false;
    }
    public void collect() {
        count++;
        if (!collected) {
            collected = true;
            System.out.println( "Power-up collected " + count + " time!" );
        } else
            System.out.println( "Power-up " + count + " times!" );
    }
}
```

Can you define a test class that has a reference variable of type `GameObject` and make it point to a `PowerUp` object? What happens if that item is collected two times? What is the output?

Next, define a class called `Treasure` that `implements GameObject`. Each treasure object should be instantiated with a unique value representing how much that treasure is worth when collected. Define a constructor for this class and override the `collect()` method as well.

Once you are done with the `Treasure` class, modify the test class you had earlier so that the same reference variable now points to a new `Treasure` object. Collect the items a couple of times. What is the output?

Exercise 2.

Let's say you were writing an employee scheduling program to track who is working when. Suppose you were given the following `Schedule` class that uses a two-dimensional array to keep track of each employee's weekly allocated hours:

```
public class Schedule implements Cloneable {

    private int[][] hours; // [7][2] -> start/end for each day
    public Schedule() {
        hours = new int[7][2];
    }

    public void setHours(int day, int start, int end) {
        hours[day][0] = start;
        hours[day][1] = end;
    }

    public int getStart(int day) { return hours[day][0]; }
    public int getEnd(int day) { return hours[day][1]; }

    // implement the clone() method here
    // ...
}
```

To make your code more reusable, you have decided that this class will implement the `Cloneable` interface. Complete this class by writing the clone() method and make sure it performs a **deep copy** of the clone.

When you are done, add a test class that creates a new `Schedule` object with some working hours set. Print that out to make sure it works as expected (you may wish to add a `toString()` method to the `Schedule` class for convenience).

After that, make a clone of the object, and change the work hours to something else. Print the hours from both objects to make sure the first one remains intact and the second one has different hours.