Answers for the first G51PRG self-test

1. What will be printed after the following lines of code are executed:

```
String string1 = new String("Hello");
String string2 = new String("Hello");
System.out.println(string1 == string2);
```

   Answer: "false". `new` allocates new memory, addresses of `string1` and and `string2` will be different, `==` compares addresses.

2. What will be printed after the following lines of code are executed:

```
String string1 = new String("Hello");
String string2 = new String("Hello");
System.out.println(string1.equals(string2));
```

   Answer: "true". `equals()` method of strings checks if they have the same characters.

3. What will be printed after the following lines of code are executed:

```
String string1 = new String("Hello");
String string2 = string1;
System.out.println(string1 == string2);
```

   Answer: "true". Addresses of `string1` and and `string2` will be the same since `string2` has been assigned reference/address of `string1`.

4. Consider the following class definition:

```
public class Employee{
   public int payroll;
   public static int retirementAge = 65;

   Employee(int i){
      payroll = i;
   }

   public static void main(String[] args){
      Employee x = new Employee(10146);
      Employee y = new Employee(10147);
      x.retirementAge = 60;
      System.out.println(y.retirementAge);
   }
}
```

What is `x` in the main() method above: (a) a class, (b) an object, (c) a number, (d) an int?
Answer: b

5. What will be printed after the main() from question 4 is executed? Answer: 60. `x` and `y` share the `retirementAge` field.

6. Replace the main() method from question 4 with the following one:

```
public static void main(String[] args){
   System.out.println(retirementAge);
}
```

Will the compiler report an error? If not, what will be printed after the main() is executed?
Answer: no error, 65. Static fields can be accessed without an object.

7. Replace the main() method from question 4 with the following one:

```
public static void main(String[] args){
   System.out.println(payroll);
}
```

Will the compiler report an error? If not, what will be printed after the main() is executed?
Answer: the compiler will report an error (static reference to an instance field). Need to know whose payroll to access.

8. Write the code to make a copy of an array of integers. It should return an array which contains the same integers but has a different address in memory. The simplest answer is like this:

```
public class ArrayUtilities {
   public static int[] copy(int[] array){
     int[] newArray = new int[array.length];
     for (int i=0; i < array.length; i++){
        newArray[i]=array[i];
     }
     return newArray;
   }
}
```

9. Complete the last line of code below to assign a copy of `myArray` to `newArray` (call the method `copy()` from the previous question).

```
int[] myArray = {1,4,5};
int[] newArray = ArrayUtilities.copy(myArray);
```

Consider the following class definitions:

```
class Rectangle {
   protected double width;
   protected double height;

   public Rectangle(double x, double y){
      this.width = x;
      this.height = y;
   }

   public double surface(){
      return width * height;
   }
}

class Square extends Rectangle{

   public Square(double x){
      super(x,x);
   }
}
```

10. Can a method in the `Square` class access `width` and `height` fields?
    Answer: yes; they are declared protected which means that methods in subclasses can access them.

11. Would any of the following lines cause a compiler or runtime error (assuming the rest of the code is fine)? If not, what will it print?

    ```
    Square s = new Square(4.0);
    System.out.println(s.surface());
    ```

    Answer: no error, will print 16.0. `Square` inherits `surface()` method from `Rectangle`.

12. Would the following line cause a compiler or runtime error (assuming the rest of the code is fine)?

    ```
    Rectangle[] r = new Rectangle[10];
    r[0] = new Square(4.0);
    ```

    Answer: no error. Can use a `Square` where a `Rectangle` is expected (inheritance polymorphism). No need to do explicit casting.

13. Would the following line cause a compiler or runtime error (assuming the rest of the code is fine)?

```
Square[] s = new Square[10];
s[0] = new Rectangle(4.0, 4.0);
```

Answer: will not compile:

```
Rectangle.java:16: incompatible types
found   : Rectangle
required: Square
      s[0] = new Rectangle(4.0, 4.0);
             ^
1 error
```

If add explicit casting:

```
s[0] = (Square) new Rectangle(4.0, 4.0);
```

will compile but produce a runtime error: ClassCastException.