COM6516 Object oriented programming and software design: Practical session 3

The main aim of this practical is to work though the material on inheritance that we covered in the lectures. You should work through this worksheet on your own or with a friend.

Part 1: Review Example code from last week
Take a look at the example solution for last weeks' tasks, and compare the program to your own. Do they produce the same output? What are the good features of each program? How could you improve on your own program?

Part 2: Java programming review
Write a Java program `Multiplication.java` that prints out a multiplication table that looks like the following:

```
    1   2   3   4   5   6   7   8   9
---------------------------------------
1 | 1   2   3   4   5   6   7   8   9
2 | 2   4   6   8  10  12  14  16  18
3 | 3   6   9  12  15  18  21  24  27
4 | 4   8  12  16  20  24  28  32  36
5 | 5  10  15  20  25  30  35  40  45
6 | 6  12  18  24  30  36  42  48  54
7 | 7  14  21  28  35  42  49  56  63
8 | 8  16  24  32  40  48  56  64  72
9 | 9  18  27  36  45  54  63  72  81
```

Try and get the alignment correct by adjusting the number of spaces for two-digit numbers.

Part 3: An excursion into recursion
Recursion is when a program calls itself – see
http://www.danzig.us/java_class/recursion.html

The code in the file `Fib.java` uses both recursion and iteration to generate a Fibonacci series, in which each successive number is the sum of the two previous numbers. If the first two numbers are set to 1, the Fibbonacci series is 1, 1, 2, 3, 5, 8, 13, …. Examine `Fib.java` to explore the difference between an iterative and a recursive solution. Compile and run the code. How long does each version take? Why is one version more efficient than the other?

Increase the maximum value from 40 to e.g. 50 and see what happens. Are the results correct? If not, why not?

Part 4: Class structure and equals methods
In Java, we can define either instance variables or methods to be `static` within a class. This means that these variables and methods are associated with the class, and not with individual objects. An object of the class does not need to be created in order to access static instance fields (sometimes called class fields) or to run static methods (sometimes called class methods).

The Java `Math` class (part of the API – see http://docs.oracle.com/javase/8/docs/api/) is an example of this idea in practice. Somewhere in the `Math` class code, `PI` is declared as

```
    public static final double PI = 3.141592653589793;
```

This means that we can access `PI` using `Math.PI` (e.g. `System.out.println(Math.PI)`) in any Java program, without creating a `Math` object. Similarly, our `main` methods are declared `static`, and can be accessed by the Java virtual machine without creating any objects. For a detailed discussion see – http://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

Take a look at the `Circle` class provided. This class has both a static (class) variable `PI`, and an instance variable `radius`. It also has a static (class) method `radToDeg`, and instance methods `area` and `circumference`.

Write a `TestCircle` class that tests the static variable and class using the following code:

```
public static void main(String args[]) {
    System.out.println(Circle.PI);
    System.out.println(Circle.radToDeg(3.141));
}
```

When might it be a good idea to use static (class) methods and variables?

Part 5: Making use of inheritance from the `Object` superclass

The idea of inheritance is a powerful aspect of OO programming, and all classes in Java inherit some basic methods and properties from the `Object` superclass. This means that when we create a `Circle` object, it inherits some functionality. Take a look at the API documentation for `Object` at http://docs.oracle.com/javase/8/docs/api/. Scroll down the left panel and click on `Object`.

Add some code to your `TestCircle` class to create a `Circle` object with radius `3`, and try out some of these methods, e.g. if your circle object is `myCircle`, then try

```
System.out.println(myCircle.toString());
System.out.println(myCircle.getClass());
```

The default `toString` method does not return a very helpful result, so write your own `toString` method in the `Circle` class. This new method will ***override*** the default `toString` method in the `Object` class.

Add code to your `TestCircle` class to create a second `Circle` object (e.g. `myCircle2`) with radius `3`. What result do you get if you use the default `equals` method to test whether `myCircle` and `myCircle2` are the same? Write your own `equals` method in the `Circle` class that tests for equality (are the object states the same?) and identity (do the object references point to the same memory location), and overrides the default `equals` method.

Part 6: An inheritance hierarchy in Java

Inheritance hierarchy is covered in the lecture notes. Work through the example code, and make sure you understand how to call methods in the superclass and subclass.

Using the example code as a template, implement a new class called `PhDThesis`, which inherits from `Publication`, and includes the additional fields `numChapters`, `university`, and `supervisor`, which describe the number of chapters in the thesis, the University where it was completed, and the name of the student's supervisor. Write a test class `PublicationTest` that (i) creates `Publication`, `Book`, `MagazineArticle`, and `PhDThesis` objects, and (ii) tests their `toString` methods. Use `super` to invoke the constructor from `Publication`.

Note that the `Publication` class has two constructors:

```
public class Publication {

    public Publication() {
        title = "default";
        author = "default";
        isbn = 0;
        numPages = 0;
    }

    public Publication(String t, String a, int i, int n) {
        title = t;
        author = a;
        isbn = i;
        numPages = n;
    }

    …
}
```

Remove (or comment out) the `super` statement in the `Book` class (or one of the other subclasses of `Publication`) as follows:

```
public class Book extends Publication {

    public Book(String t, String a, int i, int n, int c) {
//      super(t, a, i, n);
       numChapters = c;
    }

    …
}
```

What happens when you create objects of the `Book` class now?

Part 7: Random number generation (a more complex task hence most likely homework)
You can use class `Random` to generate random numbers –
http://docs.oracle.com/javase/7/docs/api/java/util/Random.html
You can use it as follows:

```
Random random = new Random();
int a = rnd.nextInt(10);  // 'a' will be in range 0..9
```

Remember to include this line at the top of your file:

```
import java.util.Random;
```

A better way is to pass an argument to the constructor like this:

```
Random random = new Random(0);
```

This gives you a sequence of pseudo-random values that is determined by that number (seed, here we chose to use `0`) and ensures that the next run of your program gives the same sequence.

Modify the program from **part 2** to number rows and columns with random numbers `1..9` without repetitions.