COMP2911 Engineering Design in Computing

Laboratory - Week 1

Exercise 1: Hello World

Hello.java

```
public class Hello {
    public static void main (String[] args) {
        System.out.println("Hello World!");
    }
}
```

javap Hello

```
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
    public Hello();
    public static void main(java.lang.String[]);
}
```

javap -c Hello

```
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
public Hello();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return

public static void main(java.lang.String[]);
  Code:
   0:   getstatic   #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:   ldc #3; //String Hello World!
   5:   invokevirtual   #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   8:   return

}
```

Exercise 2: Hello Wayne's World

Hello.java

```
public class Hello {
    static String wayne = "Hello Wayne";
    public static void main (String[] args) {
        System.out.println(wayne);
    }
}
```

javap Hello

```
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
    static java.lang.String wayne;
    public Hello();
    public static void main(java.lang.String[]);
    static {};
}
```

javap -c Hello

```
Compiled from "Hello.java"
public class Hello extends java.lang.Object{
static java.lang.String wayne;

public Hello();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return

public static void main(java.lang.String[]);
  Code:
   0:   getstatic   #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:   getstatic   #3; //Field wayne:Ljava/lang/String;
   6:   invokevirtual   #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
   9:   return

static {};
  Code:
   0:   ldc #5; //String Hello Wayne
```

```
   2:   putstatic   #3; //Field wayne:Ljava/lang/String;
   5:   return

}
```

*Omitting the static declaration for the **wayne** field*

```
public class Hello {
    String wayne = "Hello Wayne";
    public static void main (String[] args) {
        System.out.println(wayne);
    }
}
```

Gives the following error:

```
Hello.java:4: non-static variable wayne cannot be referenced from a static context
        System.out.println(wayne);
                           ^
1 error
```

Works if I do this

```
public class Hello {
    public static void main (String[] args) {
        String wayne = "Hello Wayne";
        System.out.println(wayne);
    }
}
```

Exercise 3: Minimum Element in an Array

I've implemented a boring iterative solution, as well as one that uses the `java.util.*` package to sort the array and return the first element.

```
import java.util.*;

class MinimiseUtils {

    /**
     * Minimum of two integers.
     * @param i first of two integers
     * @param j second of two integers
     * @return minimum of i and j
     */
    static int min2(int i, int j) {
        return i < j ? i : j;
    }

    static int min3(int i, int j, int k) {
        int temp = min2(j, k);
        int result = min2(i, temp);
        return result;
    }

    static int min(int[] data) {
        return min(data, 0, data.length - 1);
    }

//  Cheating Version of array minimum computation
/*
    private static int min(int[] data, int start, int end) {
        Arrays.sort(data);
        return data[0];
    }
*/

//  Iterative Version of array minimum computation
    private static int min(int[] data, int start, int end) {
        int min = data[start];
        for (int i = start ; i <= end ; i++) {
            for (int j = i + 1 ; j <= end ; j++) {
                if (min > min2( data[i], data[j] )) {
                    min = min2( data[i], data[j] );
                }
            }
        }
        return min;
    }

//  Recursive Version of array minimum computation
/*
    private static int min(int[] data, int start, int end) {
        return start == end ?
                data[start] : // There is only one element
                start > end ?
                    Integer.MAX_VALUE : // Return a massive value I guess to show that there the min is doesn't really exist in a biz
                    min2(data[start], min(data, start + 1, end)); // Find the min recursively
    }
*/

    public static void main(String[] args) {
```

```
        int i = 99;
        int j = 55;
        int k = 11;
        System.out.print("Minimum of " + i + ", " + j + ", " + k + " is ");
        System.out.println(min3(i, j, k));

        int[] data = {45, 23, 65, 24, 36, 63, 62, 15};
        System.out.println("Minimum of test data array is " + min(data));
    }
}
```

Challenge

Implementation

Computing factorials recursively (perhaps not so appropriate for large n), using `int`, `double` and `BigInteger` declarations.

```
import java.math.BigInteger;

public class Factorial {

    static int factorial_int ( int n ) {
        return n < 2 ? 1 : n*factorial_int(n-1);
    }

    static double factorial_double ( int n ) {
        return n < 2 ? 1 : n*factorial_double(n-1);
    }

    static BigInteger factorial_bigInteger ( int n ) {
        return n < 2 ? BigInteger.ONE : BigInteger.valueOf(n).multiply(factorial_bigInteger(n-1));
    }

    public static void main (String[] args) {
        int n = Integer.parseInt(args[0]);
        System.out.println(factorial_int(n));
        System.out.println(factorial_double(n));
        System.out.println(factorial_bigInteger(n));
/*
        for (int i = 1 ; i <= Integer.parseInt(args[0]) ; i++) {
            System.out.println(i+"!=");
            System.out.println(factorial_int(i)+" (Integer)");
            System.out.println(factorial_double(i)+" (Double)");
            System.out.println(factorial_bigInteger(i)+" (BigInteger)");
        }
*/
    }
}
```

Testing

- **How do you know what code is OK?**

  Not entirely sure what this means.

- **How do you know what 60! is?**

  By inspection. *Just kidding*.

  We can compute this in well-tested mathematical computing software such as Matlab or Maple. Or we can work out the correct answer by hand. I prefer the former.

- **How do you know the answer is correct? Exactly?**

  To get a rough idea of the accuracy of the result, we can apply a special case of De Polignac's formula to see that the number of trailing zero is correct. But knowing is the answer is **exactly** correct might be more difficult.