**Practical 03**

***Recursion***

Recursive programming can be somewhat difficult to understand initially, especially when as novice programmers we are ‘*trained’* to think iteratively rather than recursively. Practice as with most things in coding, can help overcome this, these practical exercise will help you better understand recursive programming techniques.

Recursion is when a method/function calls itself. That is, in the course of the function definition there is a call to that very same function. At first this may seem like a never ending loop, or like a dog chasing its tail. It can never catch it. So too it seems our method will never finish. This might be true in some cases, but in practise we can check to see if a certain condition is true and in that case exit (return from) our method.

1. From the accompanying (Word) tutorial, implement the recursive factorial method.  
   **Completed in Factorial.java**
2. Using the accompanying (PowerPoint) lecture, implement the code for the Sentence Class along with its isPalindrome() method. **Completed in Sentence.java**
3. Write a simple helper (tester) class that will create a Sentence object and invoke the isPalindrome() method on a sentence. **Completed in Q3.java**
4. Implement the improved isPalindrome()method using substrings of the original sentence. **Completed in Sentence.java, Q3.java**
5. Add a reverseString() recursive method to the Sentence Class (Hint: think about recursively parsing each character from the string before building it in reverse order).  
   **Completed in Sentence.java**
6. Write a simple helper (tester) class that will create a Sentence object and invoke the reverseString() method printing the original sentence along with it in reverse.  
   **Completed in Q6.java**
7. Using your code from Practical’s 01 & 02, implement the **StopWatch** class to measure the efficiency of each of the 4 algorithms (Bubble Sort, Enhanched Bubble Sort, Insertion Sort & Selection Sort) used to date. Do this with each of the 3 differently arranged data sets as per steps 6, 7 and 8 of Practical’s 01 & 02. See Quadratic Algorithms lecture for notes on creating a timer using the System.currentTimeMillis() method in the **StopWatch** class – **use the nanosecond version of this**, as the millisecond version may not record a time (it may compete the sort under a millisecond depending on your system). **Completed in StopWatch.java, Q7p1.java, Q7p2.java, Q7p3.java**