Laboratory 1 – Week 2
Familiarisation with the Java Programming Environment

1.1 Introduction

This module requires **quite a lot** of Java programming. During the exam period and the summer break, you might not have had much Java programming practice. This laboratory is a "gentle" Java refresher.

1.2 Preliminaries

If you are working from a university machine, make sure you have access to your home directory (we will assume this is the H: drive). It might be worth creating a sub-directory where you store your CS2004 Java programs. Within these laboratories we will be creating a number of programs, you may edit existing programs or create new ones as you see fit, however it may be better if you create new programs so that you have all of the programs you have created for revision and reference.

1.3 Eclipse

Within this module we will be using the Eclipse programming environment to develop our Java programs. If you need a refresher on Eclipse, Table 1.1 has some suggested links.

Link	Notes
http://eclipsetutorial.sourceforge.net/totalbeginner.html	A video tutorial - needs audio
http://www.vogella.com/tutorials/Eclipse/article.html	Eclipse IDE tutorial
http://www.tutorialspoint.com/eclipse/	Eclipse tutorial

Table 1.1. Eclipse Tutorial Links

1.4 Exercise 1

Start Eclipse and create a new project called "Lab1", and then create a new class called "IloveCS2004". In the editor of the class ILoveCS2004, write/insert/copy the following code:

```
/**
 * filename: ILoveCS2004.java
 * Author: Your Name!
 * Date: Jan. 01, 1978 (or Today's date!)
 */
public class ILoveCS2004
{
      public static void main(String[] args)
      {
          System.out.println("I love CS2004!");
      }
}
```

Compile the class. If you get no errors, then run the program from the `Run` menu. You should get

the message "I love CS2004!" appearing in the `Console Window`. If you wish to save any of the text that appears as output in this window, then hold the mouse over the window, right click the mouse button, and you can then copy and paste as appropriate, e.g. click *Select All*, and then *Copy* and then save it to a text file.

1.5 Exercise 2

Now, let's consider our first algorithm (in CS2004), namely an approximation algorithm to compute square roots that was known to the ancient Greeks, called *Heron's* algorithm. The algorithm starts by guessing a value *x* that might be somewhat close to the desired square root $\sqrt{a}$. The initial value does not have to be very close; *x*=*a* is a perfectly good choice (but obviously wrong!).

Consider the quantities of *x* and a/x.

If $x < \sqrt{a}$, then $a/x > a/\sqrt{a}$ thus $a/x > \sqrt{a}$. Similarly, if $x > \sqrt{a}$, then $a/x < a/\sqrt{a}$ thus $a/x < \sqrt{a}$. Combining these simple equations results in us knowing that $\sqrt{a}$ always lies between *x* and a/x. We can use this fact to construct an algorithm to compute the square root of a number.

We make the midpoint of the interval our improved guess of the square root, i.e., let $x_{new} = (x + a/x)/2$ and then we repeat the procedure – that is, compute the midpoint between x_{new} and a/x_{new}. We stop when two successive approximations differ from each other by a very small amount. The method converges very rapidly.

Since this is the first laboratory session for implementing algorithms, you will be provided with the Java source code. However, from the next laboratory onwards, you will be expected to code most of the algorithms yourself.

Follow the steps given below to implement the *Root Approximation* algorithm.

1) Make sure that you have understood the above algorithm.
2) Create a class called `Numeric`, which is to judge where two numbers are close enough (equal). The codes follows.
3) Test this program to make sure it works. You can do this by adding a `main` method.

```
public class Numeric
{  /**
      Tests whether two floating-point numbers are
      equal, except for a roundoff error
      x a floating-point number
      y a floating-point number
      returns true if x and y are approximately equal
   */
   public static boolean approxEqual(double x, double y)
   {
      final double EPSILON = 1E-10;
      if (Math.abs(x-y)<EPSILON)
      {
        return(true);
      }
      return(false);
   }
   // more numeric methods can be added here
}
```

4) Create a class called `RootApproximator` to implement the Heron's algorithm. The codes is as follows:

```
/**
Computes approximations to the square root of
a number, using Heron's algorithm
*/
public class RootApproximator
{
	/**
	Constructs a root approximator for a given number
	aNumber the number from which to extract the square root
	(Precondition: aNumber >= 0)
	 */
	public RootApproximator(double aNumber)
	{
		a = aNumber;
		xold = 1;
		xnew = a;
	}
	/**
	Compute a better guess from the current guess.
	returns the next guess
	 */
	public double nextGuess()
	{
		xold = xnew;
		if (xold != 0)
		{
			xnew = (xold + a / xold) / 2;
		}
		return xnew;
	}
	/**
	Compute the root by repeatedly improving the current
	guess until two successive guesses are approximately equal.
	@return the computed value for the square root
	 */
	public double getRoot()
	{
		while (!Numeric.approxEqual(xnew, xold))
		{
			nextGuess();
		}
		return xnew;
	}
	private double a; // the number whose square root is computed
	private double xnew; // the current guess
	private double xold; // the old guess
}
```

5) Test your program by adding a `main` method, creating an instance of `RootApproximator` (with a parameter) and then calling the `getRoot` method to see the result.
6) Write a simple test program by creating a class called `RootApproximationTest`. The code is as follows:

```
import javax.swing.JOptionPane;

/**
   This program computes square roots of user-supplied inputs.
*/
public class RootApproximationTest
{
   public static void main(String[] args)
   {
      boolean done = false;
      while (!done)
      {
         String input = JOptionPane.showInputDialog("Enter a number, Cancel to
quit");

         if (input == null)
         {
            done = true;
         }
         else
         {
            double x = Double.parseDouble(input);
            RootApproximator r = new RootApproximator(x);
            double y = r.getRoot();

            System.out.println("square root of " + x + " = " + y);
         }
      }
   }
}
```

Run the program and test the algorithm.

1.6 Exercise 2

Think about the following questions:

1) How to randomly generate inputs of a given size within `RootApproximationTest`?
2) How to measure the running time of an algorithm?