A Utility class

We wish to form a class that contains some useful methods for common statistical problems such as finding the mean deviation, and median of an array of doubles. We shall also experiment with the modifier class to see its' effect in an application.

We start with an empty class Utility which we have declared to be static:

```java
public static class Utility
{
}
// end class Utility
```

Compiling at this stage leads to an error message:

```
Z:\11Pic20\Lecture_8\U2>javac Utility.java
Utility.java:1: The type type Utility can't be declared static. It is already top-level, since it is a member of a package.
public static class Utility
1 error
Z:\11Pic20\Lecture_8\U2>
```

The message segment "It is already top-level" means that it is not some subclass of a larger class. So, the modifier static has to be dropped here. (You can check that if the word static is erased from the first line then this empty program does compile.)

To continue, we add a method mean(double A[]), modified by static to the class. Methods within a class can be declared static.

```java
public class Utility
{
    public static double mean(double A[])
    {
        int n = A.length;
        double sum = 0;
        for (int i = 0; i < n; i++)
            sum = sum + A[i];
        double av = sum/n;
        return av;
    } // end double mean(double A[])
}
// end class Utility
```
To check that this works we form an application to run it:

```java
public class Test {
    public static void main(String args[]) {
        double a[] = {1.2, 3.4, 4.7, 2.1, 1.1};
        String out = '
            mean ='
        System.out.println(out + Utility.mean(a));
    }
}
```

Note that we do not instantiate the class Utility in the class Test to utilize the method mean(). We simply refer to it as `Utility.mean(a)`. The output is:

```
mean =2.5
```

The deviation

Recall that if \(x_0, x_1, \ldots, x_{n-1}\) is a sequence of numbers then their mean is defined by

\[
    \text{mean} = \frac{x_0 + x_1 + \ldots + x_{n-1}}{n}
\]

then the deviation is defined by

\[
    \text{deviation} = \sqrt{\frac{(x_0 - \text{mean})^2 + (x_1 - \text{mean})^2 + \ldots + (x_{n-1} - \text{mean})^2}{n-1}}.
\]

Some books might use an \(n\), instead of an \(n-1\), in the denominator of the fraction that defines the deviation. Thus the java method for defining the deviation would be defined as below:

```java
public class Utility {
    public static double mean(double A[]) {
        int n = A.length;
        double sum = 0;
        for (int i = 0; i < n; i++)
            sum = sum + A[i];
        double av = sum/n;
        return av;
    }
}
```
The method `mean()` is called by the method `deviation()` in line 1.

The corresponding modification of `Test` is:

```java
public class Test
{
    public static void main(String args[])
    {
        double a[] = {1.2, 3.4, 4.7, 2.1, 1.1};
        String out = "\n            mean =\n";
        String out2 = "\n            deviation =\n";
        System.out.println(out + Utility.mean(a));
        System.out.println(out + Utility.deviation(a));
    }
}
```

The output here is:

```
mean =2.5
deviation =0.35
```

The median

The median of a sequence $x_0, x_1, \ldots, x_{n-1}$ of numbers is, roughly, a number $u$ such that half of the $x_i$ are above $u$ and half are below $u$.  

To determine the median we first sort the numbers into a new, increasing, sequence. That is, we re-label them as \(y_0, y_1, \ldots, y_{n-1}\) taking care that \(y_0 \leq y_1 \leq \ldots \leq y_{n-2} \leq y_{n-1}\).

Then, if \(n\) is even, say \(n = 2k\), we take \((y_{k-1} + y_k)/2\) as the median. To see this take a simple numerical example with \(n = 8\), so \(k = 4\) and we have

\[
y_0 \leq y_1 \leq y_2 \leq y_3 \leq y_4 \leq y_5 \leq y_6 \leq y_7,
\]

and half the numbers lie below \((y_3 + y_4)/2\) and half lie above this average.

If \(n\) were odd, say \(n = 2k+1\), then for \(n = 9\) and \(k = 4\) we would have

\[
y_0 \leq y_1 \leq y_2 \leq y_3 \leq y_4 \leq y_5 \leq y_6 \leq y_7 \leq y_8.
\]

In this case "half above, half below" definition is imprecise so we fudge by taking \(y_4\) as the median. In general, if \(n = 2k+1\) is odd we take \(y_k\) as the median.

So, let us go back to the problem of sorting the original sequence into an increasing sequence. We do this by the `bubblesort()`, whose code is:

```java
public static double[] bubblesort(double A[])
{
    int n = A.length;
    double b[] = new double[n];

    for (int i = 1; i < n-1; i++)
        for (int j = 0; j < n-i; j++)
            if (A[j] > A[j+1])
                { double q = A[j+1];
                  A[j] = q;
                }

    b = A;
    return b;
}
```

To see how this works we go through the loop for \(i = 1\) (and \(n = 6\)):

```java
for (int j = 0; j < n-i; j++) // i = 1, n = 6
    if (A[j] > A[j+1])
        { double q = A[j+1];
          A[j] = q;
        }
```

The general argument is then applied to the first $A[0], \ldots, A[4]$ in the $i = 2$ loop. The argument is applied recursively; each increment of $i$ is applied to shorter array.

We add `bubblesort(double A[])` to our class `Utility`

```java
public class Utility
{
    public static double mean(double A[])
    {
        int n = A.length;
        double sum = 0;
        for (int i = 0; i < n; i++)
            sum = sum + A[i];
        double av = sum/n;
        return av;
    } // end double mean(double A[])

    public static double deviation(double A[])
    {
        int n = A.length;
        double m = mean(A); // 1
        double sum = 0;
        for (int i = 0; i < n ; i++)
            sum = (A[i]-m)*(A[i]-m);
        sum = Math.sqrt(sum);
        double r = sum/(n-1);
        return r;
    } // end // end double mean(double A[])

    public static double[] bubblesort(double A[])
    {
        int n = A.length;
        double b[] = new double[n];
        for (int i = 1; i < n-1; i++)
            for (int j = 0; j < n-i; j++)
                if (A[j] > A[j+1])
                {
                    double q = A[j+1];
                    A[j] = q;
                }
        return b;
    }
}
```
Note that the return value of \texttt{bubblesort} is an array \texttt{double[]}.

To continue, we modify \texttt{Test} to check \texttt{bubblesort}.

\begin{verbatim}
public class Test
{
    public static void main(String args[])
    {
        double a[] = {1.2, 3.4, 4.7, 2.1, 1.1}; //1
        String out = "\n            mean =\n"
        String out2 = "\n            deviation =\n";
        System.out.println(out + Utility.mean(a));
        System.out.println(out2 + Utility.deviation(a));

        Utility.bubblesort(a);
        System.out.print("\nSorted array values are: ");
        for( int i = 0; i < a.length; i++)
            System.out.print(a[i] +","); //2
        System.out.println();
    }
}
\end{verbatim}

The output is:

\begin{verbatim}
mean =2.5
deviation =0.35
Sorted array values are: 1.2,1.1,2.1,3.4,4.7,
\end{verbatim}

Note that the values of the array \texttt{a[]} printed out by (//2) are not the values defined in (//1), but the values produced by \texttt{Utility.bubblesort(a)}. We shall return to this point later.
The Complete Utility class

We complete this class by adding a method for defining the median:

```java
public class Utility
{
    public static double mean(double A[])
    {
        int n = A.length;
        double sum = 0;
        for (int i = 0; i < n; i++)
            sum = sum + A[i];
        double av = sum/n;
        return av;
    } // end double mean(double A[])

    public static double deviation(double A[])
    {
        int n = A.length;
        double m = mean(A); // 1
        double sum = 0;
        for (int i = 0; i < n ; i++)
            sum = (A[i]-m)*(A[i]-m);
        sum = Math.sqrt(sum);
        double r = sum/(n-1);
        return r;
    } // end double deviation(double A[])

    public static double[] bubblesort(double A[])
    {
        int n = A.length;
        double b[] = new double[n];
        for (int i = 1; i < n-1; i++)
            for (int j = 0 ; j < n-i; j++)
                if (A[j] > A[j+1])
                {
                    double q = A[j+1];
                    A[j] = q;
                }
        b = A;
        return b;
    } // end bubblesort(double A[])

    public static double median(double A[])
    {
        int n = A.length;
        double c[] = new double[n];
        c = bubblesort(A);
        double med;
        if (n%2 == 0 ) med = (c[n/2-1]+c[n/2])/2;
        else med = c[n/2];
        return med
    } // end median(double A[])
}
```
For the sake of simplifying the code in the `main()` function, we add a `PrintOut()` routine to `Test`. Note that it, too, is static.

```java
public class Test {
    public static void main(String args[]) {
        double a[] = {1.2, 3.4, 4.7, 2.1, 1.1};
        PrintOut(a);
        double b[] = {1.2, 3.4, 4.7, 2.1, 1.1, 0.5, 4.4, 4.9};
        PrintOut(b);
    }

    public static void PrintOut(double A[]) {
        System.out.println();
        Utility.bubblesort(A);
        System.out.println("Sorted array values are: ");
        for (int i = 0; i < A.length; i++)
            System.out.print(A[i] + " ");

        String out = " mean =";
        String out2 = " deviation =";
        String out3 = " median =";
        System.out.println();
        System.out.println(out + Utility.mean(A));
        System.out.println(out2 + Utility.deviation(A));
        System.out.println(out3 + Utility.median(A));
    }
}
```

The output now is:

```
Sorted array values are:
1.2 1.1 2.1 3.4 4.7
    mean =2.5
    deviation =0.55
    median =2.1

Sorted array values are:
0.5 1.1 1.2 2.1 3.4 4.4 4.7 4.9
    mean =2.7875000000000005
    deviation =0.36178571428571427
    median =2.75
```
Non-static methods

We demonstrate on how non-static methods might be handled. To start take the class Utility and erase the modifier static from the definition of each of the four methods it contains. Then rename the resulting class Utility2.

Next make the corresponding changes in one of versions of Test, and call it Test2. The changes, as in //(1) refer to Utility2.

    public class Test2
    {
        public static void main(String args[])
        {
            double a[] = {1.2, 3.4, 4.7, 2.1, 1.1};
            String out = "\n mean =";
            String out2 = " deviation =";
            System.out.println(out + Utility2.mean(a)); //1
            System.out.println(out2 + Utility2.deviation(a));
        }
    }// end class Test2

Compiling this produces error messages:

`javac Test2.java`

`Test2.java:8: Can't make static reference to method Utility2.
    System.out.println(out + Utility2.mean(a)); //1`

`Test2.java:9: Can't make static reference to method class Utility2.
    System.out.println(out2 + Utility2.deviation(a));`

The methods in the class Utility2 can be accessed by instantiating in in Test2:

    public class Test2
    {
        public static void main(String args[])
        {
            Utility2 u2 = new Utility2();
            double a[] = {1.2, 3.4, 4.7, 2.1, 1.1};
            String out = "\n mean =";
            String out2 = " deviation =";
            System.out.println(out + u2.mean(a));
            System.out.println(out2 + u2.deviation(a));
        }
    }// end class Test2

This works:
Z:\Lecture_8\U2>javac Test2.java

Z:\Lecture_8\U2>java Test2

mean = 2.5
deviation = 0.35