Question #2

Fill in the blanks in the **averageOfMaxValues** method so that it will use the **findMax** and **findAverage** methods to calculate the average of the maximum values in each array in A; note that A is a two-dimensional array, i.e. an array of arrays.

For instance, if A = { { 3, 5, 6 } , { 2, 8, 1, 4 } , { 9, 3, 4, 5 } }, then:

- the max of A[0] is 6
- the max of A[1] is 8
- the max of A[2] is 9
- their average is (6 + 8 + 9) / 3 = 7.667.

```
   public static int findMax(int[] A) {
        if (A.length == 0) return 0;
        int max = A[0];
        for (int a : A) {
            if (a > max) max = a;
        }
        return max;
    }

    public static double findAverage(int[] A) {
        int sum = 0;
        for (int a : A) sum += a;
        return ((double)sum) / A.length;
    }

    public static double averageOfMaxValues(int[][] arr2d) {
          // this array should hold the max value of each array in arr2d
1:        int[] maxValues = new int[arr2d.length];
2:        for (int i = 0; i < A.length; i++) {
3:              maxValues[i] = findMax(arr2d[i]);
4:        }
5:        return findAverage(maxValues);
    }
```

Question #5

Write a Java method **stddev()** that calculates the standard deviation of an array ***a***. The standard deviation is defined as follows:

$$\sigma = \sqrt{((a_0 - \mu)^2 + \ (a_1 - \mu)^2 + \ldots + (a_{n-1} - \mu)^2)/(n-1)}$$

Where μ is the average of $(a_0, \ldots a_{n-1})$.

Implement the **stddev()** method so that it uses the **average()** method that is defined below.

```
public static double average (double[] arr) {
   double sum = 0;
   for (int i=0; i<arr.length; i++) {
        sum += arr[i];
   }
   return sum / arr.length;
}



public static double stddev (double[] aarr) {

   double average = average(aarr);

   double sum = 0;
   for (int i=0; i< aarr.length; i++) {
        sum += Math.pow(aarr[i] - average, 2);
   }

   return Math.sqrt(sum / (aarr.length - 1));
}
```

Question #6

The following code is attempting to use recursion to compute the summation 1+2+3+…n.

Implement the `sumHelper()` method so that it takes two arguments -- n and the accumulating `sum` -- and is defined as follows:

- sumHelper(0, x) = x
- sumHelper(n, x) = sumHelper(n-1, x+n)

```
public static int sum (int n) {
   return sumHelper(n, 0);
}

// Implement the sumHelper method here

public static int sumHelper(int n, int x) {

   if (n == 0) {
      return x;
   }
   else {
      return sumHelper(n - 1, x + n);
   }
}
```

Question #7

Write a complete Java program named **Print** that takes 2 arguments from the command line, call those two arguments "N" and "text", and then prints **text** to standard-out **N** times.

For instance, if the command line contains the following:

```
java Print 3 dog
```

the the program should produce the output:

```
dog
dog
dog
```

And if the file **words.txt** contains the following:

```
java Print 4 cat
```

Then the program should produce the output:

```
cat
cat
cat
cat
```

Your program can assume that there are at least two command line arguments and that the the first command line argument can be converted to an int. The program should not print anything if the int is zero or negative.

```
public class Print {

   public static void main(String[] args) {

      int N = Integer.parseInt(args[0]);
     String text = args[1];
     for (int i = 0; i < N; i++) {
           System.out.println(text);
      }
   }
}
```

Question #8

Assume that the following code compiles:

```
double m = StdIn.readDouble();
int x = (int)fun(m * 2);
```

Assuming that the **fun** method is defined in the same class as the code above, which of the following are possible legal implementations of the **fun** method? Select all that apply:

The correct answers are B and D.

A.

```
public static void fun(double a) {
      System.out.println("fun?" + a);
}
```

This is not a legal implementation because this method returns void; however, the code at the top of the page treats **fun** as if it has a non-void return value.

B.

```
public static double fun(double a) {
      return a * a;
}
```

This is a legal implementation. The argument that is passed (m*2) is a double, as is the parameter a. And it is possible for this method to return a double and have the caller cast it to an int.

C.

```
public static int fun(int a) {
      return -1 * a;
}
```

This is not a legal implementation because the argument that is passed (m*2) is a double, which we would not be able to store in the int parameter without casting.

D.

```
public static int fun(double a) {
      return (int)Math.round(a);
}
```

This is a legal implementation. The argument that is passed (m*2) is a double, as is the parameter a. And even though the calling code casts the return value to an int, it is okay to cast something to an int even if it's already an int!