Programming with methods, objects, and arrays

1. Methods and objects
   a. Write a method to display the size of the perimeter of a triangle. The arguments to the method will be 3 Point objects giving the vertices of the triangle. For example:

   ```java
   Point p1 = new Point(2, 1);
   Point p2 = new Point(4, 8);
   Point p3 = new Point(7, 5);
   displayPerimeterSize(p1, p2, p3);
   
   should display something like:
   
   perimeter = 6.4 + 4.2 + 7.3 = 17.9
   
   Note that this is only an example. Your method should work for any 3 Point objects, not just the ones in the example above.
   
   b. Modify your method so that instead of displaying the perimeter on the screen, it returns the value of the perimeter.
   
   c. Write a main() method that uses this new displayPerimeterSize() method to compute the perimeter of two different triangles.
   
   d. **Change your main() method to compute the perimeter of a pentagon (a 5-sided figure). To do this, imagine cutting up the Pentagon into 3 triangles, like this:
e. Write a method to move a triangle to the origin. As before, we will represent triangles as 3 Point objects, each Point object being a vertex of the triangle. To move the triangle to the origin, first pick one of the vertices that will move to the origin. Then subtract the x and y coordinates of this vertex from all of the vertices.

To start this method, first figure out the method signature (first line of the method definition, which starts “public static …”):

f. Next, decide on an algorithm, or recipe, for taking the inputs to this command (the arguments) and computing the outputs (the return value). Write down a step-by-step procedure, in English, for moving the triangle to the origin.

g. Next, translate your algorithm into code. If your method signature has a return type other than “void”, then remember to include a return statement with a value of the appropriate type.

h. Create a program whose main() method creates 3 Point objects, displays them, moves them to the origin using your method, and displays them again.

i. Notice that your method changes the contents of the Point objects even though it doesn’t (shouldn’t!) return anything. Why is that possible?

2. Methods and arrays

j. Let’s say I have an array of integers entered by the user. Write a method that displays the biggest integer in the array.

k. Write a main() method that creates an array of integers, and uses the method from (j) to find and display the biggest integer.

l. Modify your method so that it returns the biggest integer in the array, instead of displaying it.

m. Change the main() method so that it uses the new version of your biggest-integer-finder, and displays the biggest integer.
Write a method that *swaps* the values stored in positions *pos1* and *pos2* in an array. For instance, if I created an array in main memory that looked like this:

```java
int [] x = {7, -5, 12, 22, 16};
```

Then, if I execute

```java
// In this example, pos1 is 2, pos2 is 4
swap(x, 2, 4);
```

Then x should contain the values `{7, -5, 16, 22, 12}` after the swap method executes, because it has swapped the values at positions 2 and 4. Note this is only an example, and your method should work for any integer array and any 2 positions between 0 and the end of the array.

Start by coming up with the method *signature*:

```java
int swap(int [], int, int);
```

Next, come up with an *algorithm* for the swap method. It helps to create a temporary variable that stores one of the two elements being swapped.

Now fill in the body of your method by turning your algorithm into code. If your method has a non-void return type, remember to include a return statement with a value of the appropriate type.

Think about how you would write a main() method that creates an array, finds the biggest value of the array, and swaps it with the value at position 0. Before you do this, answer this: why can you NOT use the method for finding the biggest integer from above?

Modify the method for finding the biggest integer (above) so that it returns the *position* rather than *value* of the biggest integer in the array.

Now write the main() method that creates an array, finds the position of the biggest value of the array (using your modified method), and swaps the biggest value with the value stored at position 0.
**Write a insertionSort() method that sorts an array. Here is the algorithm for insertionSort:

**InsertionSort**

**Input:** an array of integers D  
**output:** D is sorted, largest element at position 0, then next-largest, and so on.

Let numSortedElements = 0;  
while numSortedElements < length of D:
    Let posNextLargest =  
        findBiggestPos(D, numSortedElements, length of D)  
    swap(D, numSortedElements, posNextLargest)  
    Let numSortedElements = numSortedElements + 1 

Note that this algorithm is only partially defined, since we made use of 2 sub-routines. We already defined the “swap” routine above. We defined a version of the findBiggestPos routine above as well, but this one’s a bit different. Here’s the algorithm for that:

**Find Biggest Position in Portion of an Array**

**Input:** an array of integers D  
    startPos, the first position to start looking  
    endPos, the last position to look at  
**Output:** the position of the biggest element in D between the positions startPos and endPos

Let posBiggest = startPos  
Let curPos = startPos + 1  
while curPos<=endPos:
    if D[curPos] > D[posBiggest]:  
        posBiggest = curPos  
    Let curPos = curPos + 1 
return posBiggest