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.
   
   ```public static void displayPerimeterSize(Point p1, Point p2, Point p3) {
   double perim = p1.distance(p2);
   perim += p2.distance(p3);
   perim += p3.distance(p1);
   System.out.println("perimeter = " + perim);
   ```
b. Modify your method so that instead of displaying the perimeter on the screen, it returns the value of the perimeter.

```java
public static double displayPerimeterSize(
    Point p1, Point p2, Point p3)
{
    double perim = p1.distance(p2);
    perim += p2.distance(p3);
    perim += p3.distance(p1);
    return perim;
}
```

c. Write a main() method that uses this new displayPerimeterSize() method to compute the perimeter of two different triangles.

```java
public static void main(String [] args)
{
    Point p1 = new Point(5, 4);
    Point p2 = new Point(4, 7);
    Point p3 = new Point(2, 9);
    double perimSize = displayPerimeterSize(p1, p2, p3);
    System.out.println(perimSize);

    p1.setLocation(2,7);
    p2.setLocation(5, 5);
    p3.translate(-2, -4);
    perimSize = displayPerimeterSize(p1, p2, p3);
    System.out.println(perimSize);
}
```
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:

```
public static void main(String [] args)
{
    Point p1 = new Point(5, 4);
    Point p2 = new Point(4, 7);
    Point p3 = new Point(2, 9);
    Point p4 = new Point(5, 10);
    Point p5 = new Point(-2, 20);
    double perimSize = displayPerimeterSize(p1, p2, p3);
    perimSize += displayPerimeterSize(p1, p3, p4);
    perimSize += displayPerimeterSize(p1, p4, p5);
    perimSize -= 2*p1.distance(p3);
    perimSize -= 2*p1.distance(p4);
    System.out.println(perimSize);
}
```
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 …”):

```java
public static void moveTriangle(Point p1, Point p2, Point p3) {
}
```

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.

Input: 3 Points, p1, p2, and p3
Let xTranslation = p1’s x coordinate
Let yTranslation = p2’s y coordinate
set p1’s x and y coordinates to 0
subtract xTranslation from p2’s x coordinate
subtract yTranslation from p2’s y coordinate
subtract xTranslation from p3’s x coordinate
subtract yTranslation from p3’s y coordinate

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.

```java
public static void moveTriangle(Point p1, Point p2, Point p3) {
    int xTrans = p1.x;
    int yTrans = p1.y;
    p1.setLocation(0, 0);
    p2.translate(-xTrans, -yTrans);
    p3.translate(-xTrans, -yTrans);
}
```
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.

```java
public static void main(String [] args)
{
    Point p1 = new Point(2, 9);
    Point p2 = new Point(3, 18);
    Point p3 = new Point(22, 150);
    System.out.println("points are " + p1 + "\n" + p2 + "\n" + p3);
    moveTriangle(p1, p2, p3);
    System.out.println("points are " + p1 + "\n" + p2 + "\n" + p3);
}
```

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?

This is possible because the Point class is an object type. Since objects are references, the method has variables that are references to the same data as the objects in the main() method. When the method changes that data, it changes the data for all references to it.
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.

```java
public static void displayBiggestInt(int[] arr)
{
    int biggest = arr[0];
    for (int i = 1; i < arr.length; i++)
    {
        if (arr[i] > biggest)
        {
            biggest = arr[i];
        }
    }
    System.out.println("biggest is " + biggest);
}
```

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

```java
public static void main(String[] args)
{
    int[] myArr = {12, 22, -17, 105, -1000, 200, -10, 26};
    displayBiggestInt(myArr);
}
```
1. Modify your method so that it *returns* the biggest integer in the array, instead of displaying it.

```java
public static int findBiggestInt(int[] arr) {
    int biggest = arr[0];
    for(int i=1; i<arr.length; i++) {
        if(arr[i] > biggest) {
            biggest = arr[i];
        }
    }
    return biggest;
}
```

m. Change the main() method so that it uses the new version of your biggest-integer-finder, and displays the biggest integer.

```java
public static void main(String[] args) {
    int[] myArr = {12, 22, -17, 105, -1000, 200, -10, 26};
    int bigVal = findBiggestInt(myArr);
    System.out.println("biggest is " + bigVal);
}
```
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
public static void swap(int [] arr, int pos1, int pos2)
{
}
```

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.

**Swap algorithm**

*Input:* array arr, position pos1, position pos2  
Let temp = arr[pos1]  
Let arr[pos1] = arr[pos2]  
Let arr[pos2] = temp

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.

```java
public static void swap(int [] arr, int pos1, int pos2)
{
    int temp = arr[pos1];
    arr[pos1] = arr[pos2];
    arr[pos2] = temp;
}
```
q. 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?

The swap method needs to know the position of the biggest value in the array, but the findBiggestInt method returns the value of the biggest value in the array.

r. 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.

```java
public static int findBiggestIntPos(int[] arr) {
    int biggestPos = 0;
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > arr[biggestPos]) {
            biggestPos = i;
        }
    }
    return biggestPos;
}
```

s. 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.

```java
public static void main(String[] args) {
    int[] myArr = {12, 22, -17, 105, -1000, 200, -10, 26};
    int bigPos = findBiggestIntPos(myArr);
    swap(myArr, bigPos, 0);
    System.out.println(Arrays.toString(myArr));
}
```
t. **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
import java.util.Arrays;

public class InsertionSorter {
    public static int findBiggestIntPos(int[] arr, int startPos, int endPos) {
        int biggestPos = startPos;
        for(int i=startPos+1; i<=endPos; i++) {
            if(arr[i] > arr[biggestPos]) {
                biggestPos = i;
            }
        }
        return biggestPos;
    }

    public static void swap(int[] arr, int pos1, int pos2) {
        int temp = arr[pos1];
        arr[pos1] = arr[pos2];
        arr[pos2] = temp;
    }

    public static void insertionSort(int[] arr) {
        int numSorted = 0;
        while(numSorted < arr.length) {
            int posBiggest = findBiggestIntPos(arr, numSorted, arr.length-1);
            swap(arr, numSorted, posBiggest);
            numSorted++;
        }
    }

    // to test our insertion sort
    public static void main(String[] args) {
        int[] test = {-5, 12, -20, 10, -15, 100, -22};
        insertionSort(test);
        System.out.println(Arrays.toString(test));
    }
}