Arrays Test 2

1. Assume that an array of integer values named `arr` has been declared and has been initialized with the following values.

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
10	20	30	40	50

Refer to the following code segment.

```
for (int k = 0; k < arr.length; k++)
{
   arr[k] = arr[k+1];
}
```

What values are contained within arr after the code segment has been executed?

(A) 20,30,40,50,50
(B) 20,30,40,50,0
(C) 10,20,30,40,50
(D) code terminates with an `IndexOutOfBoundsException`

2. Assume that an array of integer values named `arr` has been declared and has been initialized with the following values.

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
10	20	30	40	50

Refer to the following code segment.

```
for (int k = 0; k < arr.length – 1; k++)
{
   arr[k+1] = arr[k];
}
```

What values are contained within arr after the code segment has been executed?

(A) 10,10,10,10,10
(B) 10,20,30,40,50
(C) 0,10,20,30,40,50
(D) 10,10,20,30,40

3. Assume that an array of integer values named `arr` has been declared and has been initialized with the following values.

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
10	20	30	40	50

Refer to the following code segment.

```
for(int k = 1; k < arr.length; k++)
{
   arr[k-1] = arr[k];
}
```

What values are contained within arr after the code segment has been executed?

(A) 20,20,20,20,20
(B) 20,30,40,50,0
(C) 20,30,40,50,50
(D) code terminates with an `IndexOutOfBoundsException`

4. The following incomplete method is intended to return the largest integer in the array numbers.

```
// precondition: numbers.length > 0
public static int findMax(int[] numbers)
{
   int posOfMax = 0;

   for (int index = 1; index < numbers.length; index++)
   {
      if( /* condition */ )
      {
          /* statement */
      }
   }
   returns numbers[posOfMax];
}
```

Which of the following can be used to replace `/* condition */` and `/* statement */` so that `findMax` will work as intended?

	/* condition */	/* statement */
(A)	`numbers[index] > posOfMax`	`posOfMax = numbers[index];`
(B)	`numbers[index] < posOfMax`	`posOfMax = numbers[index];`
(C)	`numbers[index] > numbers[posOfMax]`	`posOfMax = numbers[index];`
(D)	`numbers[index] > numbers[posOfMax]`	`posOfMax = index;`

Questions 5-6 refer to the following method.

```
//precondition: arr.length > 0
public void mystery(int[] arr)
{
   int s1 = 0;
   int s2 = 0;

   for (int k = 0; k < arr.length; k++)
   {
       int num = arr[k];

       if ((num > 0) && (num % 2 == 0))
          s1 += num;
       else if (num < 0)
          s2 += num;
    }

    System.out.println(s1);
    System.out.println(s2);
 }
```

5. Which of the following best describes the value of `s1` output by the method mystery?

(A) The sum of all values greater than 2 in arr
(B) The sum of all positive even values in arr
(C) The sum of all positive values in arr
(D) The sum of all positive odd values in arr

6. Which of the following best describes the value of s2 output by the method mystery?

(A) The sum of all positive values in arr
(B) The sum of all negative values in arr
(C) The sum of all positive even values in arr
(D) The sum of all negative even values in arr

7. Consider the following two methods that appear within a single class.

```
public void doWhat(int[] list, int num)
{
   list = new int[5];
   num = 0;

   for (int x = 0; x < list.length; x++)
      list[x] = 0;
}

public void run()
{
   int[] nums = {10, 20, 30, 40, 50};
   int value = 100;

   doWhat(nums, value);

   for (int k = 0; k < nums.length; k++)
      System.out.print(nums[k] + " ");

   System.out.print(value);
}
```

What is printed as a result of the call `run()`?

(A) `10 20 30 40 50 100`
(B) `10 20 30 40 50 0`
(C) `0 0 0 0 0 100`
(D) `0 0 0 0 0 0`

8. Which of the following will display all of the values contained within the array `nums` without causing a run-time exception?

(A)
```
for (int k = nums.length; k >=0; k--)
      System.out.print(nums[k] + " ");
```

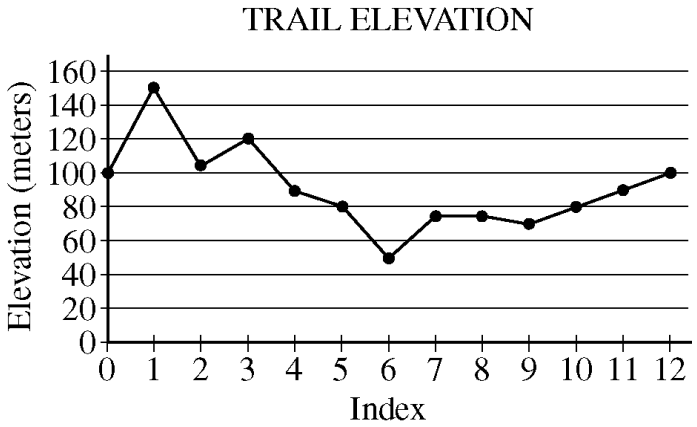
(B)
```
for (int k = 0; k < nums.length; k++)
      System.out.print(k + " ");
```

(C)
```
for (int x : nums)
      System.out.print(x + " ");
```

(D)
```
for (int x : nums)
      System.out.print(nums[x] + " ");
```

Free Response

A hiking trail has elevation markers posted at regular intervals along the trail. Elevation information about a trail can be stored in an array, where each element in the array represents the elevation at a marker. The elevation at the first marker will be stored at array index 0, the elevation at the second marker will be stored at array index 1, and so forth. Elevations between markers are ignored in this question. The graph below shows an example of trail elevations.



The table below contains the data represented in the graph.

Trail Elevation (meters)

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Elevation	**100**	**150**	**105**	**120**	**90**	**80**	**50**	**75**	**75**	**70**	**80**	**90**	**100**

The declaration of the `Trail` class is shown below. You will write two unrelated methods of the `Trail` class.

```
public class Trail
{
   /** Representation of the trail. The number of markers on the trail is markers.length.
    */
   private int[] markers;

   /** Determines if a trail segment is level. A trail segment is defined by a starting
     * marker, an ending marker, and all markers between those two markers.
     * A trail segment is level if it has a difference between the maximum elevation
     * and minimum elevation that is less than or equal to 10 meters.
     * @param start the index of the starting marker
     * @param end the index of the ending marker
     * Precondition: 0 <= start < end <= markers.length - 1
     * @return true if the difference between the maximum and minimum
     * elevation on this segment of the trail is less than or equal to 10 meters;
     * false otherwise.
     */
   public boolean isLevelTrailSegment(int start, int end)
   { /* to be implemented in part (a) */ }

   /** Determines if this trail is rated difficult. A trail is rated by counting the
     * number of changes in elevation that are at least 30 meters (up or down) between
     * two consecutive markers. A trail with 3 or more such changes is rated difficult.
     * @return true if the trail is rated difficult; false otherwise.
     */
   public boolean isDifficult()
   { /* to be implemented in part (b) */ }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

a) Write the `Trail` method `isLevelTrailSegment`. A trail segment is defined by a starting marker, an ending marker, and all markers between those two markers. The parameters of the method are the index of the starting marker and the index of the ending marker. The method will return `true` if the difference between the maximum elevation and the minimum elevation in the trail segment is less than or equal to 10 meters.

For the trail shown at the beginning of the question, the trail segment starting at marker 7 and ending at\ marker 10 has elevations ranging between 70 and 80 meters. Because the difference between 80 and 70 is equal to 10, the trail segment is considered level.

The trail segment starting at marker 2 and ending at marker 12 has elevations ranging between 50 and 120 meters. Because the difference between 120 and 50 is greater than 10, this trail segment is not considered level.

Complete method isLevelTrailSegment below.

```
/** Determines if a trail segment is level. A trail segment is defined by a
 *  starting marker, an ending marker, and all markers between those two markers.
 *  A trail segment is level if it has a difference between the maximum elevation
 *  and minimum elevation that is less than or equal to 10 meters.
 *  @param start the index of the starting marker
 *  @param end the index of the ending marker
 *  Precondition: 0 <= start < end <= markers.length - 1
 *  @return true if the difference between the maximum and minimum
 *  elevation on this segment of the trail is less than or equal to 10 meters;
 *  false otherwise.
 */
public boolean isLevelTrailSegment(int start, int end)
```

b) Write the `Trail` method `isDifficult`. A trail is rated by counting the number of changes in elevation that are at least 30 meters (up or down) between two consecutive markers. A trail with 3 or more such changes is rated difficult. The following table shows trail elevation data and the elevation changes between consecutive trail markers.

Trail Elevation (meters)

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Elevation	**100**	**150**	**105**	**120**	**90**	**80**	**50**	**75**	**75**	**70**	**80**	**90**	**100**
	\/	\/	\/	\/	\/	\/	\/	\/	\/	\/	\/	\/	
Elevation change	50	-45	15	-30	-10	-30	25	0	-5	10	10	10	

This trail is rated difficult because it has 4 changes in elevation that are 30 meters or more (between markers 0 and 1, between markers 1 and 2, between markers 3 and 4, and between markers 5 and 6).

Complete method `isDifficult` below.

```
/** Determines if this trail is difficult. A trail is rated by counting the number
 *  of changes in elevation that are at least 30 meters (up or down) between
 *  two consecutive markers. A trail with 3 or more such changes is rated
 *  difficult.
 *  @return true if the trail is rated difficult; false otherwise.
 */
public boolean isDifficult()
```