Arrays Test 1

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]
1	2	3	4	5

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) 1,2,3,4,5
(B) 0,1,2,3,4,5
(C) 1,1,2,3,4
(D) 1,1,1,1,1

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]
1	2	3	4	5

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) 2,2,2,2,2
(B) 2,3,4,5,5
(C) 2,3,4,5,0
(D) code terminates with an `IndexOutOfBoundsException`

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]
1	2	3	4	5

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) 1,2,3,4,5
(B) 2,3,4,5,5
(C) 2,3,4,5,0
(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] > numbers[posOfMax]`	`posOfMax = numbers[index];`
(B)	`numbers[index] > numbers[posOfMax]`	`posOfMax = index;`
(C)	`numbers[index] > posOfMax`	`posOfMax = numbers[index];`
(D)	`numbers[index] < posOfMax`	`posOfMax = numbers[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 positive values in arr
(B) The sum of all positive even values in arr
(C) The sum of all positive odd values in arr
(D) The sum of all values greater than 2 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 positive even values in arr
(C) The sum of all negative 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 = {1, 2, 3, 4, 5};
   int value = 6;

   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) `0 0 0 0 0 0`
(B) `0 0 0 0 0 6`
(C) `1 2 3 4 5 6`
(D) `1 2 3 4 5 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 x : nums)
      System.out.print(x + " ");
```

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

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

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

Free Response

A multiplayer game called `Token Pass` has the following rules.

Each player begins with a random number of tokens (at least 1, but no more than 10) that are placed on a linear game board. There is one position on the game board for each player. After the game board has been filled, a player is randomly chosen to begin the game. Each position on the board is numbered, starting with 0.

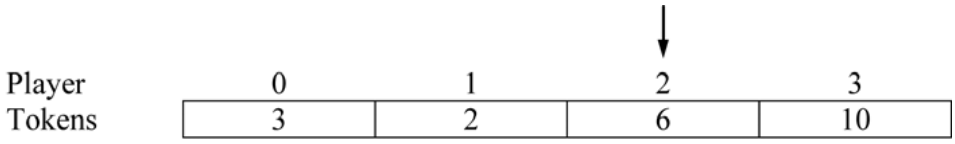
The following rules apply for a player's turn.

- The tokens are collected and removed from the game board at that player's position.
- The collected tokens are distributed one at a time, to each player, beginning with the next player in order of increasing position.
- If there are still tokens to distribute after the player at the highest position gets a token, the next token will be distributed to the player at position 0.
- The distribution of tokens continues until there are no more tokens to distribute.

The Token Pass game board is represented by an array of integers. The indexes of the array represent the player positions on the game board, and the corresponding values in the array represent the number of tokens that each player has. The following example illustrates one player's turn.

Example

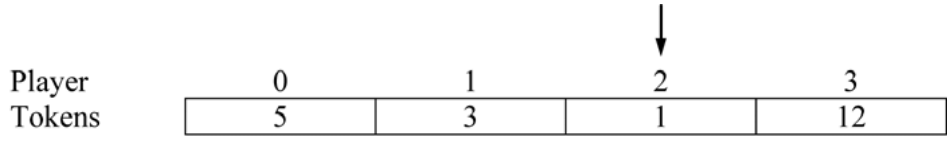
The following represents a game with 4 players. The player at position 2 was chosen to go first.



The tokens at position 2 are collected and distributed as follows.

1st token - to position 3 (The highest position is reached, so the next token goes to position 0.)
2nd token - to position 0
3rd token - to position 1
4th token - to position 2
5th token - to position 3 (The highest position is reached, so the next token goes to position 0.)
6th token - to position 0

After player 2's turn, the values in the array will be as follows.



The Token Pass game is represented by the `TokenPass` class.

```
public class TokenPass
{
   private int[] board;
   private int currentPlayer;
   /** Creates the board array to be of size playerCount and fills it with
    * random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
    * random integer value in the range between 0 and playerCount-1, inclusive.
    * @param playerCount the number of players
    */
   public TokenPass(int playerCount)
   { /* to be implemented in part (a) */ }

  /** Distributes the tokens from the current player's position one at a time to each
   *  player in the game. Distribution begins with the next position and continues
   * until all the tokens have been distributed. If there are still tokens to distribute
   * when the player at the highest position is reached, the next token will be
   * distributed to the player at position 0.
   * Precondition: the current player has at least one token.
   * Postcondition: the current player has not changed.
   */
   public void distributeCurrentPlayerTokens()
   { /* to be implemented in part (b) */ }

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

a) Write the constructor for the `TokenPass` class. The parameter `playerCount` represents the number of players in the game. The constructor should create the board array to contain `playerCount` elements and fill the array with random numbers between 1 and 10, inclusive. The constructor should also initialize the instance variable `currentPlayer` to a random number between 0 and playerCount-1, inclusive.

Complete the `TokenPass` constructor below.

```
/** Creates the board array to be of size playerCount and fills it with
 * random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
 * random integer value in the range between 0 and playerCount-1, inclusive.
 * @param playerCount the number of players
 */
 public TokenPass(int playerCount)
```

b) Write the `distributeCurrentPlayerTokens` method. The tokens are collected and removed from the game board at the current player's position. These tokens are distributed, one at a time, to each player, beginning with the next higher position, until there are no more tokens to distribute.

Class information repeated from the beginning of the question

```
public class TokenPass

private int[] board
private int currentPlayer
public TokenPass(int playerCount)
public void distributeCurrentPlayerTokens()
```

Complete method distributeCurrentPlayerTokens below.

```
/** Distributes the tokens from the current player's position one at a time to each
 *  player in the game. Distribution begins with the next position
 *  and continues until all the tokens have been distributed.
 *  If there are still tokens to distribute when the player at the
 *  highest position is reached, the next token will be distributed to the player
 *  at position 0.
 *  Precondition: the current player has at least one token.
 *  Postcondition: the current player has not changed.
*/
public void distributeCurrentPlayerTokens()
```