**Option D and General Programming**

*Revision 1*

Code to **declare** a variable of type String with an **identifier** name

```
String name;
```

Code to **declare** an array of type int with **length** of 10

```
int[] nums = new int[__];
```

Code to **iterate** through an array of any length

```
for(int i = 0; i<nums.length; i++){

   System.out.println(nums[__]);
}
```

An example of a **static** function ie it can be called without requiring an instance of an object:

```
public static _______ displayMsg(){
  System.out.println("Hello");
}
```

A class with a method simulating a number cube returning a value in the range 1-6 inclusive

```
public class NumberCube{

   //constructor method(s) not shown

   public int getToss(){
     return (int)(Math.random()*___) + ___;
   }
}
```

Code to **construct/istantiate** a NumberCube object

```
_______________ cube = new NumberCube();
```

## Sample Exam question

### PART A

The number cube is represented by the following class:

```
public class NumberCube {
      /** @return an integer value between 1 and 6, inclusive */

      public int toss() { /* implementation not shown */ }

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

Write a static method `getCubeTosses` that takes a `NumberCube` and a number of tosses as parameters.

The method should *return an array of the values produced by tossing the number cube the given number of times*.

Complete method `getCubeTosses` below.

/** Returns an array of the values obtained by tossing a number cube numTosses times.
@param `cube` a `NumberCube`
@param `numTosses` the number of tosses to be recorded

Precondition: numTosses > 0
@return an array of values
*/

```
public static int[] getCubeTosses(NumberCube cube, int numTosses){
```

**PART B**

Write a static method `countNum` that takes an array of values and a value to be counted. For example, if the array of values is:

`[4,6,1,3,4,3,4,5]`

And the value to be counted is 4, then the method will return 3.

Complete method `countNum` below.

```
/** Returns the count of a specified number in an array of numbers.
@param values an array of integers
@param numTosses the number of tosses to be recorded

Precondition: values.length > 0
@return an int stating the number of times num was found in values
 */


public static int countNum(int[] values, int num){
```

**PART C**

Write the static method `getLongestRun` that takes as its parameter an array of integer values representing a series of number cube tosses.

The method returns the starting index in the array of a run of maximum size.

A run is defined as the repeated occurrence of the same value in two or more consecutive positions in the array.

For example, the following array contains two runs of length 4, one starting at index 6 and another starting at index 14. The method may return either of those starting indexes. If there are no runs of any value, the method returns -1.

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| result | 1 | **5** | **5** | 4 | 3 | 1 | **2** | **2** | **2** | **2** | 6 | 1 | **3** | **3** | **5** | **5** | **5** | **5** |

Complete method `getLongestRun` below.

```
/** Returns the starting index of a longest run of two or more consecutive repeated values
* in the array values.
* @param values an array of integer values representing a series of number cube tosses
* Precondition: values.length > 0
* @return the starting index of a run of maximum size;
* -1 if there is no run
*/

public static int getLongestRun(int[] values) {
```

| | | |
|---|---|---|
| 4.1.1 | Identify the procedure appropriate to solving a problem. | |
| 4.1.2 | Evaluate whether the order in which activities are undertaken will result in the required outcome. | |
| 4.1.3 | Explain the role of sub-procedures/methods/functions in solving a problem. | They perform a specific task and can be called when required. |
| 4.1.4 | Identify when decision-making is required in a specified situation. | |
| 4.1.5 | Identify the decisions required for the solution to a specified problem. | |
| 4.1.6 | Identify the condition associated with a given decision in a specified problem. | You also need to understand when to use AND, OR, NOT when constructing conditions. |
| 4.1.7 | Explain the relationship between the decisions and conditions of a system. | IF…THEN…ELSE |
| 4.1.8 | Deduce logical rules for real-world situations. | Apply IF…THEN…ELSE and AND/OR/NOT to specific scenarios |
| 4.1.11 | Explain the need for pre-conditions when executing an algorithm. | A condition that must be true before a block of code or function can execute correctly |