CMSC B113 Fall 2020 Exam #1 - Solutions

Note that in some cases, there are multiple possible correct answers.

Part 1: Syntax (1 point each; 15 points total)
In the space provided, write the Java instruction that would do each of the following. Your code must be syntactically correct; no partial credit will be awarded.

(1.1) Declare, but do not initialize, an **int** variable named **k**.
int k;

(1.2) Set the value of the variable **k** (from above) to 5.
k = 5;

(1.3) Declare a double variable named **m** and initialize it to 37.8; do this as a single instruction.
double m = 37.8;

(1.4) Increment the value of the variable **m** (from above) by the value of the variable k (from above).
m = m + k; // or m += k; note that casting is not necessary since k is "promoted" to a double

(1.5) Convert the value of **m** to an **int** and then store its value in k; do this as a single instruction.
k = (int) m;

(1.6) Set the value of the variable **m** to the square root of the variable **k**.
m = Math.sqrt(k);

(1.7) Set the value of the variable **k** to be a random integer in [0, 5], i.e. between 0 and 5, inclusive.
k = (int)(Math.random() * 6);

(1.8) Assume you have **int** variables **a**, **b**, and **c**. Set the value of the variable **k** to be equal to **a** times the sum of **b** and **c**; do this as a single instruction.
k = a * (b + c); // k = a * b + c is incorrect because it would first multiply a times b

(1.9) Assume you have boolean variables **x** and **y**. Declare a **boolean** variable **n** and initialize it so that its value is true if both **x** and **y** are true, but is false otherwise; do this as a single instruction.
boolean n = x && y;

(1.10) Assume you have an **int** variable **p**. Set the value of **n** (from above) to be true if **p** is even and false if **p** is odd; do this as a single instruction.
n = p % 2 == 0;

~~(1.12) Declare and initialize a variable called **nums** as an array of 10 **int**s.~~
~~int[] nums = new int[10]; // int nums[] = new int[10] is okay, too~~

~~(1.13) Set the element at index #5 of the **nums** array (from above) to 113.~~
~~nums[5] = 113; // some students used nums[4]; that is the "fifth element" of the array, but not the one at index #5~~

~~(1.14) Set the last element of the **nums** array to be equal to 8.~~
~~nums[9] = 8; // or nums[nums.length-1] = 8 would work, too; note that nums[10] would be outside the bounds of the array~~

~~(1.15) Assume you have an array named **vals**. Set the value of the variable **k** (from above) to the number of elements in the **vals** array.~~
~~k = vals.length;~~

Part 2: Analysis

(2.1) What are the values of **x** and **y** after the following code is run, i.e. after line 18 is reached? Note that the line numbers on the left are *not* part of the program.

```
1   int x = 0, y = 1, z = 0;
2
3   if (x >= y) {
4         z = 1;
5   }
6   else if (z == 1) {
7         x = y;
8   }
9
10  if (y == 1) {
11        if (x == 0 || z == 0) {
12              y = x + z;
13        }
14        x = 2;
15  }
16  else {
17        x = 5;
18  }
```

Line 3 evaluates to false, then line 6 evaluates to false, so there is no change there.
Then 10 evaluates to true because y is 1, and 11 evaluates to true because x is 0.
Line 12 sets y to x + z = 0 + 0 = 0.
Line 14 sets x to 2.
At the end, x = 2 and y = 0

(2.2) How many times is “Hello!” printed in the following code? Note that the line numbers on the left are *not* part of the program.

```
1   int n = 5;
2
3   for (int i = 0; i < n; i++) {
4
5         for (int j = i; j < n; j++) {
6
7               System.out.println("Hello!");
8         }
9
10  }
```

- For the first iteration of the outer loop, i is 0 so the inner loop runs five times, for j ranging from 0 to 4, inclusive.
- For the second iteration of the outer loop, i is 1 so the inner loop runs four times, for j ranging from 1 to 4, inclusive.
- For the third iteration of the outer loop, i is 2 so the inner loop runs three times, for j ranging from 2 to 4, inclusive.
- For the fourth iteration of the outer loop, i is 3 so the inner loop runs two times, for j ranging from 3 to 4, inclusive.
- For the fifth iteration of the outer loop, i is 4 so the inner loop runs once, with j = 4.

At that point the outer loop stops, and “Hello!” has been printed 5+4+3+2+1 = 15 times.

(2.3) Consider the following algorithm: assuming that ints a and b have been declared and initialized:

1. declare an int called c and initialize it to 0
2. as long as a is greater than or equal to b, do the following:
    1. increment c
    2. set a to be equal to a minus b

Implement the above algorithm in Java. You do not have to write an entire program, just the part described above.

```
int c = 0;
while (a >= b) {
    c = c + 1; // or c++;
    a = a - b;
}
```

After executing this code, what is the value of c in terms of a and b? You can describe it in English or as a Java expression.

c equals a divided b (as ints), i.e. a / b

Part 3: Modifying Code

(3.1) Assume that A and B are int arrays that have been declared and initialized, and that each element in A is distinct, and that each element in B is distinct. The following code is attempting to count the number of elements that A and B have in common; note that the line numbers on the left are *not* part of the program.

```
1   int count;
2
3   for (int a : A) {
4
5         for (int b : B) {
6
7               if (a = b) {
8
9                     count += 1;
10
11              }
12        }
13  }
```

There are exactly **two** errors in this code that would cause it not to compile. In the space below, identify the line numbers on which the errors occur, and how you would fix them so that the code works as expected.

- Line 2: count should be initialized to zero, i.e. int count = 0;
- Line 7: should use double-equals to compare the values, i.e. if (a == b) {

(3.2) In the space below, rewrite the following code so that it maintains the same behavior but uses a while-loop instead of a for-loop. You can assume that the variables n, k, and vals have all been declared and initialized:

```
for (int i = 0; i <= n; i++) {
    vals[i] = Math.pow(k, i);
}
```

```
int i = 0;
while (i <= n) {
    vals[i] = Math.pow(k, i);
    i++;
}
```

(3.3) In the space below, rewrite the following code so that it maintains the same behavior but uses if-else statements instead of a switch statement. You can assume that the variable x has been declared as an int and has been initialized.

```
int a;
switch (x) {
    case 0:
        a = 1;
        break;
    case 1:
        a = 0;
        break;
    default:
        a = -1;
}
```

```
int a;
if (x == 0) {
    a = 1;
}
else if (x == 1) {
    a = 0;
}
else {
    a = -1;
}
```

Part 4: Writing Code

~~(4.1) Assume that nums is an int array that has been declared and initialized. Write code that would display the elements of nums in reverse order using System.out.println. That is, it should first display the last element, then the second-to-last, and so on. You do not need to write an entire program, just the code that performs this operation.~~

```
for (int i = nums.length - 1; i >= 0; i--) {
    System.out.println(nums[i]);
}
```

(4.2) Write a complete Java program called Average that takes one or more doubles as its runtime arguments and displays the average of the positive values, i.e. the ones that are greater than zero.

For instance:
`java Average 2.2 0.0 -4.5 3.5`
should display 2.85 since it is the average of the two positive values, 2.2 and 3.5.

Your program can assume that at least one of the runtime arguments is a positive number and that all runtime arguments represent valid doubles. You may assume that there are exactly 4 runtime arguments.

```
public class Average {
	public static void main(String[] args) {
		double sum = 0;
		int count = 0;
		for (int i = 0; i < 4; i++) {
			double value = Double.parseDouble(args[i]);
			if (value > 0) {
				sum = sum + value;
				count = count + 1;
			}
		}
		System.out.println(sum/count);
	}
}
```