CIS 110 Fall 2020 Sample Midterm 1
Answer Key

1a. The following code compiles: (True)

```
int[] arr = {1, 'a', 2};
```

1b. `System.out.println(arr[-1])` prints the last element of `arr (False)`

1c. The following code compiles: (False)

```
String[] arr = {"Hey", 1};
```

1d. It is possible to exit out of a while loop that starts like this: (True)

(i) while(true) {

1e. The following statement evaluates to true (True)

(ii) (true && (false || !false)) == (!false && true))

1f. The following code will print out 10: (False)

```
for (int i = 0; i < 11; i++) {
        // do nothing
}
System.out.println(i);
```

1g. It is possible for a for loop to be infinite (True)

1h. Given any `int x`, the following code never prints "3" (False)

```
if (x % 2 == 0) {
      System.out.println("1");
} else if (x % 2 == 1) {
      System.out.println("2");
} else {
      System.out.println("3");
}
```

1i. To check if an `int x` is divisible by both 2 and 3, you can use the following condition: (True)

```
(x % 6 == 0)
```

1j. The following code compiles (False)

```
if (true) {
       // something
} else {
       // something
} else {
       // something
}
```

1k. An int array can hold values of both type double and type int (False)

1l. This is a valid way to initialize a String array: (True)

```
String[] arr = {"", "Hello", "Students"};
```

1m. This will result in a compilation error: (False)

```
int[] arr = new int[4];
arr[4] = 3;
```

1n. The size of arrays can be increased after initialization (False)

1o. The value of int y = 14: (True)

```
int[] arr = {1, 14, 2};
int[] foo = arr;
```

```
int y = foo[1];
```

1p. This will print out the values of the array arr: (False)

```
int[] arr = new int[4];
System.out.println(arr);
```

1q. A variable must have a value manually assigned to it at the time it is declared. (False)

1r. In the following expression, x can be any of our four primitive data types (int, double, char, and boolean) and the program will compile. (False)

```
'U' + x
```

1s. In the following expression, x can have any of our four primitive data types (int, double, char, and boolean) and the program will compile. (True)

```
"U" + x
```

2a. Fill in the blank to *cast* the expression on the right hand side of the assignment operator so that this line would compile. Note that the parentheses here are part of the code, and the blank to fill is represented just by the underscores.

```
int justZero = (____) 0.0;
/* answer: int */
```

2b. Of the four primitive data types we have studied, _____(boolean) is the type with the smallest range of values. This type only has _____(two) possible values.

2c. Fill in the blank with the correct data type so that this program prints the String “CIS110” **exactly**:

```
____ x = 'A' + 2;
System.out.println(x + "IS110");
/* answer: char */
```

2d. The following snippet represents a function with most of its signature missing. Fill in the blank to specify the function’s return type, name, and input argument(s) so that the program will compile. If a variable is used but not declared in the function body, you can assume that it is an input argument to the function. The order of the input arguments does not matter. **Make sure to use good style and give the function a name that is related to its behavior.**

```
public static ________ _________________ (________ _________, ________ ________) {
   double[] copied = new double[arr.length];
   for (int i = 0; i < arr.length; i++) {
     copied[i] = arr[i] * factor;
   }
   return copied;
 }
/* answer: double[] copyAndMultiply(double[] arr, double factor) */
At least one of arr or factor should probably be doubles, although technically they
wouldn’t need to be. Also, the name is flexible and I would say that the only
important part of the name would be “multiply”
```

(3a) Jules the Jeweler

Jules decided to open up her very own Jewelry store after Michael bet her $10 she wouldn't do it. As part of a promotional sale, Jules will allow a lucky customer to participate in a contest. The customer will be shown three doors: behind one door (the winning door) is a prize (a free wedding ring), and behind the other two losing doors are Ring Pops. The customer will first randomly select one of the three doors; the host, Jules, will then reveal one of the ring pops from one of the two unchosen doors. Jules knows which door has the prize, and will never show the customer that door. The customer is then left with a choice to either switch to the only other unopened door, or to keep their original choice. The customer will then receive whatever is behind their chosen door. This problem involves determining if it is better to always switch or not, and what the probability you win is for each 'policy'.

To generalize:

If your policy is to always switch and you initially choose the wrong door, you will always win as Jules opens the other wrong door, and you switch to the winning door with the wedding ring. However, if you choose the right door initially, then you will switch to a losing door and always lose.

If your policy is to never switch, then you win if you initially select the correct door, and lose if not.

Michael, now trying to offset his losing bet with a brand new wedding ring, is writing a program to determine the probability of winning the prize for a given 'policy'.

Given an integer numberOfRuns [the number of trials you'd like to run your policy] and a boolean alwaysSwitch [an indicator of your policy], fill in the blanks so the function returns a double that is the percentage of times Michael wins. Note, this probability must be in the range of 0 to 1 (i.e. a 50% probability should be returned as the double 0.5).

```java
public static double chanceOfWin(int numberOfRuns, boolean alwaysSwitch){
        double wins = 0.0;
        for(int i = 0; i < numberOfRuns; i++){
            boolean[] doors = {false, false, false};
            doors[(int) (Math.random() * 3)] = true;
            int selected = (int) (Math.random() * 3);
            if(alwaysSwitch){
                if(doors[selected]){
                    continue;
                }

                if(!doors[selected]){
                    wins++;
                }
            } else {
                if(doors[selected]){
                    wins++;
                }
            }
        }
        return wins / numberOfRuns;
    }
```

1.) i < number of runs
2.) (int) (Math.random() * 3)
3.) (int) (Math.random() * 3)
4.) wins++;
5.) doors[selected]
6.) wins
7.) numberOfRuns

(4a) Version 1- Min Max Mean

Nick is suffering from a severe case of senioritis, and wants your help with his statistics homework. He has been given a file containing 110 integers (one on each line) and has been asked to find the minimum, maximum, and average of the numbers. Given that he is too lazy to write three functions, he is instead going to calculate all three numbers and return their product. Help him figure out the bugs in the code below assuming all of the numbers in the file are between 1 and 100 inclusive. The first one is filled in as an example.

```
public static double minMaxMean(String filename){
    int max = 0;
    int min = 0;
    double sum = 0;
    int[] nums = new int[110];
    In inStream = new In("filename");
    for (int i = 0; i < nums.length; i++) {
        nums[i] = inStream.readAll();
    }

    for(int i = nums.length - 1; i >= 0; i--){
        if(nums[i] > max){
            max += nums[i];
        }
       if(nums[i] < max){
            min = nums[i];
        }
        sum = sum + nums;
    }
   return min * max * sum;

}
```

LINE	ISSUE	CORRECT
3	Min is below the range	int min = 100;
6	filename should not be in quotes	In inStream = new In(filename);
8	inStream.readAll()	inStream.readInt()
13	Max is incorrectly summing	max = nums[i]
15	Wrong conditional	nums[i] < min
18	Cannot add array	sum+=nums[i]
20	Must divide sum by length	return min*max*sum/nums.length

(5a) Version 1: Michael's Coffee Conundrum

Michael would love to buy a coffee machine so he doesn't have to walk to Starbucks every morning. The price of the machine (in dollars) will be represented by the int variable price. Michael also has many separate piggy banks of money; the number of dollars in each piggy bank is stored in an int array named money. Michael would like to use up to 2 piggy banks to buy the machine. Michael is able to buy it if either the sum of the 2 piggy banks is equal to or greater than the price of the machine OR if a single piggy bank is greater than or equal to the price.

You are given the int array *money* and the int *price* of the machine. How many possible unique combinations of piggy banks of money can Michael come up with to buy the coffee machine?

Take the example below:
Inputs:
money = {14, 25, 33, 82, 12};
price = 57

Return Value: 6
The 6 possible combinations are: (14, 82), (25, 33), (25, 82), (33, 82), (82, 12), (82)
Note that a symmetric pair such as (25, 33) and (33, 25) is only counted once.

First, write a function that returns the number of combinations using just 1 piggy bank:

```
public static int combinationsOnePiggyBank(int[] money, int price) {
	int count = 0;


	for (int item: money) {
		if (item >= price) {
			count++;
		}
	}
	return count;
}
```

Next, write a function that returns the number of combinations using exactly 2 piggy banks:

```
public static int combinationsTwoPiggyBanks(int[] money, int price) {
      int count = 0;

      for (int i = 0; i < money.length; i++) {
            for (int j = i + 1; j < money.length; j++) {
                  if (money[i] + money[j] >= price) {
                      count++;
                  }
            }
      }
      return count;
}
```

Now, use the two functions above to write a function that returns the total number of combinations using 1 or 2 piggy banks:

```
public static int combinations(int[] money, int price) {
    return combinationsOnePiggyBank(money, price) +
combinationsTwoPiggyBanks(money, price)
}
```

(6) Version 3 - #everything

Hashtags are everywhere these days. Here at the CIS 110 course staff, we #cannot #get #enough #of #them. In our opinion, you should be able to use hashtags for more than just words and phrases. Why not hashtag almost every single letter in a String? And what's the point of having a hashtag come at the start of the tag? We think it would be more fun if you could flip them around.

Help make the CIS 110 staff's dreams come true. The new way to hashtag should be to place a "#" after every character in a String, excluding spaces and the last character. Write a function to recursively convert regular Strings to our new hashtag style. Your function **must be recursive** to receive credit for your solution.

"Now trending" -> "N#o#w# t#r#e#n#d#i#n#g"
"CIS110" -> "C#I#S#1#1#0"
"This will boost engagement" -> "T#h#i#s# w#i#l#l# b#o#o#s#t# e#n#g#a#g#e#m#e#n#t"
"" -> ""
"C" -> "C"

Remember the function str.substring(x,y) creates a substring of str, starting with the character at index x and ending with the character at index y-1. You can also use str.substring(x), which would take the substring starting at index x and going through the end of the string.

```
public static String hashtagify(String s) {
      if (s.length() <= 1) {
            return s;
      }
      if (s.charAt(0) == ' ') {
            return "" + s.charAt(0) + hashtagify(s.substring(1));
      }
      return "" + s.charAt(0) + '#' + hashtagify(s.substring(1));
}
```

(7) Version 2: Fawad Runs SRS

Fawad wants to plan a fun activity for a Sunday Review Session; the only drawback is that the activity requires the number of students present to be a perfect square. He'll have to save his activity for a week in which the number of students is a perfect square. He's super impatient, so he decides to write a program that will simulate student attendance and which weeks he might be able to play his game. He knows that he never gets more than *maxStudents* each week, so he'll randomly generate numbers between 1 and *maxStudents* and check if they're a perfect square. Some weeks he might have a perfect square but have more important material to cover, so he decides he needs to find at least *numChances* weeks that he could possibly play his game. He only has *numWeeks* left in the semester, so once he's tried that many times, if he hasn't found *numChances* perfect squares, he'll give up and take a nap.

First, help Fawad write a function to simulate how many students show up one week. This method should take in an int *maxStudents* that represents the highest possible number of students and return a random number between 1 and *maxStudents*, inclusive.

```
public static int simulateAttendance(int maxStudents) {
    return (int) (Math.random() * maxStudents) + 1;
}
```

Next, Fawad wants to write a function that checks whether an input *number* is a perfect square or not. Hint: when you take Math.sqrt() of a perfect square (x) and cast it to an int (y), x == y * y will be true. However, this will not hold for numbers that are not perfect squares.

```
public static boolean isPerfectSquare(int number) {
      int squareRootTruncated = (int) Math.sqrt(number);
      int allegedSquare = squareRootTruncated * squareRootTruncated;

      return allegedSquare == number;
}
```

Finally, we can write a function to simulate the entire semester. Implement the function below which takes in *maxStudents*, *numChances* and *numWeeks* (as described above) and prints each of the perfect squares on a new line. If you generate the student attendance *numWeeks* times and haven't found at least *numChances* perfect squares, then print "Take a nap" to the console. Once you find *numChances* perfect squares, the program should stop and print "You Win!" to the console. You can assume that *maxStudents* >= 0, *numChances* >= 0, and *numWeeks* >= *numChances*. Make sure to use the functions you have already written!

```
	public static void srsGame(int maxStudents, int numChances, int
numWeeks) {
	int count = 0;
	for (int i = 0; i < numWeeks; i++) {
		int attendance = simulateAttendance(maxStudents);
		if (isPerfectSquare(attendance)) {
			System.out.println(attendance);
			count++;
		}
		if (count >= numChances) break;
	}

	if (squaresFound < numChances) {
		System.out.println("take a nap");
	} else {
		System.out.println("You win!");
	}
	}
```