CSE142 Sample Midterm
Winter 2020

1. Expressions. For each expression in the left-hand column, indicate its value in the right-hand column. Be sure to list a constant of appropriate type (e.g., 7.0 rather than 7 for a double, Strings in quotes, true and false for booleans).

Expression	Value
`43 % 15 / 3 + 15 / 2`	_____
`6.2 * 5 / 10 + 3.5`	_____
`6 * 2.5 / 4 + (2.3 + 2.7) / 4`	_____
`"18" + 3 * 4 + (8 + 5)`	_____
`59 % 10 / (2 + 2) * 2.5 / 2`	_____

2. Parameter Mystery. Consider the following program.

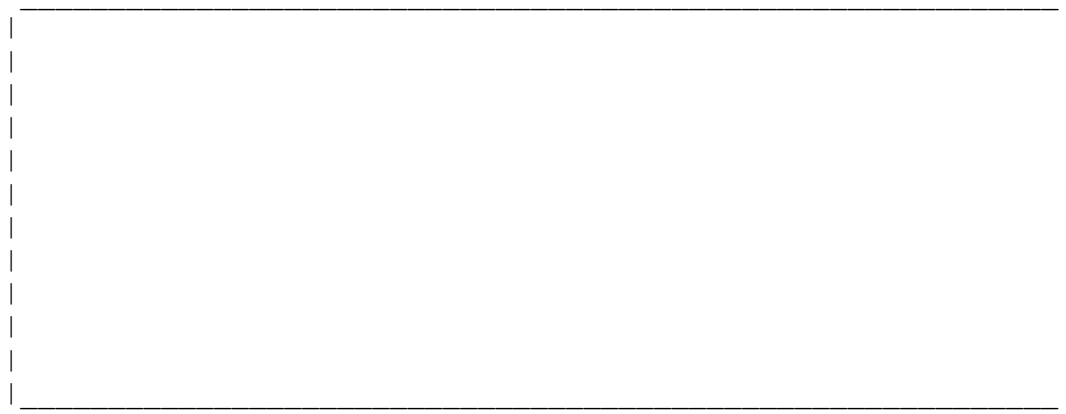
```
public class ParameterMystery {
    public static void main(String[] args) {
        String fish = "one";
        String two = "fish";
        String one = "red";
        String blue = "two";
        String red = "blue";

        mystery(two, blue, fish);
        mystery(one, two, red);
        mystery(fish, "two", one);

        fish = "blue";
        mystery("fish", fish, one);
    }

    public static void mystery(String fish, String two, String one) {
        System.out.println(one + " " + fish + ", " + two + " " + "fish");
    }
}
```

In the box below, write the output produced by this program ***exactly*** as it would appear on the console.



3. If/Else Simulation. Consider the following method.

```
public static void ifElseMystery(int a, int b) {
    int c = 5;
    if (c > a) {
        c = c - a;
    } else if (b > a) {
        b = b - a;
    }
    if (a + b > c) {
        a = a + 10;
    } else {
        b = b + 10;
    }
    System.out.println(a + " " + b + " " + c);
}
```

In each blank below, write the output produced by the method call ***exactly*** as it would appear on the console.

Method Call	Output Produced
ifElseMystery(3, 1);	_____
ifElseMystery(6, 9);	_____
ifElseMystery(5, -1);	_____
ifElseMystery(1, 2);	_____

4. While Loop Simulation. Consider the following method:

```
public static void whileMystery(int x, int y) {
    while (x > 0 && y > 0) {
        y = y - x;
        x--;
        System.out.print(y + " ");
    }

    System.out.println(y);
}
```

In each blank below, write the output produced by the method call ***exactly*** as it would appear on the console.

Method	Output Produced
mystery(5, 7);	_____
mystery(4, 20);	_____
mystery(10, 40);	_____
mystery(5, 15);	_____

5. Assertions. You will identify various conditions as being either always true, never true or sometimes true/sometimes false at various points in program execution. The comments in the method below indicate the points of interest.

```
public static int mystery(int a) {
    int b = 0;
    int c = 0;

    // Point A
    while (a != 0) {
        // Point B
        c = a % 10;

        if (c % 2 == 0) {
            b++;
        } else {
            b = 0;
            // Point C
        }

        a = a / 10;
        // Point D
    }

    // Point E
    return b;
}
```

Fill in the table below with the words ALWAYS, NEVER, or SOMETIMES.

	a != 0	c % 2 == 0	b > 0
Point A			
Point B			
Point C			
Point D			
Point E			

6. Debugging. Consider a static method called testFairCoin that takes a console Scanner as a parameter and prompts the user for a series of coin flips. The user will input one of three words: "heads" if they flip a heads, "tails" if they flip a tails, or "done" to stop entering flips. Your method should compute and output the percentage of times the user flipped heads.

Below are the interactions produced by two sample calls to testFairCoin (user input bold and underlined):

```
next flip? heads
next flip? heads
next flip? tails
next flip? tails
next flip? done
was heads 50.0% of the time
```

```
next flip? tails
next flip? heads
next flip? tails
next flip? done
was heads 33.33333333333333% of the time
```

In the log on the left, the user enters four flips: two heads and two tails, resulting in 50 percent heads. Then they enter "done" to stop entering flips. In the log on the right, the user enters three flips before quitting: one of which is heads and two of which are tails, resulting in 33. 33333333333333% heads. Notice that the coin flips are being entered by the user, not generated by the method. You may assume the user enters at least one flip before entering "done".

Below is a proposed implementation of testFairCoin:

```
public static void testFairCoin(Scanner console) {
   int heads = 0;
   int total = 0;

   System.out.print("next flip? ");
   String flip = console.next();
   while (flip.equals("done")) {
      if (flip == "heads") {
         heads++;
      }
      total++;

      System.out.print("next flip? ");
      flip = console.next();
   }

   double pct = 100 * heads / total;
   System.out.println("was heads " + pct + "% of the time");
}
```

This implementation has one or more bugs. Rewrite the implementation so that it behaves as described above. Your rewritten method should retain the same basic approach to the problem as the buggy implementation; you should not write an entirely new implementation. **Write the entire method, including your changes, as your answer.** You do not need to indicate where you made changes; simply write the final, modified version of the method.

7. Programming. Write a static method noBigger that accepts an integer parameter, max, and generates a sequence of random integers from 1 to max, inclusive. The method should continue to generate integers until it generates an integer larger than the previous one. Your method should print out each generated integer as well as the probability that the next value generated will continue the non-increasing streak. The method should return the number of values generated in the streak.

For example, suppose the following calls were made:

```
Random rand = new Random();
int streak = noBigger(100, rand);
```

These calls would produce output like the following:

```
Picking numbers from 1 to 100
Number: 93
Probability to continue: 0.93
Number: 83
Probability to continue: 0.83
Number: 60
Probability to continue: 0.6
Number: 25
Probability to continue: 0.25
Number: 17
Probability to continue: 0.17
Number: 70
Streak ends
```

After this call, the variable streak would contain the value 6. Suppose the following subsequent call is then made:

```
streak = noBigger(10, rand);
```

This call would produce output like the following:

```
Picking numbers from 1 to 10
Number: 8
Probability to continue: 0.8
Number: 6
Probability to continue: 0.6
Number: 6
Probability to continue: 0.6
Number: 4
Probability to continue: 0.4
Number: 5
Streak ends
```

After this call, the variable streak would contain the value 5.

You may assume the value passed as max is greater than 1. You are not required to round the probability. You must use a Random object to generate the values; use of Math.random() is not allowed. You must exactly reproduce the format of these logs, though the values may be different due to randomness.

8. Programming. Write a method named trackInvestment that computes the value of a financial investment over time. Once per year, the bank pays interest to the investor at a given rate. As the investment earns interest, that interest is added to the investment and the interest begins earning interest of its own (known as "compounding").

Your method should accept three parameters: a console Scanner, the initial amount of the investment in dollars (as a real number), and the number of years for which to track the investment invest (as an integer). For each year, your method should prompt the user for an integer representing that year's rate of return, as a percentage. Your method should then compute the interest earned in that year, add it to the current value of the investment, and print the new value. After the specified number of years have been computed, your method should print the total interest earned since the initial investment.

For example, suppose the following calls were made:

```
Scanner console = new Scanner(System.in);
trackInvestment(console, 100.00, 5);
```

This call would produce output like the following (user input bold and underlined):

```
Starting with: $100.0
This year's return? 10
After year 1: $110.0
This year's return? 5
After year 2: $115.5
This year's return? 20
After year 3: $138.6
This year's return? 2
After year 4: $141.372
This year's return? 3
After year 5: $145.61316
Total interest earned: $45.61316
```

Notice that the interest added each year is computed based on the value after the previous year, including all prior interest. The first year adds 10% of $100.00, bringing the total to $110.00. The second year adds 5% of $110.00, or $5.50, leading to a total of $115.50. The third year adds 20% of $115.50, or $23.10, leading to a total of $138.60.

You may assume that all parameter values passed are non-negative. You are not required to round the output; you should print the exact value computed.

9. String Manipulation Programming. Write a static method called longestStreak that takes a String, str, as a parameter and returns the largest number of times a single character appears in a row in the parameter. For example, longestStreak("abracadabra") would return 1 because no character appears multiple times in a row in "abracadabra". However, longestStreak("Gooooo!") would return 5 because there is a sequence of 5 o's in the string "Gooooo!" (Notice that the string may contain characters other than letters.)

Below are some more sample calls to longestStreak and their return values.

Method Call	Return Value
longestStreak("abcdef")	1
longestStreak("beeeeeeeees!")	9
longestStreak("bookkeeper")	2
longestStreak("Go Huskies!")	7 (spaces)
longestStreak("Baa baa blacksheep")	2
longestStreak("Aaaaaah!")	5
longestStreak("x")	1

You may assume that the string passed is not null or empty. You are limited to the methods on the cheat sheet in solving this problem.