CSE142 Sample Self Assessment 1, Summer 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).

Expression	Value
`5 * 6 - (4 + 3) * 2 - 2 * 3`	________________
`208 / 20 / 4 + 12 / 10.0 + 0.4 * 2`	________________
`8 - 2 + "8 - 2" + 8 * 2 + 8`	________________
`4 * 5 % 6 + 297 % 10 + 4 % 8`	________________
`13 / 2 * 3.0 + 5.5 * 3 / 2`	________________

2. Parameter Mystery. Consider the following program.

```
public class Mystery {
    public static void main(String[] args) {
        String x = "chair";
        String y = "notes";
        String z = "table";
        String meeting = "x";
        String notes = y + meeting;

        meeting(x, y, z);
        meeting(y, meeting, "notes");
        meeting(y + z, y + meeting, notes);
        meeting = "boring";
        meeting(meeting, "today", x);
    }

    public static void meeting(String z, String y, String x) {
        System.out.println(x + " and " + z + " like " + y);
    }
}
```

List below the output produced by this program.

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

```
public static void ifElseMystery(int a, int b) {
    if (a < b && b % 2 == 1) {
        a = a + 4;
    } else if (a % 2 == 0) {
        a++;
    }
    if (a < b || a % 2 == 0) {
        a = a + 2;
        b--;
    } else {
        a = a - 2;
        b = b + 10;
    }
    System.out.println(a + " " + b);
}
```

For each call below, indicate what output is produced.

```
Method Call               Output Produced

ifElseMystery(6, 5);      _______________

ifElseMystery(4, 6);      _______________

ifElseMystery(9, 5);      _______________

ifElseMystery(3, 6);      _______________

ifElseMystery(2, 7);      _______________

ifElseMystery(1, 3);      _______________
```

4. While Loop Simulation. Consider the following method:

```
public static void mystery(int n) {
    int x = 0;
    int y = 0;
    while (n > 0 && n % 2 == 0) {
        x++;
        y = y * 10 + n % 10;
        n = n / 10;
    }
    System.out.println(x + " " + y);
}
```

For each call below, indicate what output is produced.

```
Method Call             Output Produced

mystery(5);             _______________

mystery(8);             _______________

mystery(346);           _______________

mystery(265408);        _______________
```

5. Assertions. You will identify various assertions 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 void mystery(Scanner console) {
    int x = 3;
    int y = 0;
    // Point A
    while (x > 0) {
        // Point B
        x = console.nextInt();
        y++;
        if (x > 0) {
            x = -x;
            // Point C
        }
        // Point D
        x = -x;
    }
    // Point E
    System.out.println("y = " + y);
}
```

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

	x > 0	x == 0	y == 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 33.333333333333336% 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. 333333333333336% 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 called printSum that takes an integer n and integers low and high as parameters and that produces a series of n integers between low and high inclusive, printing various information about the integers. The method should construct and use a Random object to generate the numbers. For example, if you make the following call:

```
printSum(15, 4, 8);
```

the method should produce three lines of output like the following:

```
sum 15 numbers 4 to 8
6 + 6 + 4 + 6 + 6 + 8 + 8 + 7 + 8 + 4 + 8 + 7 + 6 + 6 + 8 = 98
max = 98
```

The first line indicates that the method is going to find the sum of 15 numbers between 4 and 8 inclusive (as indicated by the parameters in the call). The second line shows the 15 numbers randomly selected by the method and their sum. The third line reports the highest partial sum for the series. A partial sum is obtained by adding up a limited number of terms starting with the first term (in the example above, a sum of 6 for the first term, a sum of 12 for the first two terms, a sum of 16 for the first three terms, and so on). In this case, the maximum sum is the overall sum, but that won't always be the case. For example, if the call instead had been:

```
printSum(10, -2, 2);
```

the output would look like the following:

```
sum 10 numbers -2 to 2
-1 + 2 + 2 + 2 + -1 + -1 + -2 + 2 + 0 + -1 = 2
max = 5
```

Here the maximum sum occurs with the first four numbers that add up to 5. You are to exactly reproduce the format of these logs. You may assume that n is greater than or equal to 1 and that low is less than or equal to high.

8. Programming. Write a static method called tallyVotes that takes a console Scanner as a parameter and that prompts the user for a series of votes, reporting and returning the percentage of yes votes. The user is prompted for a series of responses that are either "y" to indicate yes, "n" to indicate no, or "q" to indicate that the user wants to quit because there are no more votes to tally. You can assume that the user will always type one of these three responses as a lowercase letter. For example, below is a sample call on the method:

```
Scanner console = new Scanner(System.in);
double percentYes = tallyVotes(console);
```

This call would produce an interaction like the following (with the user typing "y", "n", or "q"):

```
vote? y
vote? y
vote? n
vote? y
vote? y
vote? y
vote? q
total votes = 6
result = 83.33333333333334%
```

Notice that the user is prompted for votes until they quit and then the method produces a line of output showing the total number of votes and a line of output showing the percent of yes votes. You do not need to round the percentage. The variable percentYes would be set to 83.33333333333334. You are to exactly reproduce the format of this log. You may assume that the user enters at least one vote before quitting.

9. Programming. Write a static method called undouble that takes a string as a parameter and that returns a new string obtained by replacing every pair of repeated adjacent letters with one of that letter. For example, the String "bookkeeper" has three repeated adjacent letters ("oo", "kk", and "ee"), so undouble("bookkeeper") should return the string "bokeper". Below are more sample calls:

Method Call	Value Returned	Method Call	Value Returned
undouble("odegaard")	"odegard"	undouble("oops")	"ops"
undouble("baz")	"baz"	undouble("foobar")	"fobar"
undouble("mississippi")	"misisipi"	undouble("apple")	"aple"
undouble("carry")	"cary"	undouble("berry")	"bery"
undouble("juggle")	"jugle"	undouble("theses")	"theses"
undouble("little")	"litle"	undouble("")	""

You may assume that the string is composed entirely of lowercase letters, as in the examples above, and that no letter appears more than two times in a row. But notice that the method might be passed an empty string, in which case it returns an empty string. You may use only the string methods included on the cheat sheet. In particular, you may not use the replace method of the String class to solve this problem.