CSE 142 Sample Midterm Exam #1

1. **Expressions (15 points)**
   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`, `String`s in quotes, `true`/`false` for a `boolean`).

Expression	Value
`3 * 4 + 5 * 6 + 7 * -2`	______________________________
`1.5 * 2.0 + (5.5 / 2) + 5 / 4`	______________________________
`23 % 5 + 31 / 4 % 3 - 17 % (16 % 10)`	______________________________
`"1" + 2 + 3 + "4" + 5 * 6 + "7" + (8 + 9)`	______________________________
`345 / 10 / 3 * 55 / 5 / 6 + 10 / (5 / 2.0)`	______________________________
`1 / 2 > 0 || 4 == 9 % 5 || 1 + 1 < 1 - 1`	______________________________

2. **Parameters (15 points)**
   At the bottom of the page, write the output produced by the following program.

```
public class ParameterMystery {
    public static void main(String[] args) {
        String x = "java";
        String y = "tyler";
        String z = "tv";
        String rugby = "hamburger";
        String java = "donnie";

        hamburger(x, y, z);
        hamburger(z, x, y);
        hamburger("rugby", z, java);
        hamburger(y, rugby, "x");
        hamburger(y, y, "java");
    }

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

3. While Loop Simulation (15 points)

For each call of the method below, write the output that is printed:

```
public static void mystery(int i, int j) {
    while (i != 0 && j != 0) {
        i = i / j;
        j = (j - 1) / 2;
        System.out.print(i + " " + j + " ");
    }
    System.out.println(i);
}
```

Method Call	Output
`mystery(5, 0);`	______________________
`mystery(3, 2);`	______________________
`mystery(16, 5);`	______________________
`mystery(80, 9);`	______________________
`mystery(1600, 40);`	______________________

4. Assertions (15 points)

For the following method, identify each of the three assertions in the table below as being either ALWAYS true, NEVER true or SOMETIMES true / sometimes false at each labeled point in the code.

```
public static int mystery(int x) {
    int y = 1;
    int z = 0;

    // Point A
    while (y <= x) {
        // Point B
        y = y * 10;
        z++;

        // Point C
    }

    // Point D
    z--;

    // Point E
    return z;
}
```

	y > x	z < 0	z > 0
Point A			
Point B			
Point C			
Point D			
Point E			

5. **Programming (15 points)**
Write a static method named `hasMidpoint` that accepts three integers as parameters and returns `true` if one of the integers is the midpoint between the other two integers; that is, if one integer is exactly halfway between them. Your method should return `false` if no such midpoint relationship exists.

The integers could be passed in any order; the midpoint could be the 1st, 2nd, or 3rd. You must check all cases.

Calls such as the following should return `true` :	Calls such as the following should return `false` :
`hasMidpoint(4, 6, 8)`	`hasMidpoint(3, 1, 3)`
`hasMidpoint(2, 10, 6)`	`hasMidpoint(1, 3, 1)`
`hasMidpoint(8, 8, 8)`	`hasMidpoint(21, 9, 58)`
`hasMidpoint(25, 10, -5)`	`hasMidpoint(2, 8, 16)`

6. **Programming (15 points)**
Write a static method named `sequenceSum` that prints terms of the following mathematical sequence:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\ldots \qquad \text{(also written as } \sum_{i=1}^{\infty}\frac{1}{i} \text{)}$$

Your method should accept a real number as a parameter representing a limit, and should add and print terms of the sequence until the sum of terms meets or exceeds that limit. For example, if your method is passed 2.0, print terms until the sum of those terms is at ≥ 2.0. The following is the output from the call `sequenceSum(2.0);`

```
1 + 1/2 + 1/3 + 1/4 = 2.083333333333333
```

(Despite the fact that the terms keep getting smaller, the sequence can actually produce an arbitrarily large sum if enough terms are added.) If your method is passed a value less than 1.0, no output should be produced. You must match the output format shown exactly; note the spaces and pluses separating neighboring terms. Other sample calls:

Calls	`sequenceSum(0.0);`	`sequenceSum(1.0);`	`sequenceSum(1.5);`
Output		`1 = 1.0`	`1 + 1/2 = 1.5`

Call	`sequenceSum(2.7);`
Output	`1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7 + 1/8 = 2.7178571428571425`

7. **Programming (10 points)**
Write a static method named `favoriteLetter` that accepts two parameters: a `Scanner` for the console, and a favorite letter represented as a one-letter `String`. The method repeatedly prompts the user until two consecutive words are entered that start with that letter. The method then prints a message showing the last word typed.

You may assume that the user will type a single-word response to each prompt. Your code should be case-sensitive; for example, if the favorite letter is **a**, you should not stop prompting if the user types words that start with an **A**. For example, the following logs represent the output from two calls to your method: (User input is underlined.)

Call

```
Scanner console = new Scanner(System.in);
favoriteLetter(console, "y");
```

Output

```
Looking for two "y" words in a row.
Type a word: hi
Type a word: bye
Type a word: yes
Type a word: what?
Type a word: yellow
Type a word: yippee
"y" is for "yippee"
```

Call

```
Scanner console = new Scanner(System.in);
favoriteLetter(console, "A");
```

Output

```
Looking for two "A" words in a row.
Type a word: I
Type a word: love
Type a word: CSE142!
Type a word: AND
Type a word: PROGRAMS
Type a word: are
Type a word: always
Type a word: Absolutely
Type a word: Awesome
"A" is for "Awesome"
```