CSE 142 Sample Midterm Exam #3

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).

Expression	Value
`1 + 2 * 3 - 4 * 5`	______________________
`5 / 2 + 9.0 / 2.0 - 2 * 1.25`	______________________
`29 % 2 % 5 + 34 % 3`	______________________
`8 + 6 * -2 + 4 + "0" + (2 + 5)`	______________________
`31 / 2 / 10.0 + 10 / (5 / 2.0)`	______________________
`(1 != 2) != (2 != 3)`	______________________

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 a = "felt";
        String b = "saw";
        String c = "drew";
        String saw = "sue";
        String drew = "b";

        mystery(a, b, c);
        mystery(b, a, saw);
        mystery(drew, c, saw);
        mystery("a", saw, drew);
        mystery(a, a, "drew");
    }

    public static void mystery(String b, String a, String c) {
        System.out.println(c + " " + a + " the " + b);
    }
}
```

3. **While Loop Simulation (15 points)**
For each call of the method below, write the value that is returned:

```
public static void mystery(int a, int b) {
    while (b != 0) {
        if (a > b) {
            System.out.print(a + " ");
            a = a - b;
        } else {
            System.out.print(b + " ");
            b = b - a;
        }
    }
    System.out.println(a);
}
```

Method Call	Output
`mystery(42, 0);`	______________________
`mystery(6, 12);`	______________________
`mystery(18, 27);`	______________________
`mystery(24, 60);`	______________________
`mystery(50, 15);`	______________________

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 assertions(int n) {
    int x = 2;

    // Point A
    while (x < n) {
        // Point B
        if (n % x == 0) {
            n = n / x;
            x = 2;
            // Point C
        } else {
            x++;
            // Point D
        }
    }

    // Point E
    return n;
}
```

	`x > 2`	`x < n`	`n % x == 0`
Point A			
Point B			
Point C			
Point D			
Point E			

5. **Programming (15 points)**
Write a static method named `enoughTimeForLunch` that accepts four integers *hour1*, *minute1*, *hour2*, and *minute2* as parameters. Each pair of parameters represents a time on the 24-hour clock (for example, 1:36 PM would be represented as 13 and 36). The method should return `true` if the gap between the two times is long enough to eat lunch: that is, if the second time is at least 45 minutes after the first time. Otherwise the method should return `false`.

You may assume that all parameter values are valid: the hours are both between 0 and 23, and the minute parameters are between 0 and 59. You may also assume that both times represent times in the same day, e.g. the first time won't represent a time today while the second time represents a time tomorrow. Note that the second time might be earlier than the first time; in such a case, your method should return `false`.

Here are some example calls to your method and their expected return results:

Call	**Value Returned**
`enoughTimeForLunch(11, 00, 11, 59)`	`true`
`enoughTimeForLunch(12, 30, 13, 00)`	`false`
`enoughTimeForLunch(12, 30, 13, 15)`	`true`
`enoughTimeForLunch(14, 20, 17, 02)`	`true`
`enoughTimeForLunch(12, 30, 9, 30)`	`false`
`enoughTimeForLunch(12, 00, 11, 55)`	`false`

6. **Programming (15 points)**
Write a static method named `printGrid` that accepts two integer parameters *rows* and *cols*. The output is a comma-separated grid of numbers where the first parameter (*rows*) represents the number of rows of the grid and the second parameter (*cols*) represents the number of columns. The numbers count up from 1 to (*rows* x *cols*). The output are displayed in column-major order, meaning that the numbers shown increase sequentially down each column and wrap to the top of the next column to the right once the bottom of the current column is reached.

Assume that *rows* and *cols* are greater than 0. Here are some example calls to your method and their expected results:

Call	`printGrid(3, 6);`	`printGrid(5, 3);`	`printGrid(4, 1);`	`printGrid(1, 3);`

Output	`1, 4, 7, 10, 13, 16`
	`2, 5, 8, 11, 14, 17`
	`3, 6, 9, 12, 15, 18`
		`1, 6, 11`
		`2, 7, 12`
		`3, 8, 13`
		`4, 9, 14`
		`5, 10, 15`
			`1`
			`2`
			`3`
			`4`
				`1, 2, 3`

7. **Programming (10 points)**
Write a static method named `countEvenDigits` that accepts an integer as its parameter and returns the number of even-valued digits in that number. An even-valued digit is either 0, 2, 4, 6, or 8.

For example, the number 8546587 has four even digits (the two 8s, the 4, and the 6),
so the call `countEvenDigits(8346387)` should return `4`.

You may assume that the value passed to your method is non-negative.