CSE 142, Winter 2007
Midterm Exam, Monday, February 12, 2007

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

Rules:

- You have 50 minutes to complete this exam.
  You may receive a deduction if you keep working after the instructor calls for papers.
- This test is open-book/notes.
- You may not use any computing devices of any kind including calculators.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- You do not need to write any `import` statements in your exam code.
- Please do not abbreviate any code on your exam, such as writing `S.o.p` for `System.out.println`.
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.
- You must show your Student ID to a TA or instructor for your submitted exam to be accepted.

Good luck!

Score summary: (for grader use only)

Problem	Description	Earned	Max
1	Expressions		10
2	Parameter Mystery		20
3	While Loop Simulation		15
4	Assertions		15
5	Programming		15
6	Programming		15
7	Programming		10
X	Extra Credit		+1
TOTAL	**Total Points**		**100**

1. Expressions (10 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)`	_____

2. Parameter Mystery (20 points)

At the bottom of the page, write the output produced by the following program, as it would appear on the console.

```
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 below to the following method, write the output that is printed, as it would appear on the console:

```
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 each of the five points labeled by comments, identify each of the following assertions as being either always true, never true or sometimes true / sometimes false.

```
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;
}
```

Fill in each box of the the table below with one of the following words: ALWAYS, NEVER or SOMETIMES. (You may abbreviate these choices as A, N, and S as long as you write legibly.)

	`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.

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`

You may assume that both parameters passed to your method are greater than 0.

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 of input as their response to each prompt. Your code should be case-sensitive; for example, if the favorite letter is lowercase a, you should not stop prompting if the user types words that start with an uppercase 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");`	`Scanner console = new Scanner(System.in);` `favoriteLetter(console, "A");`

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"`
		`Looking for two "A" words in a row.`
		`Type a word: I`
		`Type a word: love`
		`Type a word: CSE142`
		`Type a word: !!!`
		`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"`

X. Extra Credit (+1 extra credit point)

If you had to describe CSE 142 in one word, what would it be? Write the word below. If there isn't a perfect word to describe 142, make up your own word and write its definition next to it. If you can't think of any word, you may draw a picture that describes your feelings about this class.

(You will get the +1 point as long as you write any word(s) or draw anything recognizable on this page.)