CSE 142, Autumn 2008
Midterm Exam, Friday, October 31, 2008

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

- You have 50 minutes to complete this exam.
  You may receive a deduction if you keep working after the instructor calls for papers.
- This exam is open-book/notes. You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.

  The only abbreviations that *are* allowed for this exam are:
  - `S.o.p` for `System.out.print`,
  - `S.o.pln` for `System.out.println`, and
  - `S.o.pf` for `System.out.printf`.
- You do not need to write `import` statements in your code.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.

Good luck!

Score summary: (for grader only)

Problem	Description	Earned	Max
1	Expressions		12
2	Parameter Mystery		12
3	If/Else Simulation		12
4	While Loop Simulation		15
5	Assertions		15
6	Programming		20
7	Programming		14
X	Extra Credit		+1
TOTAL	**Total Points**		**100**

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 and capitalization.
e.g., `7` for an `int`, `7.0` for a `double`, `"hello"` for a `String`, `true` or `false` for a `boolean`.

Expression	Value
`-(6 + 3 - 2 * 3)`	_____
`15 % 6 + 5 % 5 + 12 % 7 % 3`	_____
`9 / 2 / 2.0 + 9 / 2.0 / 2`	_____
`5 + 6 + "7" + 8 + 9 + 2 * 3`	_____
`!(3 < 7 && -1 != 8)`	_____

2. Parameter Mystery

At the bottom of the page, write the output produced by the following program, as it would appear on the console.
(Though the program uses words related to arithmetic, the output does not necessarily follow the rules of addition.)

```
public class ParameterMystery {
    public static void main(String[] args) {
        String i = "j";
        int j = -1;
        int k = 2;
        String x = "5";
        int y = 7;

        silly(k, i, j);
        silly(y, x, k);
        silly(k, "y", 4);
        silly(j + 1, x + 1, j);
    }

    public static void silly(int k, String i, int j) {
        System.out.println(j + " + " + k + " + " + i);
    }
}
```

3. If/Else Simulation

For each call below to the following method, write the output that is produced, as it would appear on the console:

```
public static void ifElseMystery(int a, int b) {
    if (a < b) {
        a = a * 2;
    }
    if (a > b) {
        a = a - 10;
    } else {
        b++;
    }

    System.out.println(a + " " + b);
}
```

Method Call	Output
`ifElseMystery(10, 3);`	_____
`ifElseMystery(6, 6);`	_____
`ifElseMystery(3, 4);`	_____
`ifElseMystery(4, 20);`	_____

4. While Loop Simulation

For each call below to the following method, write the output that is produced, as it would appear on the console:

```
public static void whileMystery(int x, int y, int z) {
    while (x < z || y < z) {
        x = x + y;
        y = y * 2;
        System.out.print(x + " " + y + " ");
    }

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

Method Call	Output
`whileMystery(0, 5, 10);`	_____
`whileMystery(4, 2, 12);`	_____
`whileMystery(10, 1, 14);`	_____

5. Assertions

For each of the five points labeled by comments, identify each of the assertions in the table below as either being *always* true, *never* true, or *sometimes* true / sometimes false.

```
public static int randomWalk(int steps) {
    Random rand = new Random();
    int x = 0;
    int r = rand.nextInt(2);

    // Point A
    while (steps > 0 || x == 0) {
        // Point B
        if (r == 0) {
            x++;
            // Point C
        } else {
            x = 0;
            // Point D
        }

        steps--;
        r = rand.nextInt(2);
    }

    // Point E
    return x;
}
```

Fill in each box below with one of ALWAYS, NEVER or SOMETIMES. (You may abbreviate them as A, N, or S.)

	`steps > 0`	`x > 0`	`r > 0`
Point A			
Point B			
Point C			
Point D			
Point E			

6. Programming

Write a static method `hopscotch` that shows a sequence of numbers similar to the appearance of a hopscotch board. (Hopscotch is a "hopping" game played by children.) For this problem, a hopscotch board is an increasing sequence of numbers in lines in an alternating pattern. Odd lines contain a single number indented by 3 spaces. Even lines contain a pair of numbers with the first number not indented, followed by 5 spaces, followed by the second number.

Your method should accept a parameter representing the number of "hops" worth of numbers to display. A board of 0 "hops" shows only the number 1, indented by 3 spaces. A "hop" is defined as a trio of additional numbers spread across 2 additional lines, following the pattern described previously. So for example, 1 hop means to show the numbers 2-3-4 to the board in the pattern shown below. 2 hops means to show the numbers 2-3-4, 5-6-7 on the board. 3 hops means to show 2-3-4, 5-6-7, 8-9-10, and so on. Assume that the number of hops passed will be at least 0.

Here are some example calls to the method and their resulting console output:

Call	`hopscotch(0);`	`hopscotch(1);`	`hopscotch(2);`	`hopscotch(3);`	`hopscotch(5);`

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

7. Programming

Write a static method named `containsBothDigits` that accepts three integer parameters *a*, *b*, and *c*. Assume that *a* is a positive integer (greater than 0) and that *b* and *c* are single-digit numbers from 0-9 inclusive. Your method should return `true` if *a* contains both *b* and *c* at among its digits, and `false` otherwise.

For example, the number 433407 contains 4, 3, 0, and 7 as its unique digit values, so a call to your method of `containsBothDigits(433407, 7, 3)` should return `true`. If *b* and *c* are the same number, your method should return `true` if that digit appears at least once among the digits; in other words, the same digit from *a* could match both *b* and *c*. So for example, a call of `containsBothDigits(433407, 7, 7)` should return `true`. Note that the digit *b* or *c* might appear more than once in *a*, as with 433407 containing two 3s.

The following table lists some additional calls to your method and their expected return values:

Call	Value Returned	Reason
`containsBothDigits(12345, 2, 5)`	`true`	12345 contains the digits 2 and 5
`containsBothDigits(3004955, 3, 0)`	`true`	3004955 contains the digits 3 and 0
`containsBothDigits(1650, 6, 6)`	`true`	1650 contains the digit 6
`containsBothDigits(12198, 1, 7)`	`false`	12198 does not contain the digit 7
`containsBothDigits(42, 9, 2)`	`false`	42 does not contain the digit 9
`containsBothDigits(904487, 2, 6)`	`false`	904487 does not contain the digit 2 or 6
`containsBothDigits(8, 2, 3)`	`false`	8 does not contain the digit 2 or 3

Note: You may not use a `String` to solve this problem.

X. Extra Credit (+1 point)

Describe CSE 142 in two words or less.

(Any word you write will get the +1 extra point.)