CSE 142, Summer 2009
Midterm Exam, Monday, July 27, 2009

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

- You have 60 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 `S.o.p` for `System.out.print`, "ditto" marks or dot-dot-dot ... marks.
- 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		10
2	Parameter Mystery		12
3	If/Else Simulation		12
4	While Loop Simulation		12
5	Assertions		15
6	Programming		15
7	Programming		15
8	Programming		9
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 and capitalization.
e.g., `7` for an `int`, `7.0` for a `double`, `"hello"` for a `String`

Expression	Value
`3 + 3 * 8 - 2`	`25`
`109 % 100 / 2 + 3 * 3 / 2.0`	`8.5`
`1 - 3 / 6 * 2.0 + 14 / 5`	`3.0`
`1 + "x" + 11 / 10 + " is" + 10 / 2`	`"1x1 is5"`
`10 % 8 * 10 % 8 * 10 % 8`	`0`

2. Parameter Mystery (12 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 soda = "coke";
        String pop = "pepsi";
        String coke = "pop";
        String pepsi = "soda";
        String say = pop;

        carbonated(coke, soda, pop);
        carbonated(pop, pepsi, pepsi);
        carbonated("pop", pop, "koolaid");
        carbonated(say, "say", pop);
    }

    public static void carbonated(String coke, String soda, String pop) {
        System.out.println("say " + soda + " not " + pop + " or " + coke);
    }
}
```

```
say coke not pepsi or pop
say soda not soda or pepsi
say pepsi not koolaid or pop
say say not pepsi or pepsi
```

3. If/Else Simulation (12 points)

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 * 2 < b) {
        a = a * 3;
    }
    if (b < a) {
        b++;
    } else {
        a--;
    }

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

Method Call	Output
`ifElseMystery(10, 2);`	`10 3`
`ifElseMystery(3, 8);`	`9 9`
`ifElseMystery(4, 4);`	`3 4`
`ifElseMystery(10, 30);`	`29 30`

4. While Loop Simulation (12 points)

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 = 0;

    while (x < y && z < 4) {
        x = x * 2;
        y = y + 2;
        z++;
    }

    System.out.println(x + " " + y + " " + z);
}
```

Method Call	Output
`whileMystery(4, 3);`	`4 3 0`
`whileMystery(5, 7);`	`10 9 1`
`whileMystery(3, 18);`	`24 24 3`
`whileMystery(0, 4);`	`0 12 4`

5. Assertions (15 points)

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 antCrawl(int max) {
    Random rand = new Random();
    int height = 0;
    int falls = 0;

    // Point A
    while (height < max) {
        int r = rand.nextInt(4);
        // Point B
        if (r == 0 && height > 0) {
            height--;
            falls++;
            // Point C
        } else {
            height++;
            // Point D
        }
    }

    // Point E
    return falls;
}
```

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

	`falls == 0`	`height > 0`	`height < max`
Point A	A	N	S
Point B	S	S	A
Point C	N	S	A
Point D	S	A	S
Point E	S	S	N

6. Programming (15 points)

Write a static method `closerDistance` that takes two pairs of integers, x_1 and y_1 and x_2 and y_2, representing two ordered pairs (x_1, y_1) and (x_2, y_2) on an x-y plane. Your method should calculate each pair's the distance from the origin (0, 0) and return the closer of the two distances as a real number. If the two points are the same distance from the origin, you may return either of the two distances, since they are equal.

For example, the point (12, 5) has a distance of 13.0 from the origin, and the point (9, 9) has a distance of 12.727922061357855 from the origin, so a call to `closerDistance(12, 5, 9, 9)` would return 12.727922061357855. Notice your method should ***not*** do any rounding.

Recall that formula to find the distance between a point (x, y) and the origin is given by the following formula:

$$Distance\ from\ origin = \sqrt{x^2 + y^2}$$

```
public static double closerDistance(int x1, int y1, int x2, int y2) {
    double dist1 = Math.sqrt(x1 * x1 + y1 * y1);
    double dist2 = Math.sqrt(x2 * x2 + y2 * y2);

    if (dist1 < dist2) {
        return dist1;
    } else {
        return dist2;
    }
}
```

7. Programming (15 points)

Write a static method named `smallest2` that accepts a `Scanner` for console input as a parameter. The method repeatedly prompts the user for a sequence of integers until the user enters a negative number. The method then prints the smallest two nonnegative numbers entered by the user. (You may assume the user will enter at least 2 nonnegative numbers.) Notice you do ***not*** have to print the two smallest *unique* numbers entered by the user. For example, if the user enters 2, 2, and 3, the two smallest numbers entered are 2 and 2.

Here are some example calls to the method and their resulting console output (user input is bolded and underlined). Assume a `Scanner` named `console` was initialized earlier in the code before each method call.

Call	`smallest2(console);`	`smallest2(console);`	`smallest2(console);`	`smallest2(console);`
Output	number? **8** number? **10** number? **2** number? **1** number? **22** number? **-1** smallest: 1 second smallest: 2	number? **5** number? **6** number? **7** number? **8** number? **9** number? **-5** smallest: 5 second smallest: 6	number? **5** number? **5** number? **5** number? **5** number? **-3** smallest: 5 second smallest: 5	number? **200** number? **100** number? **-103** smallest: 100 second smallest: 200

Hint: If you are stumped, first write the code to keep track of and print out just the smallest number entered by the user. This code alone will get substantial partial credit.

```
public static void smallest2(Scanner console) {
   System.out.print("number? ");
   int num1 = console.nextInt();
   System.out.print("number? ");
   int num2 = console.nextInt();
   int smallest = Math.min(num1, num2);
   int secondSmallest = Math.max(num1, num2);

   int num = secondSmallest;
   while (num >= 0) {
       if (num < smallest) {
           secondSmallest = smallest;
           smallest = num;
       } else if (num < secondSmallest) {
           secondSmallest = num;
       }
       System.out.print("number? ");
       num = console.nextInt();
   }

   System.out.println("smallest: " + smallest);
   System.out.println("second smallest: " + secondSmallest);
}
```

8. Programming (9 points)

Write a method called `printSquare` that takes in two integer parameters, a *min* and a *max*, and prints the numbers in the range from *min* to *max* inclusive in a square pattern. The square pattern is easier to understand by example than by explanation, so take a look at the sample method calls and their resulting console output in the table below.

Call	`printSquare(1, 5);`	`printSquare(3, 9);`	`printSquare(0, 3);`	`printSquare(5, 5);`
Output	12345 23451 34512 45123 51234	3456789 4567893 5678934 6789345 7893456 8934567 9345678	0123 1230 2301 3012	5

Each line of the square consists of a circular sequence of increasing integers between *min* and *max*. Each line prints a different permutation of this sequence. The first line begins with *min*, the second line begins with *min* + 1, and so on. When the sequence in any line reaches *max*, it "wraps around" back to *min*.

You may assume the caller of the method will pass a *min* and a *max* parameter such that *min* <= *max*.

For a maximum of 4 points, you may instead write a different method called `printSquareLite` that takes only one integer parameter representing the *max* number in the range and prints the numbers in the range from 0 to *max* inclusive in the same square pattern described above. The third column of output in the table above produces the same output as the call `printSquareLite(3)`.

```
public static void printSquare(int min, int max) {
    int range = max - min + 1;
    for (int i = 0; i < range; i++) {
        for (int j = 0; j < range; j++) {
            System.out.print((j + i) % range + min);
        }
        System.out.println();
    }
}

// one of many alternate solutions
public static void printSquare(int min, int max) {
    int range = max - min + 1;
    for (int i = 0; i < range; i++) {
        for (int j = min + i; j <= max; j++) {
            System.out.print(j);
        }
        for (int j = min; j < min + i; j++) {
            System.out.print(j);
        }
        System.out.println();
    }
}

public static void printSquareLite(int max) {
    for (int i = 0; i <= max; i++) {
        for (int j = 0; j <= max; j++) {
            System.out.print((j + i) % (max + 1));
        }
        System.out.println();
    }
}
```

X. Extra Credit (+1 point)

Describe CSE 142 or your TA in two words or less.

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