CSE 142, Autumn 2007
Midterm Exam, Friday, November 2, 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		15
2	Parameter Mystery		20
3	While Loop Simulation		15
4	Assertions		15
5	Programming		20
6	Programming		15
TOTAL	**Total Points**		**100**

NOTE: There is *not* an extra credit question on this exam.

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
`3 + 4 * 5 / 2`	_____
`13 % 5 + 43 % (11 % 3)`	_____
`1.5 * 3.0 + 25.0 / 10.0`	_____
`5 / 2 + 123 / 10 / 10.0`	_____
`5 + 2 + "(1 + 1)" + 4 + 2 * 3`	_____

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 p = "cause";
        String q = "support";
        String r = "troops";
        String support = "hillary";
        String cause = "rudy";

        troops(p, q, r);
        troops(q, r, p);
        troops(support, p, cause);
        troops(r, "p", support);
        troops(q, "cause", q);
    }

    public static void troops(String r, String p, String q) {
        System.out.println(q + " gave " + r + " to the " + p);
    }
}
```

3. While Loop Simulation (15 points)

For each call below to the following method, write the value that is returned:

```
public static int mystery(int x) {
    int a = 1;
    int c = 0;
    while (x > 0) {
        a = x % 2;
        if (a == 1) {
            c++;
        }
        x = x / 2;
    }
    return c;
}
```

Method Call	Value Returned
`mystery(2);`	_____
`mystery(-1);`	_____
`mystery(7);`	_____
`mystery(18);`	_____
`mystery(43);`	_____

4. Assertions (15 points)

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

```
// Counts number of "coin tosses" until we get heads 3 times in a row.
public static int threeHeads() {
    Random rand = new Random();
    int flip = 1;
    int heads = 0;
    int count = 0;

    // Point A
    while (heads < 3) {
        // Point B
        flip = rand.nextInt(2);   // flip coin
        if (flip == 0) {  // heads
            heads++;
            // Point C
        } else {          // tails
            // Point D
            heads = 0;
        }
        count++;
    }

    // Point E
    return count;
}
```

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.)

	`flip == 0`	`heads == 0`	`flip > heads`
Point A			
Point B			
Point C			
Point D			
Point E			

5. Programming (20 points)

Write a static method named `monthApart` that accepts four integer parameters *month1*, *day1*, *month2*, and *day2*, representing two calendar dates. Each date consists of a month (1 through 12) and a day (1 through the number of days in that month [28-31]). The method returns `true` if the dates are at least a month apart and `false` otherwise.

Dates are defined to be exactly a month apart if their day values are the same and their months differ by 1. For example, January 20 and February 20 are exactly one month apart, and July 16 and August 16 are exactly one month apart. Note that it does not matter exactly how many days the given months have.

For example, the following dates are all considered to be at least a month apart from 9/19 (September 19): 2/14, 7/25, 8/2, 8/19, 10/19, 10/20, and 11/5. The following dates are NOT at least a month apart from 9/19: 8/20, 9/1, 9/25, 10/1, 10/15, and 10/18. Note that the first date could come before or after, or be the same as, the second date. Assume that all parameter values passed are valid.

Sample calls:

```
monthApart( 6, 14,  9, 21) should return true,   because June 14 is at least a month before September 21
monthApart( 4,  5,  5, 15) should return true,   because April 5 is at least a month before May 15
monthApart( 4, 15,  5, 15) should return true,   because April 15 is at least a month before May 15
monthApart( 4, 16,  5, 15) should return false,  because April 16 isn't at least a month apart from May 15
monthApart( 6, 14,  6,  8) should return false,  because June 14 isn't at least a month apart from June 8
monthApart( 7,  7,  6,  8) should return false,  because July 7 isn't at least a month apart from June 8
monthApart( 7,  8,  6,  8) should return true,   because July 8 is at least a month after June 8
monthApart(10, 14,  7, 15) should return true,   because October 14 is at least a month after July 15
```

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 maximum limit, and should repeatedly add and print terms of the sequence until the overall sum of terms meets or exceeds the maximum limit that was passed. For example, if your method is passed the value of 2.0, you should print terms of the sequence until the sum of all those terms is at least 2.0. The following output would result 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 any value less than 1.0, no output should be produced. You must match the format of the output 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`