

CIS 110 — Introduction To Computer Programming

October 5th, 2011 — Exam 1

Answer key

Scores:

1	
2	
3	
4	
5	
6	
Total (100 max)	

## CIS 110 Exam 1 Instructions

- You have 50 minutes to finish this exam. Time will begin when called by a proctor and end precisely 50 minutes after that time. If you continue writing after the time is called, you will receive a zero for the exam.
- This exam is *closed-book*, *closed-notes*, and *closed-computational devices* except for a one page sheet (8.5" by 11") of double-sided notes.
- When writing code, the only abbreviations you may use are for `System.out.println` and `System.out.print` as follows:

`System.out.println` → S.O.PLN

`System.out.print` → S.O.P

Otherwise all code must be written out as normal.

- Please do not separate the pages of the exam. If a page becomes loose, please make sure to write your name on it so that we don't lose it, and use the provided staplers to reattach the sheet when you turn in your exam.
- If you require extra paper, please use the backs of the exam pages or the extra sheet paper provided at the end of the exam. Clearly indicate on the question page where the graders can find the remainder of your work (e.g., "back of page" or "on extra sheet").
- If you have any questions, please raise your hand and an exam proctor will come to answer them.
- When you turn in your exam, you will be required to show ID. If you forgot to bring your ID, please talk to an exam proctor immediately.

*Good luck, have fun!*

## CIS 110 Exam 1 Cheat Sheet

```
/** 1. For syntax, look at the format of the code presented in the questions
 *      (unless otherwise stated, it is all syntactically correct code). */
```

```
// class declarations      // method declarations
public class <name> {      public static <type> <name>(<params>) {
    <methods>                <statements>
}                            }
```

```
/** 2. Useful methods for String objects */
```

```
charAt(index)           // Returns the character at the given (zero-based) index.
endsWith(text)          // Returns true if the string ends with the given text.
indexOf(character)      // Returns the (zero-based) index of the given character.
length()                // Returns the length of the string.
startsWith(text)        // Returns true if the string starts with the given text.
substring(start, stop) // Returns the characters from the start index to just
                        //   before the stop index
toLowerCase()           // Returns a new string with all lower case characters.
toUpperCase()           // Returns a new string with all upper case characters.
```

```
/** 3. Useful methods for Graphics objects */
```

```
drawLine(x1, y1, x2, y2) // Draws a line between (x1, y1) and (x2, y2)
drawOval(x, y, width, height) // Draws the outline of the largest oval that
                                //   fits in the specified rectangle.
drawRect(x, y, width, height) // Draws the outline of the specified rectangle.
drawString(msg, x, y)         // Draws the given text with its lower-left
                                //   corner at (x, y)
fillOval(x, y, width, height) // Fills the largest oval that fits in the
                                //   specified rectangle.
fillRect(x, y, width, height) // Fills the outline of the given rectangle.
setColor(color)               // Sets the graphics context to use the color.
```

## Expression Evaluating Machines

1. (10 points) Evaluate these Java expressions to their final values.

(a)  $10 - 5 * 5 * (2 - 6 + 8) \rightarrow$

(b)  $3 / (\text{double}) 2 + 3 / 2 + (\text{double}) 3 / 2 \rightarrow$

(c)  $(11 / 5 + 11 \% 5) \% 5 \rightarrow$

(d)  $9 - 6 + \text{"-heads-"} + 2 * 5 + \text{"-tails-"} + (6 / 2) \rightarrow$

(e) `String s = "cis110 rocks";  
"!" + s.charAt(2) + s.charAt(8) + ": " + s.length() + "!"`  $\rightarrow$

**Grading notes:** *We awarded all-or-nothing points for each part of this question. Common mistakes included*

(a) *Forgetting the ".0" on part (b) since the result is a double and*

(b) *Off-by one errors with charAt in part (e).*

## Trace You A Loop

2. (10 points) Trace the execution of the following loop by filling in the provided table. Each row of the table corresponds to the state of the program after executing the line marked **HERE**. You may skip any cases of the variables that do not reach execution of the line marked **HERE**.

```
for (int i = 1; i <= 2; i++) {
    for (int j = 4; j >= -4; j -= 4) {
        for (int k = -1; k < 3; k += 2) {
            System.out.println(i * j * k); // HERE
        }
    }
}
```

i	j	k	Output (each line)
1	4	-1	-4
1	4	1	4
1	0	-1	0
1	0	1	0
1	-4	-1	4
1	-4	1	-4
2	4	-1	-8
2	4	1	8
2	0	-1	0
2	0	1	0
2	-4	-1	8
2	-4	1	-8

**Grading notes:** We grouped the tests into buckets based off of the patterns that emerged:

**+9:** Math errors or other minor mistakes (e.g., many people had  $1 * 0 * 2 = 2$ ).

**+8:** Misinterpretation of a bound or increment/decrement (e.g., if the table was correct if we assume  $j < 4$  instead of  $j \leq 4$

**+7:** Correct output is there, but there is extra output (e.g., from looping too many times).

**+5:** Correctly interpreted outer for-loop but the inner for-loops are wrong.

**+3:** Incremented all three loops at the same same (so  $i$ ,  $j$ , and  $k$  all stepped together on each iteration instead of just one of them).

**+1:** Attempt at the problem but not much else correct.

## Method Mystery

3. (15 points) Write the output of this program in the space below.

```
public class Mystery {
    public static String prepend(String s) {
        return ":" + s;
    }

    public static void silly(String x, String y, String z) {
        System.out.println(z + ":" + y + ":" + x);
    }

    public static void main(String[] args) {
        silly("hi", "how is the", "weather?");
        silly("pokedex " + 25, "weight " + 13.2, "initial " + 'p');
        silly("////", "\\\"\\\"\\\"\\\"\\\"\\\"\\\"");
        silly("1958", "Silverstone", prepend("Grand V"));
        silly(prepend(prepend("0")), "0", prepend("0"));
    }
}
```

```
weather?:how is the:hi
initial p:weight 13.2:pokedex 25
\\:"":////
:Grand V:Silverstone:1958
:0:0:::0
```

**Grading notes:** *We assigned three points to each of the five method calls and subtracted points for each thing that we found wrong. Most commonly, forgetting a backslash in the third method call or a colon in the fifth resulted in a point lost.*

## Patterns, Patterns, Patterns

4. (20 points) Consider the following ASCII picture:

```
@&-()  
@@&&-&-()  
@@@@&&-&-&-()  
@@@@@@@@&&-&-&-&-()  
@@@@@@@@@@@@&&-&-&-&-&-()
```

(a) What is the form of each line (i.e., what is the pattern of characters on each line)?

```
"@", "&-", "()"
```

(b) For each pattern of characters you identify, give a formula for the number of occurrences of each pattern on each line in terms of the row number  $i$ . You may choose to number the first row either 0 or 1.

```
(With  $i$  starting at 0.)  
"@" =  $2 * (i + 1)$   
"&~" =  $i + 1$   
"()" =  $1$ 
```

(c) Finally, write a method, `drawShape()` that draws the above diagram in terms of the formulae that you derived.

```
public static void drawShape() {  
    for (int i = 0; i < 5; i++) {  
        for (int j = 0; j < 2 * (i + 1); j++) {  
            System.out.print("@");  
        }  
        for (int j = 0; j < i + 1; j++) {  
            System.out.print("&-");  
        }  
        System.out.println("()");  
    }  
}
```

**Grading notes:** Part (a) was 4 points. We were extremely lenient, giving anyone that showed evidence that they understood the line was composed of spaces @s, &-s, and ()s full credit.

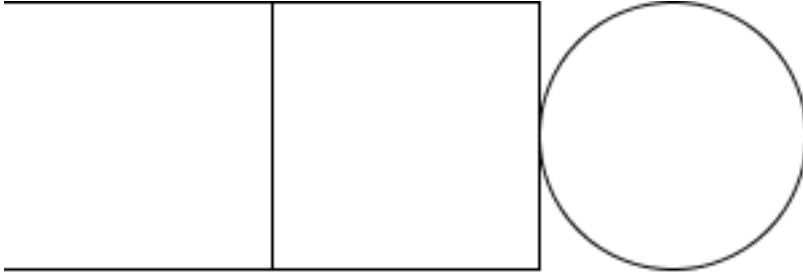
Part (b) was 6 points, 2 points each of the formulae you should've produced. We accepted any formulae that worked, e.g., grouping the ampersands into 2s vs. 1s.

Part (c) was 10 points. We took away a point for each mistake we found, in particular, for getting the boundaries of the loops wrong, forgetting to print out certain characters such as the parentheses, or not having a function header.

## Parameterization

5. (20 points)

- (a) Write a method, `drawFigure`, that takes a `Graphics` object as input and draws the following figure to that `Graphics` object.



The figure is anchored at  $(0,0)$ , and its overall size is  $300 \times 100$ . There are two lines, a box, and a circle:

- The two lines have initial points  $(0,0)$  and  $(0,100)$ , and both have length 100.
- The square is positioned right at the endpoints of the lines and has width 100.
- The circle is positioned against the right side of the square and has diameter 100.

```
public static void drawFigure(Graphics g) {
    g.drawLine(0, 0, 100, 0);
    g.drawLine(0, 100, 100, 100);
    g.drawRect(100, 0, 100, 100);
    g.drawOval(200, 0, 100, 100);
}
```

(Part (b) is on the next page.)



- (b) Parameterize `drawFigure` further so that the caller of `drawFigure` can *change the width of the figure without repositioning it*. While the figure changes in width, it should still have height 100 and remain rooted at (0,0). To do this, you should add one extra parameter to control the width of the figure.

As an example, if someone calls `drawFigure(g, 150)`:

- The lines will have width 50.
- The square will be positioned at (50,0), and its width will change appropriately.
- The circle will be positioned at (100,0), and its width will also change appropriately.

You may either write your parameterized `drawFigure` method in the space below or change your answer to part (a) to be parameterized.

(*Hint*: What is the relationship between the width of the overall figure and the widths of its subfigures? This value, the `subWidth`, is important in solving the problem.)

```
public static void drawFigure(Graphics g, int width) {
    int subWidth = width / 3;
    g.drawLine(0, 0, subWidth, 0);
    g.drawLine(0, 100, subWidth, 100);
    g.drawRect(subWidth, 0, subWidth, 100);
    g.drawOval(2 * subWidth, 0, subWidth, 100);
}
```

**Grading notes:** *Part (a) was 6 points given as follows:*

- +2 *Correct method declaration.*
- +2 *Made 4 correct method calls.*
- +2 *Correct parameters passed to those method calls.*

*Part (b) was given 14 points as follows:*

- +2 *Correct method declaration.*
- +2 *Made 4 correct method calls.*
- +3 *Includes the concept of `subWidth` (i.e., `subWidth = width / 3`) into their code.*
- +1 *Correct starting x/y coordinates for `drawLines`.*
- +2 *Correct ending x coordinates for `drawLines`.*
- +2 *Correct ending y-coordinate for `drawLine` and heights for `drawRect` and `drawOval`.*
- +2 *Correct widths for `drawRect` and `drawOval`.*

## Gonna Fly Now

6. (25 points)

- (a) Write a method, `printRow(x, y, n)` that writes a row of  $y$  successive numbers starting from  $x$  to  $x + y - 1$  modded by  $n$ . For example, `printRow(2, 8, 5)` will print out the following eight numbers starting at two.

2 3 4 0 1 2 3 4

```
public static void printRow(int x, int y, int n) {
    for (int i = x; i < x + y; i++) {
        System.out.print(i % n + " ");
    }
}
```

(Part (b) is on the next page.)

- (b) Use `printRow(x, y, n)` to write a second method `printModTriangles(k, n)` that writes repeated triangles consisting of such rows. The method prints out  $k$  rows that make up a set of repeated triangles where the lengths of each row are also modded by  $n$ . For example, `printModTriangles(8, 5)` should print:

```
(0) 0
(1) 1 2
(2) 3 0 1
(3) 2 3 0 1
(4) 2
(5) 3 0
(6) 1 2 3
(7) 0 1 2 3
```

(*Hint*: The difficult part is determining what values to pass to `printRow(x, y, n)`. Tackle each parameter independently based off of the patterns that you see in the above picture.)

```
public static void printModTriangles(int x, int n) {
    int currentValue = 0;
    for (int i = 0; i < x; i++) {
        System.out.print("(" + i + ") ");
        printRow(currentValue, i \% (n-1) + 1, n-1);
        currentValue += i + 1;
        System.out.println();
    }
}
```

**Grading notes:** *Woo boy. We were well-aware that this question was the hardest one on the test. So we shifted most of the points to part (a) and looked to give as much partial credit as possible. Part (a) was 15 points. This question was very difficult to do without mod, so we broke up the points accordingly.*

- *If mod was used to generate output, you started at 15 pts:*
  - 1: *Incorrect method header.*
  - 1: *Multiple syntax errors.*
  - 1: *Missing space between numbers in output.*
  - 2: *Loop variable is initialized wrong.*
  - 2: *Loop bound is wrong.*
  - 3: *Extra unnecessary for-loops.*
  - 2: *Stringifying mod (i.e., something like a string "5 % 3"*
  - 3: *Hardcoded in 5 into the method.*
  - 2: *Unnecessary return statements.*
  - 2: *Improper scoping of variables.*

- *If mod was not used to generate output, you started at 0 pts:*
  - +1: Method header correct.*
  - +1: Having a for-loop.*
  - +1: Printing something.*
  - +1: Printing with spaces.*
  - +2: Using loop variables to generate output.*

*Part (b) was given 10 points.*

- +1: Having parameters and a correct method header.*
- +1: Having a for-loop.*
- +1: Printing out the number "(#)" each iteration.*
- +1: Calling `printRow`*
- +2: Passing correct first parameter to `printRow`.*
- +2: Passing correct second parameter to `printRow`.*
- +2: Passing correct third parameter to `printRow`.*
- +1: Printing a newline at the end of each iteration of the loop.*

*For part (b), we accepted either  $n$  or  $n - 1$  for the various parameter calls. Technically the example works as given (corresponding to  $n - 1$ ) but our intention was for that example to be `printModTriangles(8, 4)` which would make the math easier (corresponding to  $n$ ).*

Postscript (extra paper)