CIS 110 Fall 2014 — Introduction to Computer Programming
17 Dec 2014 — Final Exam

Name: ______________________________

Recitation # (e.g., 201): ______________________________

Pennkey (e.g., eeaton): ______________________________

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

______________________________ ____________________
Signature Date

Instructions:

(a) **Do not open this exam until told by the proctor.** You will have exactly 120 minutes to finish it.
(b) **Make sure your phone is turned OFF (not to vibrate!) before the exam starts.**
(c) Food, gum, and drink are strictly forbidden.
(d) **You may not use your phone or open your bag for any reason**, including to retrieve or put away pens or pencils, **until you have left the exam room**.
(e) This exam is closed-book, closed-notes, and closed-computational devices.
(f) If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
(g) All code must be written out in proper Java format, including all curly braces and semicolons.
(h) Do not separate the pages. If a page becomes loose, reattach it with the provided staplers.
(i) Staple all scratch paper to your exam. Do not take any sheets of paper with you.
(j) If you require extra paper, please use the backs of the exam pages or the extra pages 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")**.
(k) Use a pencil, or blue or black pen to complete the exam.
(l) If you have any questions, raise your hand and a proctor will come to answer them.
(m) When you turn in your exam, you may be required to show ID. **If you forgot to bring your ID, talk to an exam proctor immediately**.
(n) We wish you the best of luck. Have a great Fall break!

Scores: [For instructor use only]

Question 0		1 pts
Question 1		10 pts
Question 2		10 pts
Question 3		12 pts
Question 4		12 pts
Question 5		8 pts
Question 6		30 pts
Question 7		30 pts
Total:		113 pts

TOY Reference Card

```
INSTRUCTION FORMATS

             | . . . . | . . . . | . . . . | . . . .|
  Format 1:  | opcode  |    d    |    s    |    t   |  (0-6, A-B)
  Format 2:  | opcode  |    d    |       addr       |  (7-9, C-F)


ARITHMETIC and LOGICAL operations
    1: add              R[d] <- R[s] +  R[t]
    2: subtract         R[d] <- R[s] -  R[t]
    3: and              R[d] <- R[s] &  R[t]
    4: xor              R[d] <- R[s] ^  R[t]
    5: shift left       R[d] <- R[s] << R[t]
    6: shift right      R[d] <- R[s] >> R[t]

TRANSFER between registers and memory
    7: load address     R[d] <- addr
    8: load             R[d] <- mem[addr]
    9: store            mem[addr] <- R[d]
    A: load indirect    R[d] <- mem[R[t]]
    B: store indirect   mem[R[t]] <- R[d]

CONTROL
    0: halt             halt
    C: branch zero      if (R[d] == 0) pc <- addr
    D: branch positive  if (R[d] >  0) pc <- addr
    E: jump register    pc <- R[d]
    F: jump and link    R[d] <- pc; pc <- addr


Register 0 always reads 0.
Loads from mem[FF] come from stdin.
Stores to mem[FF] go to stdout.
```

0.) THE EASY ONE (1 point total)

(a) Check to make certain that your exam has all 11 pages (excluding the cover sheet).

(b) Write your name, recitation number, and PennKey (username) on the front of the exam.

(c) Sign the certification that you comply with the Penn Academic Integrity Code.

1.) I CHOOSE THEREFORE I AM (10 points total)

For each question below, circle the correct answer(s):

1.1) (2 points) Which data structure(s) give(s) the fastest access to the median element (e.g. 5th-highest element out of 11)?

(a) binary search tree (assume balanced)
(b) stack
(c) queue
(d) array
(e) graph
(f) sorted array
(g) None of the above

1.2) (2 points) Which of the following are correct method signatures for the `toString()` method?

(a) `public void toString()`
(b) `public void toString(String s)`
(c) `public String toString()`
(d) `public String toString(String s)`
(e) `public static void toString()`
(f) `public static void toString(String s)`
(g) `public static String toString()`
(h) `public static String toString(String s)`
(i) None of the above

1.3) (2 points) What memory address comes ten words after memory address `0x15` in TOY?

Your answer: ____________

1.4) (2 points) Before covering objects, we insisted that every method be `public static` because:

(a) That is the only way one method can call another method.

(b) You can only use non-`static` method if you have global (instance) variables.

(c) `main()` is a `static` function and can only make calls to other `static` method.

(d) You can't call `static` method without instantiating an object first.

(e) None of the above.

1.5) (2 points) The implementation of a well-written recursive method will **always** involve:

(a) A `while` loop
(b) A `for` loop
(c) A conditional
(d) A global variable
(e) A base case
(f) A print statement
(g) None of the above

2.) TO ERR IS HUMAN. TO COMPLAIN IS JAVA (10 points total)
For each of the run-time exceptions below, write a sequence of at most two simple **Java statements** that, if contained in a `main()` function, will **always** trigger the exception.

2.1) (2 points) `ArrayIndexOutOfBoundsException`

____

____

2.2) (2 points) `NullPointerException`

____

____

2.3) (2 points) `StringIndexOutOfBoundsException`

____

____

2.4) (2 points) `NumberFormatException`

____

____

2.5) (2 points) `ArithmeticException`

____

____

3.) EXCEPTIONALLY BAD MAZES (12 points total)

Instead of reading in a `.maze` file, you would like to create a `Maze` object from an array of `Vertex` objects (none of which contain any edges yet), and an *adjacency matrix* `adj`. An adjacency matrix is a 2-D `boolean` array where the value at row `i` and column `j` is `true` (`T`) if there is an edge from vertex `i` to vertex `j`, and `false` (`F`) otherwise. For example

$$\begin{pmatrix} \mathtt{F} & \mathtt{T} & \mathtt{F} \\ \mathtt{F} & \mathtt{F} & \mathtt{T} \\ \mathtt{F} & \mathtt{F} & \mathtt{T} \end{pmatrix}$$

indicates there are edges from vertex 0 to vertex 1, from vertex 1 to vertex 2, and from vertex 2 to itself.

In the code below, fill in the blanks to complete the new `Maze` constructor. We have written everything that calls `Vertex` methods for you, so you shouldn't need to remember any details of `Vertex` or `Maze`. The line numbers are for reference and are not part of the code.

```
 1: public class Maze {
 2:     private Vertex[] rooms;
 3:
 4:     _______________ Maze(_______________ rooms, _______________ adj) {
 5:         this.rooms = rooms;
 6:
 7:         for (int i = 0; i < rooms.length; i++)
 8:             if (!rooms[i].getAdjacent())
 9:                 throw new RuntimeException("Vertex " + i +
10:                     " already contains outgoing edges.");
11:
12:         int numRows = Math.max(rooms.length, adj.length);
13:
14:         for (int row = __________; __________ numRows; _______________) {
15:             int numCols = Math.max(rooms.length, adj[row].length);
16:
17:             for (int col = __________; _________ numCols; _______________)
18:
19:                 if (____________________)
20:
21:                     rooms[__________].addEdge(rooms[__________]);
22:         }
23:     }
24:
25:     // ... (imagine the rest of the Maze class is here)
26: }
```

4.) EXCEPTIONALLY BAD MAZES II (12 points total)
When you test your new constructor in the previous question with a variety of inputs, you start seeing runtime errors. For each of the errors below, describe in 20 words or less what could have caused the error. If there is more than one possible cause, list them all. The first one has been completed for you.

(a) `NullPointerException` at line 7
`rooms` is `null`

(b) `NullPointerException` at line 8

(c) `NullPointerException` at line 12

(d) `NullPointerException` at line 15

(e) `ArrayIndexOutOfBoundsException` at line 15

(f) `ArrayIndexOutOfBoundsException` at line 19

(g) `ArrayIndexOutOfBoundsException` at line 21

5.) IMPLEMENTING INTERFACES OF INTEGERS (8 points total)
Write a class `IntegerBox` that implements the `SwappableInteger` interface below. `SwappableInteger` defines a integer type whose values can be easily swapped with each other. We provide the skeleton of `IntegerBox` for you. **You only need to fill in the method implementations.** You do not need to perform any error checking or write any comments. Your code should be short and simple.

```
interface SwappableInteger {
    public String toString();              // returns string representation
    public void setValue(int val);         // set value
    public int  getValue();                // get value
    public void swap(SwappableInteger si); // swaps value with si's value
}

class IntegerBox implements SwappableInteger {
    private int val;

    public IntegerBox(int val) {


    }

    public IntegerBox(String val) {


    }

    public String toString() {


    }

    public void   setValue(int val) {


    }

    public int    getValue() {


    }

    public void   swap(SwappableInteger si) {





    }
}
```

6.) A IS FOR ARVIND, B IS FOR BENEDICT (30 points total)
The code on the following page relies on your `IntegerBox` class from the previous question. (Assume that your code works correctly.) When the program `B` is run with no arguments, each point marked "`// Point XX`" in the code below will be reached exactly once. Fill in the table below with the order in which the points are reached and the value of each listed variable at that point (immediately after the preceding line is executed). If any variable is not in scope, write **N/A**. The first row is filled in for you. You may yank out the code page, and staple it to the back of your exam when you turn it in.

Point	`a`	`b`	`this.a`	`this.b`	`B.a`
M1	2	1	N/A	N/A	2

THE CODE YOU NEED TO TRACE IS ON THE NEXT PAGE

A is for Arvind, B is for Benedict (Cont'd)

Fill in the table on the previous page based on this code:

```
public class A {
    public SwappableInteger b = new IntegerBox(6);
    public SwappableInteger a = new IntegerBox(7);

    public A(SwappableInteger b, SwappableInteger a) {
        // Point A1

        this.b.swap(b);
        // Point A2

        this.a.swap(a);
        // Point A3
    }
}

public class B {
    public SwappableInteger b = new IntegerBox(4);
    public static SwappableInteger a = new IntegerBox(5);

    public B(SwappableInteger b, SwappableInteger a) {
        // Point B1

        a.swap(b);
        // Point B2

        B.a.swap(this.b);
        // Point B3

        this.b.swap(b);
        // Point B4

        A ba = new A(this.b, B.a);
        // Point B5
    }

    public static void main(String[] args) {
        SwappableInteger b = new IntegerBox(1);
        a = new IntegerBox(2);
        // Point M1

        SwappableInteger a = new IntegerBox(3);
        // Point M2

        B ba = new B(b, a);
        // Point M3
    }
}
```

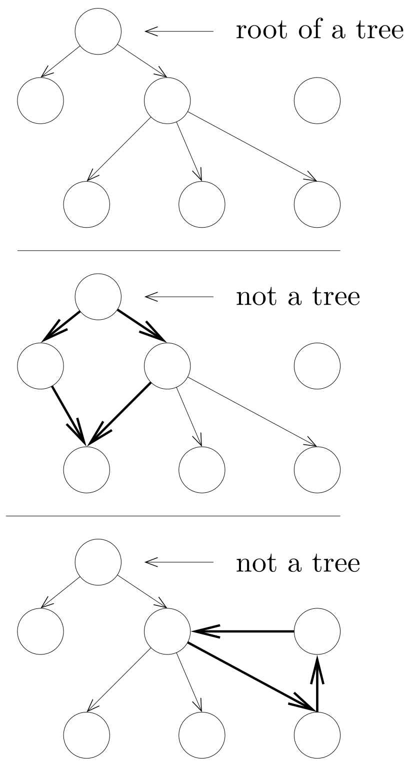
7.) WHAT A TREE-EET! (30 points total)

Recall that a graph vertex is the root of a `tree` if none of the paths leaving it loop back on themselves. Your job is to write a method to tell if a graph vertex is the root of a tree, i.e. if there is at most one path from there to any other vertex.

The `ListNode` class below represents a linked lists of vertices similar to the one from your Maze assignment. The `GraphVertex` class represents a vertex in a graph with a linked list of edges leaving it, just like your Maze assignment. Each vertex stores a `value` and has a `boolean` variable `mark` for internal use by the class's methods. (You may do whatever you like with `mark`.) The `allGraphVertices` variable is a linked list of *every* `GraphVertex` that has been created in your program.

Write a public method `isTreeRoot` that takes no arguments and returns `true` if the `GraphVertex` it is called on is the root of a tree (there is at most one path from it to every other vertex), and `false` otherwise.

- Assume that the graph structure is well-formed (e.g. the `graphVertex` field of an existing `ListNode` object is never `null`).
- Create as many private helper methods as you need in addition to `isTreeRoot()`. We will assume these methods are part of the `GraphVertex` class.
- You are not required to write any comments, but you will not penalized for doing so.
- Do not assume that `mark` starts out `true` or `false`. You may change its value however you like.
- Do not add any instance variables to either class.
- Do not write the class structure, only the methods.
- Do not change any vertex's `value`.
- Do not use `new`.



```
public class ListNode {
    public GraphVertex graphVertex;
    public ListNode next;
}

public class GraphVertex {
    private static ListNode allGraphVertices;
    private int value;
    private boolean mark;
    private ListNode edges;

    // Your isTreeRoot() method and any helper methods
    // will go here
}
```

WRITE YOU ANSWER ON THE FOLLOWING PAGE

What a Tree-eet! (Cont'd)

Write your `isTreeRoot` method and any helper methods you need on this page. *(Hint: Use at least one recursive helper method.)*

[**Scrap Paper: This Page Intentionally Blank**]