CIS 110 — Introduction to Computer Programming

27 June 2013 — Final Exam

Name: _____

Recitation # (e.g. 201): _____

Pennkey (e.g. bjbrown): _____

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

_____ _____

Signature Date

Scores:

0		1
1		5
2		12
3		10
4		18
5		24
Total:		70

CIS 110 Exam Instructions

- You have 115 minutes to finish this exam. Time will begin when called by a proctor and end precisely 120 minutes after that time. If you continue writing after the time is called, you will receive a zero for the exam.
- **Make sure your phone is turned off before the exam starts. If it vibrates or rings during the exam, you will receive a substantial penalty.**
- Food and drink are strictly forbidden, **including water and gum**.
- 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**.
- This exam is *closed-book, closed-notes, and closed-computational devices*. Except where noted, code included in the questions is correct and you may use it as a reference for Java syntax.
- If you get stuck part way through a problem, it may be to your advantage to go on to another problem and come back later if you have time.
- All code must be written out as normal, including all curly braces and semicolons, unless the question states otherwise.
- Do not separate the pages of the exam. If a page becomes loose, write your name on it and use the provided staplers to reattach the sheet when you turn in your exam so that we don't lose it. We reserve the right not to grade any answers on loose sheets of paper.
- Turn in all scratch paper that you use during the exam. Do not take any sheets of paper with you or leave them behind.
- If you require extra paper, please use the backs of the exam pages or the extra sheet(s) of 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"). Staple an extra sheets you use to the back of your exam when you turn it in using the provided staplers.
- Use a pencil, or blue or black pen to complete the exam. All other colors are reserved for grading. If you do not have an appropriate writing utensil, raise your hand, and we will give you a pencil.
- If you have any questions, raise your hand and an exam proctor will come to answer them.
- 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.

Good luck and have fun!

Miscellaneous

0. (1 point)

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

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

True/False

1. (5 points)

Say whether each of the statements below is **True**, **False**, or **Sometimes** true and sometimes false.

(a) `Array`s have a method named `length` that returns their size.

True False Sometimes

(b) The default value for `boolean` variables is `false` if you don't explicitly initialize them.

True False Sometimes

(c) Each class can have only one constructor.

True False Sometimes

(d) It would make more sense to use a linked list than an array for the `RingBuffer` object in the Guitar Hero assignment.

True False Sometimes

(e) Linked list elements are always stored sequentially in memory.

True False Sometimes

Short Answer (2 Pages)

2. (12 points)
Answer each of the three questions below and on the next page:

(a) List all combinations of method signatures below that could ***not*** simultaneously be part of the same class. Draw a box around your answer so we can distinguish it from your scratch work.

i. `public int add(Node foo, Node bar, Node foobar)`

ii. `public int add(Node b, int a, Node c)`

iii. `public int add(int c, Node b, Node a)`

iv. `public int add(Node b, Node d, Node foo)`

v. `public int add(Node bar, Node foobar)`

vi. `public double add(Node a, Node b, Node c, Node d)`

vii. `public double add(int a, Node c, Node b)`

viii. `private int add(Node head, int x, Node tail)`

Short Answer(Cont'd)

(b) Consider a stack of integers `s` with a `push()` method to add elements and a `pop()` method to remove and return elements. Assuming `s` starts out empty, fill in the blanks in the code sequence below so it is correct.

```
s.push(4);

s.push( _____ );
s.push(8);

s.pop(); returns  _____
s.push(6);

s.push( _____ );
s.push(7);
s.pop(); returns 7
s.pop(); returns 4

s.pop(); returns  _____
s.pop(); returns 6

s.pop(); returns  _____
```

(c) Give one example (10 words or less) of a situation where you would use a more complex data structure than a linked list, array, single object, or variable. *Hint: This is a short answer question, don't over-think it.*

Amaze Us (2 Pages)

3. (10 points)

The following methods are all things we thought about making you write as extra methods in the `Maze` class for your final project. But then we relented. For each one, state in a single sentence (20 words or less), what it does. We do not need a detailed description of when the method will succeed or fail, nor do you need to explain *how* it works. You only need to tell us *what* its purpose is assuming the inputs are valid. *(Hint: Remember that the list of edges leaving a* `Vertex` *starts with a dummy node.*

In the methods below, `rooms` is the array of `Vertex`es in the maze.

(a)
```
public static int A(Vertex v) {
  int a = 0;
  for (EdgeListNode n = v.edges.next; n != null; n = n.next)
    a++;
  return a;
}
```

(b)
```
public double B() {
  double b = 0;
  for (int i = 0; i < rooms.length; i++)
    b += A(rooms[i]);
  return b / rooms.length;
}
```

(c)
```
public boolean C(int x, int y) {
  for (EdgeListNode n = rooms[x].edges.next; n != null; n = n.next)
    if (n.target == y) return false;
  return true;
}
```

Amaze Us (Cont'd)

(d)
```
public int[] D() {
  int[] d = new int[rooms.length];
  for (int i = 0; i < d.length; i++)
    for (EdgeListNode n = rooms[i].edges.next; n != null; n = n.next)
      d[n.target]++;
  return d;
}
```

(e)
```
public void E(int x, int y) {
  EdgeListNode e = rooms[x].edges;
  while (e.next != null) e = e.next;
  e.next = new EdgeListNode();
  e.next.target = y;
}
```

(f)
```
public void F() {
  for (int i = 0; i < rooms.length; i++)
    for (EdgeListNode n = rooms[i].edges.next; n != null; n = n.next)
      if (C(n.target, i)) E(n.target, i);
}
```

Let's Be Leakers

4. (18 points) Assume you call the `manning` function below with three integer arguments. For each of the six points indicated in the program, say whether the listed conditions are sometimes, always, or never true. Write **S**, **A**, and **N** for sometimes, always, and never.

```
public class Leakers {
  private static int x, y, z;

  public static void manning(int x, int y, int z) {
    z = y * y;
    if (y < 0) y = -y;
    y++;
    x = x / y;
    // Point A
    if (x < 0)      z = y + x;
    else if (x > 0) z = y - x;
    else            z = y;
    // Point B
    z = snowden(2 * y + Math.abs(x), y, z);
    y = snowden(z, 2 * z, y);
    // Point C
    if (x < y) {
      x = ellsberg(2 * z, z, y);
      y = x;
      // Point D
    } else {
      x++;
      y = y - x;
      // Point E
    }
    // Point F
  }

  private static int snowden(int x, int y, int z) {
    while(x > y) {
      z++;
      x--;
    }
    return z;
  }

  private static int ellsberg(int x, int y, int z) {
    if (x > y) return ellsberg(x - 1, y, z) + 1;
    else       return z;
  }
}
```

	z > y	x > 0	y >= 0
A			
B			
C			
D			
E			
F			

Link, Don't Blink! (3 Pages)

5. (24 points)

In a doubly linked list, each node contains links to both the next *and* previous elements so you can move backwards and forwards in the list. Consider the following classes to represent a doubly linked-list node of integers, and to encapsulate a doubly linked list. For the list to be valid (not containing any loops or other oddities), certain conditions have to be true on each node and on the list's head and tail; these conditions are documented in comments in the code below.

```
public class DNode {
  // to be a valid doubly linked-list (no loops or anything else funny),
  // the following expressions should always be true for all DNodes:
  // (next == null) || next.prev == this
  // (prev == null) || prev.next == this

  public DNode next; // the next node in the list
  public DNode prev; // the previous node in the list
  public int   val;  // the value stored at this node
}

public class DLinkedList {
  // to be a valid doubly linked list, either the list should be empty:
  // (head == null) && (tail == null)
  // or the following expressions should be true:
  // (head != null) && (head.prev == null)
  // (tail != null) && (tail.next == null)

  public DNode head; // the first node in the list
  public DNode tail; // the last node in the list
}
```

Your job is to write a constructor and a method for the `DLinkedList` class, described on the next two pages. You do not need to do any error checking that is not specified in the questions, and you do not need to write any comments or a containing class. Your code can be iterative or recursive, or a combination of the two.

Link, Don't Blink! (Cont'd)

(a) Write a constructor for `DLinkedList` that takes an array of integers and inserts them in order into the double linked list. If the array is null or empty, the list should remain empty.

Link, Don't Blink! (Cont'd)

(b) Write a method, `reverse`, for the `DLinkedList` class that reverses the list's direction so the head becomes the tail and vice versa. For full credit you must not use the `new` keyword.