CIS 110 — Introduction to Computer Programming
Spring 2016 — 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:

- **Do not open this exam until told by the proctor**. You will have exactly 110 minutes to finish it.
- **Make sure your phone is turned OFF (not to vibrate!) before the exam starts.**
- Food, gum, and drink are strictly forbidden.
- **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*.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All answers must be written on the exam booklet.
- All code must be written out in proper java format, including all curly braces and semicolons.
- Do not separate the pages. If a page becomes loose, re-attach it with the provided staplers.
- Scratch paper is provided at the end of the exam. Do not take any sheets of paper with you.
- 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").**
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a 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.**
- We wish you the best of luck.

Scores:

[For instructor use only]

0. Cover Page Info		1 pt
1. Miscellaneous		10 pts
2. Sorting and Searching		18 pts
3. LinkedLists & ArrayLists		25 pts
4. Recursion		10 pts
5. Linked Data Structures		10 pts
6. Debugging		12 pts
7. Tracery		10 pts
8. OO Memory		14 pts
Total:		110 pts

0. (1 pt) Cover Page Information:

- Check that your exam has all 12 pages (excluding the scratch paper).
- Write your name, recitation number, and PennKey (username) on the front of the exam.
- Sign the certification that you comply with the Penn Academic Integrity Code.

SECTION 1: MISCELLANEOUS (10 pts; 1 pt each)

Is each of the following statements true or false? (Circle the correct answer)

TRUE or FALSE 1.1) A java class with the following two methods will compile:

```
public void add(int e, int index)
public boolean add(int equals, int i)
```

TRUE or FALSE 1.2) Each class can only have one constructor.

TRUE or FALSE 1.3) The java compiler will warn you if your code will generate `RuntimeException`s.

TRUE or FALSE 1.4) `IllegalArgumentException`s are typically used to signify that a method has been called with inappropriate values for the arguments.

TRUE or FALSE 1.5) Static variables belong to the class; as opposed to fields, which belong to an instance of a class.

Should each of the following variables/methods be declared static or non-static? (Circle one)

STATIC or NON-STATIC 1.6) A timer keeping track of the number of seconds since 1970

STATIC or NON-STATIC 1.7) An instance-level `printGreeting()` method that always outputs the string `"Hello and welcome!"`

STATIC or NON-STATIC 1.8) The sine function that takes in a number `x` and returns `sin(x)`

STATIC or NON-STATIC 1.9) A method `getAverage()` in a `StudentRecord` class that returns the average of a student's test scores

STATIC or NON-STATIC 1.10) A variable `numTimesPlayed` in the class `SitarString` that tracks the number of times that `SitarString` was plucked

SECTION 2: SORTING AND SEARCHING (18 pts total)

2.1) (6 pts) A student intends to evaluate selection sort, insertion sort and mergesort, based on the number of times each algorithm compares a pair of elements in the array. How many comparisons will each sorting algorithm make on the array: `{8, 1, 5, 2, 4, 1}`?

Algorithm	Number of Comparisons
Selection Sort	
Insertion Sort	
Merge Sort	

For each of the following situations, choose the algorithm we studied that will perform the best.

2.2) (2 pts) You are sorting data that is stored on a remote file server over the network. Using the network connection, it is extremely expensive to "swap" two elements. However, looping over the elements and looking at their values is very inexpensive. You want to minimize swaps above all other factors.

a. selection sort b. insertion sort c. mergesort d. None of the above

2.3) (2 pts) You have a fast computer with many processors and lots of memory. You want to choose a sorting algorithm that is fast and can also be parallelized easily to use all processors to help sort the data.

a. selection sort b. insertion sort c. mergesort d. None of the above

2.4) (2 pts) You have an array that is already sorted. Periodically, some new data arrives and is added to the array at random indexes, messing up the ordering. You know the indices of the new data, and need to re-sort the array to get it back to being fully ordered as efficiently as possible.

a. selection sort b. insertion sort c. mergesort d. None of the above

2.5) (2 pts) Now that you have a fair knowledge about programming, you decide to build cool things on your own. You start off by building a search engine! The graphical user interface is done and you now wish to add an autocomplete feature that will fill in the word as you type it.

a. selection sort b. insertion sort c. mergesort d. None of the above

2.6) (2 pts) What is the computational efficiency of mergesort when it is applied on a sorted array?

a. O(lg N) c. O(0.5N lg(0.5n)) e. $O(N^2)$

b. O(N) d. O(N lg n) f. None of the above

2.7) (2 pts) Recall that each step of insertion sort uses linear search to determine the index to insert the next item into the sorted portion of the array. An intrepid student proposes a new form of insertion sort: they plan to locate the index of insertion using binary search. What is the computational complexity of their algorithm?

a. O(lg N) c. O(0.5N lg(0.5n)) e. $O(N^2)$

b. O(N) d. O(N lg n) f. None of the above

SECTION 3: ARRAYLISTS AND LINKEDLISTS (25 pts total)

3.1 (10 pts; 2 pts each) For each of the methods/properties listed below, circle which data structure is more efficient (i.e., has lower computational complexity). If the efficiency is the same for multiple data structures, circle multiple answers per row. Assume efficient implementations for all methods.

`list.add(0,x)`	Singly Linked List	Doubly Linked List	ArrayList
`list.add(x)`	Singly Linked List	Doubly Linked List	ArrayList
`list.get(list.size()/2)`	Singly Linked List	Doubly Linked List	ArrayList
`list.remove(0)`	Singly Linked List	Doubly Linked List	ArrayList
`total memory usage`	Singly Linked List	Doubly Linked List	ArrayList

3.2 (15 pts) One important aspect of programming is using existing generic data structures to implement new data structures. Assume that you are given `SinglyLinkedList<T>` and `ArrayList<T>` classes, which both implement the `List<T>` interface:

```
public Interface List<T> {
   public boolean add(T x);
   public void add(int index, T x);
   public void clear();
   public boolean contains(T x);
   public T get(int index);
   public int indexOf(T x);
   public boolean isEmpty();
   public Iterator<T> iterator();
   public T remove(int index);
   public T set(int index, T x);
   public int size();
}
```

```
public Interface Iterator<T> {
   public boolean hasNext();

   // returns the current element
   // and advances the iterator
   public T next();

   // removes the item previously
   // returned by next()
   public void remove();
}
```

The Identical Twin Convention of America is looking to implement a new registration system, and needs your help! Implement a new data structure called `PairedSet<T>` that contains a `List` object as a private field, either the `SinglyLinkedList<T>` or `ArrayList<T>` (choose the best option!) The `PairedSet<T>` is just like a list, but it can only contain up to two copies of a single item, the order of the items does not matter, and it implements a reduced interface called `Set<T>`. The `PairedSet<T>` should throw a `RuntimeException` if someone attempts to add more than two copies of a single item.

```
public Interface Set<T> {
   public void add(T x);
   public void clear();
   public void contains(T x);
   public Iterator<T> iterator();
   public boolean isEmpty();
   public void remove(T x);
   public void size();
}
```

Put your implementation of the `PairedSet<T>` class below:

SECTION 4: RECURSION (10 pts)

Assume that you're given an implementation of a singly linked list of strings without sentinel nodes. Add a new method to intersperse a value throughout the list. `Node` is defined as:

```
class Node {
     String value;
     Node next;
}
```

Write a pair of functions: a public method, and the corresponding private helper method. Your implementation must be recursive. The header for the public method is:

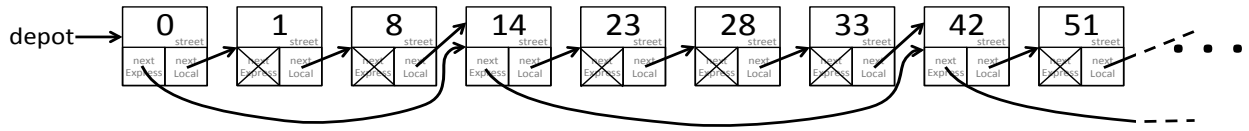
```
/**  Function Name: intersperse
 *   Parameters:  value – String to be interspersed
 *   Examples:
 *   1.) Given list: head-> "A" -> "B"-> "C" -> "D" -> null
 *    intersperse("N") should modify the list to be:
 *    "A" -> "N" -> "B" -> "N" -> "C" -> "N" -> "D" -> null
 *   2.) intersperse("N") would not modify an empty list
 */
```

SECTION 5: LINKED DATA STRUCTURES (10 pts)

The New York City Subway's Lexington Avenue Line has a local train and an express train. The local train stops at all stations, the express train only stops at the **express** stations (in **bold**).

- **Train Depot (0th)**
- 1st Street (Bleecker)
- 8th Street (Astor Pl)
- **14th St (Union Sq)**
- 23rd Street
- 28th Street
- 33rd Street
- **42nd St -Grand Central**
- 51st Street
- **59th Street**
- 68th St (Hunter College)
- 77th Street
- **86th Street**
- 96th Street
- 103rd Street
- 110th Street
- 116th Street
- **125th Street**

We can represent this as a class `SubwayLine` that is a singly linked list (without sentinel nodes), where each `Station` node has two references (to the next local and the next express station):




```
public class Station {
    public int street;                  // street number
    public Station nextExpressStation; // next express stop; null if local station
    public Station nextLocalStation;   // next local stop; null at end of the line

    public Station(int street, Station nextLocalStation, Station nextExpressStation){
        this.street = street;
        this.nextLocalStation = nextLocalStation;
        this.nextExpressStation = nextExpressStation;
    }
}
```

Write a method `printRoute(int street)` for `SubwayLine` that prints the station numbers on the shortest route from `depot` to the station on or before that street. E.g., `printRoute(32)` would output "`0,14,23,28`". Throw an `IllegalArgumentException` if the `street` is negative.

```
public void printRoute(int street){




}
```

SECTION 6: DEBUGGING (12 pts; 2 pts each)

An enterprising individual attempted to complete the implementation of the `SubwayLine` class from the previous problem, but made a number of errors. The code compiles, but the implementation does not work correctly. Find and correct the logical bugs in the code on the following page.

There are six logical bugs total. After some initial debugging, you determine that the method `buildExpressStation()` works correctly, and so all the bugs are in `buildLocalStation()` and/or `demolishStation()`. Each bug can be fixed in one line of code.

Line Num	Description of Bug	Correction

```
01 public class SubwayLine {                                                                    01
02     // both the express and local tracks begin with the depot at 0th street                  02
03     public final Station depot = new Station(0, null, null);                                03
04                                                                                             04
05     public void buildLocalStation(int i) {                                                  05
06         if (i < 0) throw new IllegalArgumentException("Invalid street");                    06
07         Station curr = null;                                                                07
08         while (curr.nextLocalStation != null && curr.nextLocalStation.street < i) {         08
09             curr = curr.nextLocalStation;                                                   09
10             if (curr.street != i)                                                           10
11                 throw new IllegalArgumentException("station exists");                       11
12         }                                                                                   12
13         curr.nextLocalStation = new Station(i, null, null);                                 13
14     }                                                                                       14
15                                                                                             15
16     public void buildExpressStation(int i) {                                                16
17         if (i < 0) throw new IllegalArgumentException("Invalid street");                    17
18         Station currExpress = depot;                                                        18
19         while (currExpress.nextExpressStation != null &&                                    19
20                currExpress.nextExpressStation.street < i) {                                20
21             currExpress = currExpress.nextExpressStation;                                   21
22             if (currExpress.street == i)                                                    22
23                 throw new IllegalArgumentException("station exists");                       23
24         }                                                                                   24
25         Station currLocal = currExpress.nextLocalStation;                                   25
26         while (currLocal.nextLocalStation != null &&                                        26
27                currLocal.nextLocalStation.street < i) {                                    27
28             currLocal = currLocal.nextLocalStation;                                         28
29             // if station exists, change it from local to express                           29
30             if (currLocal.street == i) {                                                    30
31                 currLocal.nextExpressStation = currExpress.nextExpressStation;              31
32                 currExpress.nextExpressStation = currLocal;                                 32
33                 return;                                                                     33
34             }                                                                               34
35         }                                                                                   35
36         Station newStation = new Station(i, currLocal.nextLocalStation,                     36
37                                             currExpress.nextExpressStation);                37
38         currLocal.nextLocalStation = newStation;                                            38
39         currExpress.nextExpressStation = newStation;                                        39
40     }                                                                                       40
41                                                                                             41
42     public void demolishStation(int i) {                                                    42
43         Station currLocal = depot;                                                          43
44         Station currExpress = depot;                                                        44
45         while (currLocal.nextLocalStation.street != i) {                                    45
46             currLocal = currLocal.nextLocalStation;                                         46
47             if (currLocal == null || currLocal.street > i) {                                47
48                 throw new IllegalArgumentException("Station doesn't exist");                48
49             }                                                                               49
50             if (currLocal.nextExpressStation != null) {                                     50
51                 //TODO                                                                      51
52             }                                                                               52
53         }                                                                                   53
54         if (currExpress.nextExpressStation == currLocal.nextLocalStation) {                 54
55             currExpress.nextExpressStation =                                                55
56               currExpress.nextExpressStation.nextExpressStation;                            56
57         }                                                                                   57
58         currLocal.nextLocalStation = currLocal.nextLocalStation;                            58
59     }                                                                                       59
60 }                                                                                           60
```

SECTION 7: TRACERY (10 pts)

What would the following program, composed of the four files below, print after being run via:

```
java AnimalHouse Fred Georgia Rita Sammy Bob
```

```
public class AnimalHouse {
    public static void main(String[] args) {
        Pet[][] pets = new Pet[2][];
        int idx = 0;
        for (int i = 0; i < pets.length; i++) {
            pets[i] = new Pet[args.length / 2 + i * args.length % 2];
            for (int j = 0; j < pets[i].length; j++) {
                if (idx % 2 == 0) {
                    pets[i][j] = new Dog(args[idx++]);
                } else {
                    pets[i][j] = new Cat(args[idx++]);
                }
            }
        }
        for (int k = 0; k < pets.length; k++) {
            int notK = (pets.length-1) - k;
            for (int i = 0; i < pets[k].length; i++) {
                System.out.print(pets[k][i]);
                for (int j = 0; j < pets[notK].length; j++) {
                    if (pets[k][i].equals(pets[notK][j])) {
                       System.out.print("*");
                    }
                }
                System.out.println();
            }
        }
    }
}
```

```
public interface Pet {
    public String getName();
    public int getIDNumber();
    public boolean equals(Pet p);
}
```

```
public class Dog implements Pet {
    private String name;
    private int id;
    private static int num = 0;

    public Dog(String name) {
        this.name = name;
        id = num++ + name.length();
    }
    public String getName() {
        return name;
    }
    public int getIDNumber() {
        return id;
    }
    public String toString() {
        return name+"("+id+")";
    }
    public boolean equals(Pet a) {
        return id == a.getIDNumber();
    }
}
```

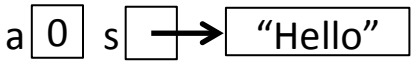
```
public class Cat implements Pet {
    private String name;
    private int id;
    private static int num = 2;

    public Cat(String name) {
        this.name = name;
        id = num;
        num *= 2;
    }
    public String getName() {
        return name;
    }
    public int getIDNumber() {
        return id;
    }
    public String toString() {
        return name+"("+id+")";
    }
    public boolean equals(Pet a) {
        return id == a.getIDNumber();
    }
}
```

Write your answer to the tracery question below, with each box representing one line of output (ignore any extra boxes).

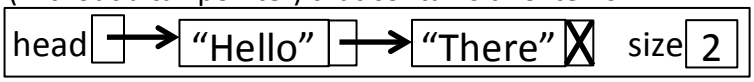
SECTION 8: OBJECT-ORIENTED MEMORY (14 pts total)

In this question, you will draw memory diagrams similar to the ones we used in lecture. Here are a few examples of code or objects and their corresponding memory diagrams:

```
int a = 0;
String s = new String("Hello");
```



A class representing a singly-linked list of strings (without a tail pointer) that contains two items.



Draw memory diagrams for each of the following. For any classes, clearly draw a box around the class to indicate all fields contained within that class, as shown above for the linked list.

8.1) (2 pts) A class representing an empty doubly linked list with sentinel nodes.

8.2) (4 pts) A class representing a singly linked list of strings (with a tail pointer) containing four items. Add a second class representing an `Iterator` object that is currently referencing the 3rd word and is capable of efficiently removing the 3rd word from the list. (Two classes total)

8.3) (4 pts) A class representing a student's academic record by name and PennID number with an `ArrayList` of classes the student has taken, represented as strings. (Two classes total)

8.4) (4 pts) A class representing a resizable two-dimensional matrix of integers that supports O(1) access to any elements and can add additional rows or columns of numbers in O(1) time.

That's all. Please go back and check your work. Also, do not discuss the exam on Piazza, as there are still several people waiting to take makeup exams. Have a great summer!!

CIS 110 Spring 2016 Final Exam Answer Key

SECTION 1: MISCELLANEOUS

1.1) **FALSE**
1.2) **FALSE**
1.3) **FALSE**
1.4) **TRUE**
1.5) **TRUE**
1.6) **STATIC**
1.7) **STATIC**
1.8) **STATIC**
1.9) **NON-STATIC**
1.10) **NON-STATIC**

SECTION 2: SORTING AND SEARCHING

2.1) Selection Sort: **15**
Insertion Sort: **14**
Merge Sort: **9 or 10 (depending on implementation)**
2.2) **a**
2.3) **c**
2.4) **b**
2.5) **d**
2.6) **d**
2.7) **e**

SECTION 3: ARRAYLISTS AND LINKEDLISTS

3.1

list.add(0,x):	**SinglyLL & DoublyLL**
list.add(x):	**All three**
list.get(list.size()/2):	**ArrayList**
list.remove(0):	**SinglyLL & DoublyLL**
total memory usage:	**ArrayList**

3.2

```
public class PairedSet<T> implements Set<T> {

      private List<T> list = new LinkedList();  // or ArrayList()

      public void add(T x) {
            Iterator<T> iter = list.iterator();
            int count = 0;
            while (iter.hasNext()) {
                  T item = iter.next();
                  if (item.equals(x)) count++;
                  if (count >= 2)
                        throw new RuntimeException(
                          "set already contains two copies of the item");
            if (count < 2) list.add(x);
      }

      public void clear() {
            list.clear();
      }

      public boolean contains(T x) {
            return list.contains(x);
      }
      public Iterator<T> iterator() {
            return list.iterator();
      }
      public boolean isEmpty() {
            return list.isEmpty();
      }
      public void remove(T x) {
            list.remove(list.indexOf(x));
      }
      public int size() {
            return list.size();
      }
}
```

SECTION 4: RECURSION

```
public void intersperse(String s) {
      intersperse(head,s);
}
private void intersperse (String s, Node n) {
    if (n.next == null) {
      return;
    }
    Node newNode = new Node();
    newNode.val = s;
    newNode.next = n.next;
    n.next = newNode;
    intersperse(s, newNode.next);
}
```

OR

```
public void intersperse(String s) {
      head = intersperse(head,s);
}
private Node intersperse (String s, Node n) {
    if (n.next == null) {
      return n.next;
    }
    Node newNode = new Node();
    newNode.val = s;
    newNode.next = intersperse(s, n.next);
    n.next = newNode;
    return n;
}
```

SECTION 5: LINKED DATA STRUCTURES

```
public void printRoute(int street) {
      if (street < 0) {
            throw new IllegalArgumentException("Invalid street number");
      }
      Station currExpress = depot;
      System.out.print(currExpress.street);
      while (currExpress.nextExpressStation != null &&
             currExpress.nextExpressStation.street < street) {
            currExpress = currExpress.nextExpressStation;
            System.out.print(", "+currExpress.street);
      }
      Station currLocal = currExpress.nextLocalStation;
      while (currLocal != null &&
             currLocal.street <= street) {
            System.out.print(", "+currLocal.street);
            currLocal = currLocal.nextLocalStation;
      }
 }
```

SECTION 6: DEBUGGING

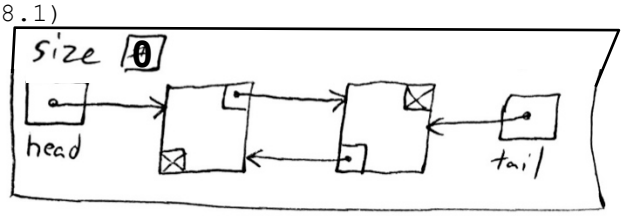
7	Iteration must start at depot	Change to "Station curr = depot";
10	Always saying station exists	Change != to ==
13	Breaks rest of list	Change to : "new Station(i, curr.nextLocalStation, null)"
45	Not protecting for nulls	Add condition "currLocal.nextLocalStation != null &&" to while
51	TODO marker should be removed	Change to "currExpress = currLocal"
58	Not deleting station	Change to "currLocal.nextLocalStation = currLocal.nextLocalStation.nextLocalStation"

SECTION 7: TRACERY

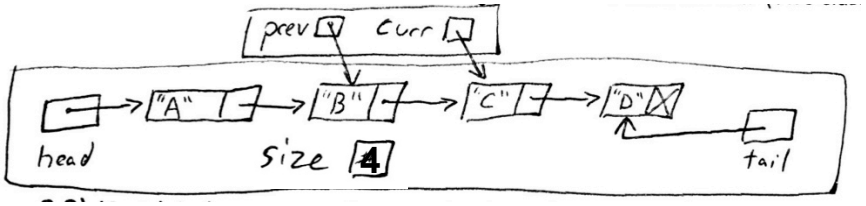
Fred(4)*
Georgia(2)
Rita(5)
Sammy(4)*
Bob(5)

SECTION 8: OBJECT-ORIENTED MEMORY

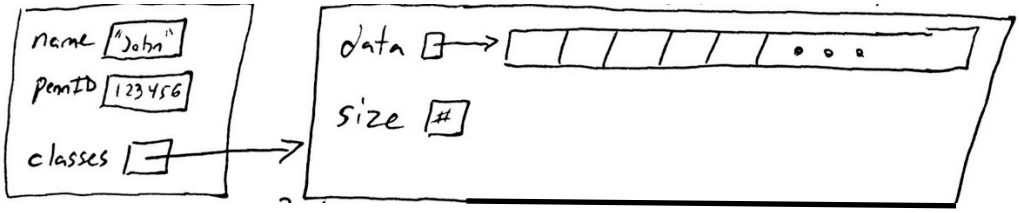
8.1)



8.2)



8.3)



8.4)