1.00 Introduction to Computers and Engineering Problem Solving

Final Examination - May 18, 2005

Name:	
E-mail Address:	
TA:	
Section:	

You have 3 hours to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary files have already been imported. Please write legibly.

Good luck!

Question	***Points***
Question 1	**/10**
Question 2	**/30**
Question 3	**/12**
Question 4	**/24**
Question 5	**/24**
Total	**/ 100**

Question 1. Short answer questions (10 Points)

1) Circle which of the following statements about Hashtables are true. The answers are the same for our lecture implementation or the Java Collections Framework class.

a. A Hashtable will keep Objects sorted in the order that they were added.

b. You can use an "int" as a key in a Hashtable.

c. A Hashtable has slower access to its elements when there are more chains (linked lists).

d. A Hashtable has slower access to its elements when each chain (linked list) is longer

2) Which digits, and in what order will be printed when the following program is run? Circle the correct answer.

```
public class MyClass {
       public static void main(String args[])
       {
           int k = 0;
           try{
                       int i = 5/k;
           } catch (ArithmeticException e1){
                       System.out.println("1");
           } catch (Exception e2){
                       System.out.println("2");
           }
           System.out.println("3");
       }
}
```

a. The program will not compile

b. The program will print 1 and 2 in that order

c. The program will print 1, 2 and 3 in that order

d. The program will print 1 and 3 in that order

3) Which of the following statement/s is/are *always* true? Circle all the correct answers

a. Subclasses must define all the methods that the super class defines

b. It is possible for a subclass to define a method with the same name and parameters as the super class

c. It is possible for a subclass to define a method with the same name but different parameters as the super class

d. It is possible for a subclass to inherit from two super classes

4) A thread can be alive, but not running due to which of the following reasons? Circle all the correct answers.

a. The `run`() method of the thread has completed

b. The `run`() method of the thread has not been invoked

c. The thread is asleep as a result of invoking its `sleep`() method

d. **The thread is not currently the highest priority thread**

5) Which of the following is a valid class definition of a class that cannot be instantiated? Circle all the correct answers.

a.
```
class MyClass {
   abstract void MyMethod();
};
```

b.
```
abstract class MyClass {
   void MyMethod() {}
}
```

c.
```
abstract class MyClass {
   void MyMethod();
}
```

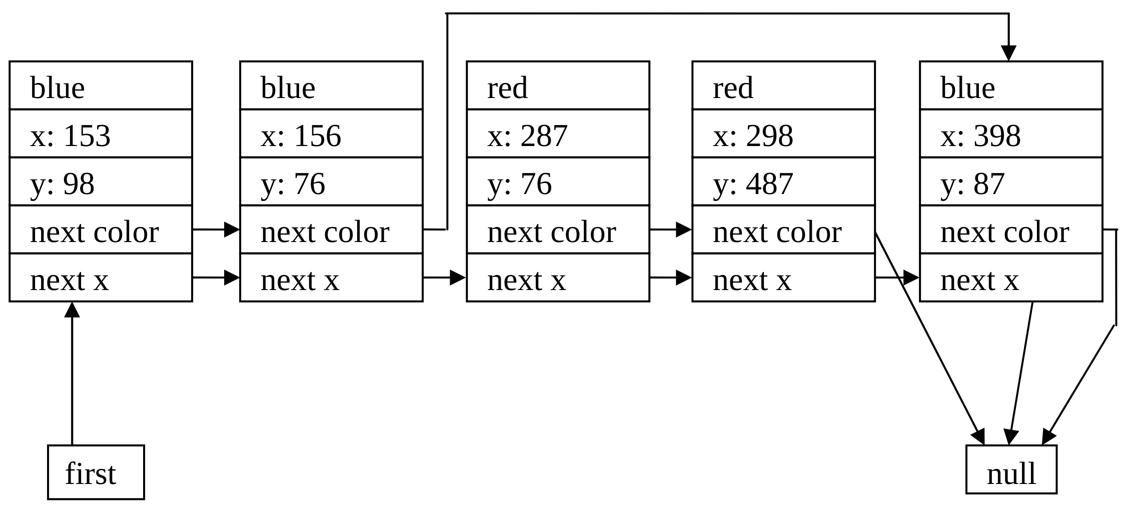
d.
```
abstract class MyClass {
   abstract void MyMethod();
}
```

Q1	**/10**

Question 2. Linked list (30 Points)

You have a group of points. Each point has an x coordinate, a y coordinate, and a color (either red or blue). The class below represents the points as a linked list. The points are linked via x coordinate and color. The list order is increasing in x, so the point with the smallest x coordinate is at the beginning of the list and the point with the largest x coordinate is at the end of the list. The points are also linked via color, so each point links to a point of the same color with the next highest x coordinate.

Here is an example:



The `ColoredPointList` data structure is populated in the `main()` method of the program. Assume that no two points have the same x coordinates.

The `ColoredPoint` class contains the color, x, y, nextX and nextColor fields. It is the "node" class for the linked list:

```
public class ColoredPoint{
      public String color;
      public int x;
      public int y;
      public ColoredPoint nextX;
      public ColoredPoint nextColor;

      public ColoredPoint() {}    // All members null, 0 by default

      public ColoredPoint(String c, int x_coord, int y_coord,
ColoredPoint nx, ColoredPoint nc){
            color = c;
            x = x_coord;
            y = y_coord;
            nextX = nx;
            nextColor = nc;
      }
      public void printMe(){
            System.out.println(color);
            System.out.println(x);
            System.out.println(y);
      }
}
```

```
// The ColoredPointList class is the 'link list' class:
public class ColoredPointList {
      private ColoredPoint first;
      public ColoredPointList(){
            first = null;
      }

      public ColoredPoint firstColor(String c){
```

// PART A
// Implement the firstColor() method which returns the first point in the list which is the
// color "c". Handle the special cases.

```
            ColoredPoint cp =first;
            while (cp != null){
                  if (cp.color.equals(c))
                        return cp;
                  else
                        cp = cp.nextX;
            }
            return null;

      }
```

```
      public static void main(String[] args) {
            ColoredPointList cpl = new ColoredPointList();
            …
            // cpl gets populated with a set of colored points
            …
```

```
// PART B
// Write code in main() that prints out the points, using the "printMe" method,
// in order of x coordinates, from smallest to largest.

                    ColoredPoint cp = cpl.first;
                    while (cp != null){
                              cp.printMe();
                              cp = cp.nextX;
                    }
```

```
// PART C
// Write code in main() that prints out the blue points, using the "printMe"
// method, in order of x coordinates, from smallest to largest. The code cannot
// visit any unnecessary red points
          cp = cpl.first;
          while (cp != null){
                if (cp.color.equals("blue")){
                    cp.printMe();
                    cp = cp.nextColor;
                }
                else
                    cp = cp.nextX;
          }
```

```
        }          // End main()
}                  // End class
```

Q2 **/30**

Question 3. Sorting (12 Points)

Your task is to help sort the students taking 1.00 by department. You are given a class `MITStudent`, which models a student. The data member `department` indicates what department the student is in.

```
public class MITStudent implements Comparable {
      public int department;
      public int compareTo(Object other) {
```

// PART A

// Implement the "compareTo" method which will be used when sorting the students.
// The students will be sorted in ascending order of department number. For example,
// a course 1 student comes before a course 2 student, a course 6 student comes before a course 10 student …

```
if (this.department < ((MITStudent)other).department)
      return -1;
if (this.department == ((MITStudent)other).department)
      return 0;
return 1;
```

```
      }
}
```

You realize that sorting the students by department is not good enough since there are too many students from one department. To solve this, you extend the "MITStudent" class for course 6 and override the "compareTo" method to first sort by course and then by program within the course.

```
public class Course6Student extends MITStudent implements Comparable
{
        // this data member indicates which program within course 6 the student is in.   If the
        // student is 6-1, program = 1; if the student is 6-2, program = 2; if the student is 6-3,
        // program = 3.
        int program;

        public int compareTo(Object other) {
```

// PART B

// Implement the "compareTo" method for a course 6 student whose program can be 1, 2, or 3.

// Students in program 1 (6-1) should come before students in program 2 (6-2), who should come

// before students in program 3 (6-3)

// Make sure to order the students by department number first and then by program. You may

// assume that, if a student's department= 6, he or she is a Course6Student.

```
if (super.compareTo(other) == 0) {
        if (this.program < ((Course6Student)other).program)
                return -1;
        if (this.program == ((Course6Student)other).program)
                return 0;
        return 1;
        }
else
        return super.compareTo(other);
```

```
        }
}
```

There is then a class with a main() method that reads students into an ArrayList and sorts them, which it can do easily since all your student objects implement the Comparable interface

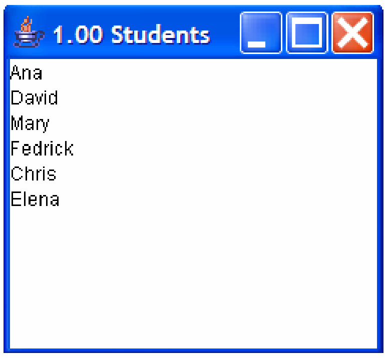
Q3	**/12**

QUESTION 4: Streams and Swing (24 Points)

The text file "`c:\studentName.txt`" contains the usernames of students in 1.00. However the file is not properly formatted as all the names appear in one single line as

Ana David Mary Fredrick Chris Elena ....

Your program is to read the usernames from this file and display them in a `JFrame` where each username appears in a single line, as shown in the figure below:



```
Ana
David
Mary
Fedrick
Chris
Elena
```

Figure 1

Read the skeleton of the class `ReadAndDisplay` given below that performs this task. You will complete the main method of the class.

```
public class ReadAndDisplay {
      public static void main(String[] args) {

            ArrayList listOfNames = new ArrayList();
            try {
                  File fin = new File("C:\\studentName.txt");
```

// Part A. In `main()`:
// 1. Read all usernames from the input file `fin`. They are on a single line.
// 2. Parse and store each username as a distinct element of the `ArrayList listOfNames`
// 3. Close any opened streams

```
FileReader fReader = new FileReader(fin);
BufferedReader reader = new BufferedReader(fReader);

String temp = reader.readLine();
StringTokenizer tokenizer = new StringTokenizer(temp);

while (tokenizer.hasMoreTokens())
{
   listOfNames.add(tokenizer.nextToken());
}

reader.close();
```

```
            } catch (IOException e) {
                  System.out.println("IO Exception");
            }
```

// Part B: Still in main():
// Display the `ArrayList` of all the usernames in a `JFrame` as shown in Figure 1.
// In the `JFrame`, create a `JTextArea`, add it to the `JFrame`'s `contentPane`.
// Add each username to the `JTextArea` using its append(String s) method
// On the frame: call `setSize()` or `pack()`; set its title; set the default close operation, set it visible

```
JTextArea area = new JTextArea();

for (int i= 0; i < listOfNames.size(); i++)
      area.append((String)listOfNames.get(i)+"\n");

JFrame frame = new JFrame();
frame.getContentPane().add(area);
frame.setTitle("1.00 Students");
frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```

```
      } // Closes public static void main
} // Closes public class ReadAndDisplay
```

Q4	**/24**

QUESTION 5. Search engines and adjacency matrices (24 Points)

A search engine represents the Internet using a directed graph: Each node in the graph corresponds to a web page and an arc coming *out* of a node represents a *different* node (or web page) that is hyperlinked *from* it. In the search engine's representation that we consider for this problem, both the nodes and arcs have weights associated with them.

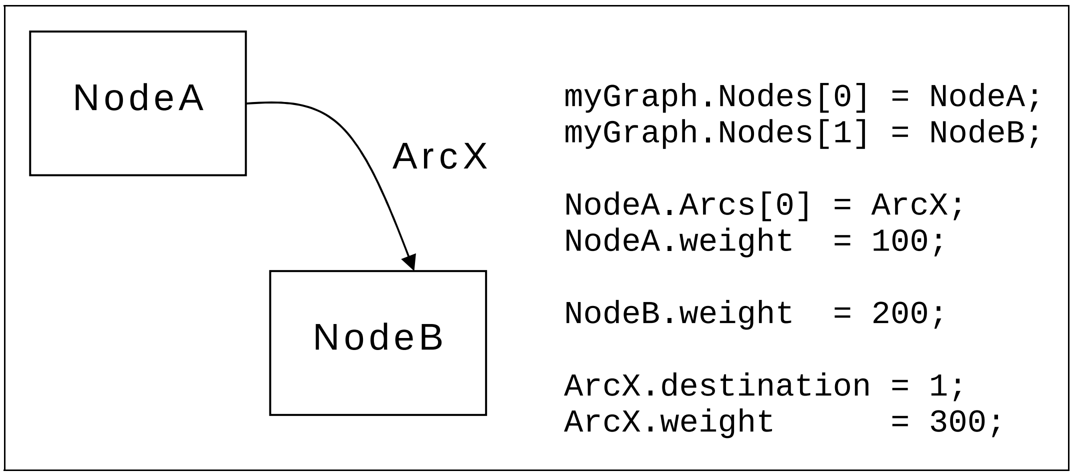
1. Each node has a positive weight.
2. If node *i* links to node *j*, the arc between *i* and *j* has a weight > 0
3. There is at most one arc between any two nodes: Even if web page *i* links to web page *j* multiple times, there is only one arc between node *i* and node *j*.

You are given a `Graph` class consisting of an array of `Node` objects, `Node[] Nodes`.

- Each `Node` object contains two fields:
  1. `int weight`: The weight associated with a particular node (web page)
  2. `Arc[] Arcs` : an array of `Arc` objects, representing only the arcs *going out* of a particular node. If a node does not have any arcs coming out of it, then the corresponding `Arcs` array is `null`
- Each `Arc` object contains two fields:
  1. `int destination`: The index of the destination node in the `Nodes` array of the graph
  2. `int weight`: The weight associated with the arc

Assume there is at least one `Node` and one `Arc` in the graph. *All variables have package access*.

An example of a graph is sketched below:



```
myGraph.Nodes[0] = NodeA;
myGraph.Nodes[1] = NodeB;

NodeA.Arcs[0] = ArcX;
NodeA.weight  = 100;

NodeB.weight  = 200;

ArcX.destination = 1;
ArcX.weight      = 300;
```

You are also given a `Matrix` class representing an *n* by *n* matrix. The `Matrix` class has three methods:

1. `public Matrix (int size)`: Takes the size of the matrix as input and initializes all elements to 0

2. `public int getElement(int row, int column)`: Returns the element at the given `row` and `column`
3. `void setElement(int row, int column, int value)`: Sets the `row` and `column` element of the matrix to the given `value`

Part A. Constructing the adjacency matrix

Write a method in the `Graph` class that computes and returns the adjacency matrix of the graph. The adjacency matrix $\mathbf{A}$ is defined as follows:

- The diagonal element of the matrix, $\mathbf{A}_{ii}$ contains the weight of node *i* in the graph.
- The off-diagonal element $\mathbf{A}_{ij}$ contain the weight of the arc from node *i* to node *j*. If there is no arc between node *i* and node *j* then $\mathbf{A}_{ij}$ =0.

For the graph illustrated in the figure, the adjacency matrix is given as

$$\mathbf{A} = \begin{bmatrix} 100 & 300 \\ 0 & 200 \end{bmatrix}$$

Assume the `Graph` has been constructed, so `Nodes` and `Arcs` exist. Classes `Matrix`, `Graph`, `Node` and `Arc` are all in the same package. Create the adjacency Matrix as defined above, by writing method `createAdjacencyMatrix` in class `Graph`:

```
public Matrix createAdjacencyMatrix()
{
      Matrix a = new Matrix(Nodes.length);

      for (int i = 0; i < Nodes.length; i++) {
         a.setElement(i, i, Nodes[i].weight);
         if(Nodes[i].Arcs != null){
            for (int j = 0; j < Nodes[i].Arcs.length; j++) {
            a.setElement(i, Nodes[i].Arcs[j].destination,
                  Nodes[i].Arcs[j].weight);
            }
         }
      }

      return a;
}
```

Part B Determining the structural symmetry of the graph

Write a `boolean` method to determine if the graph is structurally symmetric (i.e., if `Node[i]` links to `Node[j]` then `Node[j]` links to `Node[i]`). This can be done using the adjacency matrix $\mathbf{A}$ you computed in the previous part. Write this method in class `Graph`. The value of the node weights does not matter; you only need to distinguish whether there is an arc $\mathbf{A}_{ij} > 0$ or no arc $\mathbf{A}_{ij} = 0$.

```
public boolean isStructurallySymmetric()
{
      Matrix adjM = createAdjacencyMatrix();

      for (int i = 0; i < Nodes.length; i ++){
            for (int j = 0; j < Nodes.length; j++){
                  if (adjM.getElement(i,j) == 0 &&
                      adjM.getElement(j,i) != 0)
                      return false;
                  else
                  if (adjM.getElement(j,i) == 0 &&
                      adjM.getElement(i,j) != 0)
                      return false;
            }
      }

      return true;
}
```

Q5 /24