1.00/1.001

Introduction to Computers and Engineering Problem Solving

Spring 2010 - Final Exam

Name:	
MIT Email:	
TA:	
Section:	

You have 180 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary packages have already been imported. Good luck!

Question 1	**/ 20**
Question 2	**/ 15**
Question 3	**/ 15**
Question 4	**/ 20**
Question 5	**/ 20**
Question 6	**/ 10**
Total	**/ 100**

Question 1 - Inheritance and Interfaces

Congratulations! Hasbro wants to hire you to build a version of Monopoly® that people can play on their computers. Using your extensive 1.00 knowledge, you set out to complete this task.

Assume you initially start with a `Player` class that simply holds a player's name. The class isn't important for this question, but we provide the code just to be complete.

```
public class Player {
     private String name;
     public Player(String name){ this.name = name; }
     public String getName() {  return name; }
}
```

You know, from the game, that the game board has two types of squares. There are *property squares* (squares that represent a certain street or railroad) and there are *action squares* (such as "Go directly to jail" or "Pick a Chance card"). You decide to use inheritance to help model these board elements.

The first thing you do is create an `abstract class` called `Property`. You make it `abstract` because you realize there are many types of "property" in the game (players can own locations on the board, houses, and hotels). The `Property` class contains:

- A `private String` called `name`, that holds the name of this `Property`.
- A `private int` called `cost`, that holds the cost to buy this `Property`.
- A `private Player` object called `owner`. Initially the `owner` data member is set equal to `null`.
- Public `get()` methods for the three private data members, along with a `setOwner()` method.
- An `abstract` method called `costToOpponent()`, which returns a value that is used to calculate how much an opponent would owe the `owner` if he/she landed on this piece of `Property`.

```
public abstract class Property {

     private String name;
     private int cost;
     private Player owner;

     public Property(String name, int cost){
          this.name = name;
          this.cost = cost;
          this.owner = null;
     }

     public String getName() { return name; }
     public int getCost() { return cost; }
     public Player getOwner(){ return owner; }
     public void setOwner(Player buyer){ this.owner = buyer; }
     public abstract int costToOpponent();
}
```

1.a The next thing you need to do is create an interface for board squares. The interface is called `BoardSquare`. Every square on the board must have a position on the board (represented as an `int`). For example, Park Place is located at position 37 on a game board. Write a `BoardSquare` interface that guarantees that an object that implements `BoardSquare` will have a method that returns its location.

1.b You now put these parts together to create a concrete class called `PropertySquare`. This class is used to represent all the locations on the board that can be bought by Players. Write the `PropertySquare` class so that:

- It inherits from the abstract `Property` class.
- It implements the `BoardSquare` interface.

The value a `Player` must pay, if he/she lands on the square, is equal to half the cost of buying the `Property`. (e.g., If it costs 350 dollars to buy "Park Place" then a player must pay 175 dollars to the person who owns "Park Place").

```
public class PropertySquare

{

}
```

Assume that you have created another concrete class called `ActionSquare` that also inherits from the `BoardSquare` interface. All squares on the board are either of type `ActionSquare` or `PropertySquare`.

1.c You should also assume there is a class called `Board` that has methods that help run the game. In the `Board` class, you want to write a `public` method called `isUnownedSquare(BoardSquare square)`. The method should:

- Take in a `BoardSquare` object.
- Return a `boolean` value of `true` if the square is a `PropertySquare` and does not have an owner. Otherwise, return `false`.

```
public boolean isUnownedSquare(BoardSquare square)
{

}
```

Question 2: Exceptions

You are writing a program that analyzes some geometrical data. The data is stored as pairs of points and values in the following class.

```
public class PointValue{

     private double x;
     private double y;
     private double value;

     public PointValue(double x, double y, double v) {
          this.x = x;
          this.y = y;
          value = v;
     }

     public double getX() {return x;}
     public double getY() {return y;}
     public double getValue() {return value;}
}
```

There is also a class `Polygon`, which stores a sequence of points. It has a method which determines whether a point is inside the polygon or not.

```
public class Polygon{

     // implementation hidden: constructor, data members, etc.

     public boolean contains(PointValue p){
          // implementation hidden
          // returns true if p is inside the Polygon
          // returns false if it is not
     }
}
```

You need to write a class that will store some `PointValue` objects, but only if they are inside a specified `Polygon`. You decide to store the `PointValues` in an `ArrayList`. You will create a `PolygonList` class that will only store `PointValues` inside a specific `Polygon`, which is specified when the object is created. You should not extend the `ArrayList` class, but instead you should have an `ArrayList` inside your class. If someone tries to add a `PointValue` to the list that is not inside the `Polygon`, the add method should throw a `PointOutOfBoundsException`, defined as follows:

```
public class PointOutOfBoundsException extends Exception {
     public PointOutOfBoundsException() {super();}
     public PointOutOfBoundsException(String s) {super(s);}
}
```

Write class `PolygonList` below:

```
public class PolygonList {

     // 2.a Put data members here





     // 2.b Write the constructor here





     // 2.c The add method checks if the PointValue is actually
     // inside the polygon. If the point is inside the polygon,
     // add it to the list. Otherwise, throw a
     // PointOutOfBoundsException. Write the add method here
```

```
    // 2.d Write a get(int n) method that returns the specified
    // point from the ArrayList. The get method in ArrayList
    // throws an IndexOutOfBoundsException exception in some
    // cases. Use a try/catch block to catch that exception and
    // return null

}
//end of PolygonList class
```

Question 3 - Sorting and Hashing

3.a Many operations can be performed faster on sorted than on unsorted data. For which of the following operations is this the case? We do not count the sorting operation cost. Circle the ones which can perform faster on sorted data.

a. Finding the minimum value in the data

b. Computing an average of values

c. Finding the middle value (the median)

d. Finding the range of the data (maximum – minimum)

3.b True or False? (Circle one)

The running time of insertion sort $O(n^2)$ is always longer than the running time of quicksort $O(n \lg n)$ on the same set of elements.

TRUE FALSE

If two objects have the same return value from their hashCode() methods, they are equal objects.

TRUE FALSE

Quick sort uses pivot elements and partitioning.

TRUE FALSE

When sorting on an already sorted data, quick sort runs more quickly than insertion sort does.

TRUE FALSE

To sort a large set of randomly ordered data, quick sort on average runs faster than insertion sort.

TRUE FALSE

3.c We are going to place the following values into a bad hash table, where chaining is not used. Instead, our hashing is implemented by two hash functions. The main hash function is (value / 100) .The secondary hash function is used when two values are hashed into the same slot using the main hash function (i.e., a collision occurs). The second hash function is (value % 100) / 10. The following elements (all integers) are to be inserted into a hashtable of size 10.

`47, 90, 426, 140, 135`

3.c.1 Draw the table after the above elements are inserted.

0	1	2	3	4	5	6	7	8	9

3.c.2 The `HashTable` class below has a `find` method to check if a particular integer value has been stored in the table. Implement the `find(int i)` method so that it returns **`true`** if the integer value `i` is in the table, and **`false`** otherwise. Your method should not loop through the entire array holding the integers.

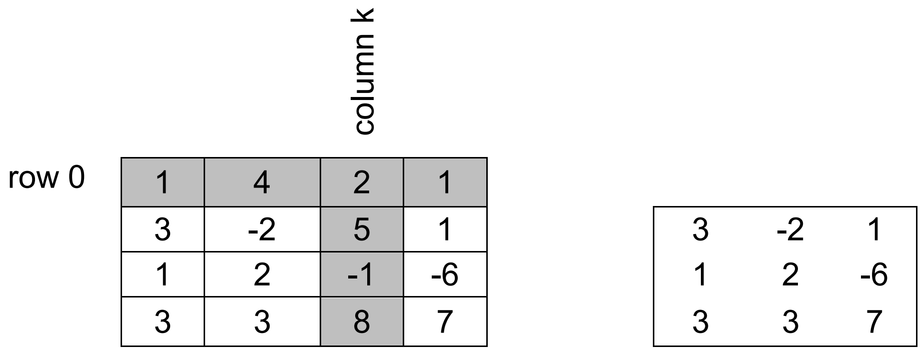
```
public class HashTable {

    private Integer[] table;

    public HashTable(int capacity){
        table= new Integer[capacity];
    }

    public boolean find (int i){



    }
}
```

Question 4 - Matrices and Recursion

In this question, you will implement a recursive method to compute the determinant of any square matrix, of any size.

4.a You first need to write a helper method, which will be used later by the method that computes the determinant.

The kth submatrix M_k of a square matrix M is obtained by deleting the first row and the kth column from M, as shown in the example below:



Matrix M

Submatrix M_k

The dimension of a square matrix is its number of rows or its number of columns, which are equal. From a square matrix of dimension n, you can therefore extract n submatrices, each of dimension (n-1).

You are given a `SquareMatrix` class. It is similar to the `Matrix` class from lecture, but can only be used to represent square matrices. For example, the `SquareMatrix` class has a single `getDimension()` method, while the `Matrix` class had the pair of methods `getRows()` and `getCols()`.

Complete the `getSubMatrix(int k)` method of the `SquareMatrix` class, which should return the kth submatrix of the square matrix on which it is invoked. **This first method does not have to be recursive**, and you can assume that 0 ≤ k ≤ n.

```
public class SquareMatrix {

     private double[][] data;

     public SquareMatrix(int n){data = new double[n][n];}

     public double getE(int i, int j){return data[i][j];}

     public void setE(int i, int j, double value)
     {
          data[i][j] = value;
     }

     public int getDim(){return data.length;}

     public SquareMatrix getSubMatrix(int k)
     {

     }
     // the getDeterminant()method (answer to next question)
     // goes here.
}
```

4.b The determinant |A| of a square matrix A of dimension n can be computed as:

$$|A| = \sum_{k=0}^{n-1} (-1)^k a_{0k} \; |A_k|$$

where:

- a_{0k} is the element of A located at row 0, column k.
- $|A_k|$ is the determinant of the submatrix A_k, which is extracted from A as described above.

For example, the determinant of a 3-by-3 matrix is:

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = (-1)^0 a_{00} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} + (-1)^1 a_{01} \begin{vmatrix} a_{10} & a_{12} \\ a_{20} & a_{22} \end{vmatrix} + (-1)^2 a_{02} \begin{vmatrix} a_{01} & a_{11} \\ a_{20} & a_{21} \end{vmatrix}$$

While the matrix A is of dimension n, each submatrix A_k is of dimension (n-1). Therefore, the determinant of a large matrix can be computed a sum of determinants of smaller matrices. When a matrix is small enough, its determinant can be computed directly. We know for example how to compute the determinant of a 2-by-2 matrix:

$$\begin{vmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{vmatrix} = b_{00} b_{11} - b_{01} b_{10}$$

And the determinant of a 1-by-1 matrix is its only element:

$$|c_{00}| = c_{00}$$

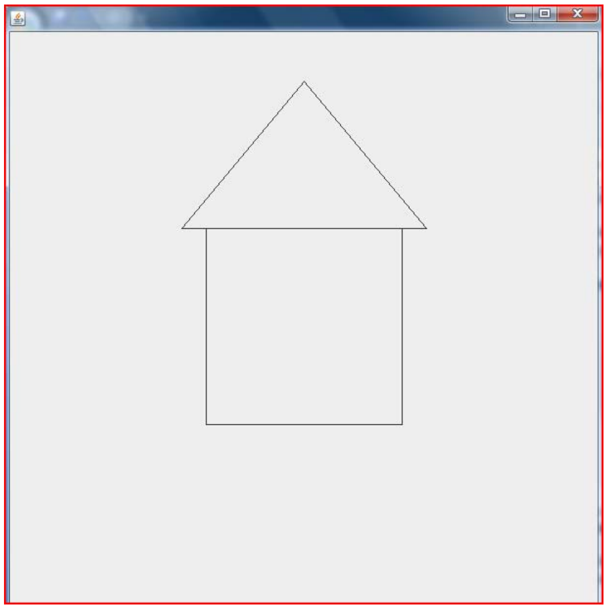
Complete the `getDeterminant()` method of the `SquareMatrix` class, which should return the value of the determinant of the square matrix on which it is invoked. **This method must be recursive**. Remember that a recursive method should directly return the result if the input is small enough to do so, but should call itself in order to solve for a larger input.

```
// This method is in the SquareMatrix class

public double getDeterminant()     // Must be recursive
{

}
```

Question 5 - Streams and Swing

It's time to get excited! This question combines two of your favorite subjects: streams and Swing. You will first be asked to write code that allows you to read and write files. You will then be asked a question about the GUI that uses this code.

Imagine that you are building a small drawing program, where you use `Line2D` objects to draw shapes on a GUI (such as the house in the screenshot below).



You quickly realize that it is important to be able to save your drawing, and load drawings into your program. You decide to store drawings as text files.

`Line2D` objects will be stored as a single line separated by commas:
Example: "`200,200,200,400`" defines a Line that starts at (200,200) and ends at (200,400).

So, the drawing of the house in the previous screenshot can be stored in a .txt file as:

```
200,200,200,400
400,200,400,400
200,400,400,400
175,200,425,200
175,200,300,50
425,200,300,50
```

You now need to create a specialized `JPanel` that can both load and save these files. We've provided a class called `StreamPanel` that extends `JPanel`. You will be asked to fill in the `loadDrawingFromFile()` and `saveDrawingToFile()` methods.

```
public class StreamPanel extends JPanel{

     // Name of file where drawings are saved to and loaded from
     private final String fileName = "examp_points.txt";

     // List of all the Lines to be drawn on the StreamPanel
     ArrayList<Line2D> lines = new ArrayList<Line2D>();

     public StreamPanel(){
          this.setPreferredSize(new Dimension(600, 600));
     }

     public void paintComponent(Graphics g){
          super.paintComponents(g);
          Graphics2D g2 = (Graphics2D) g;
          //draw all the lines
          for (Line2D l: lines) g2.draw(l);
     }

     // Loads in a drawing
     public void loadDrawingFromFile(){// FILL IN FOR PART 5A }

     // Saves the current drawing
     public void saveDrawingToFile(){// FILL IN FOR PART 5B }

     // Returns a Line2D object represented by String values
     private Line2D makeLine(String x1, String y1, String x2,
                                                  String y2){
          Double x1Val = Double.parseDouble(x1);
          Double y1Val = Double.parseDouble(y1);
          Double x2Val = Double.parseDouble(x2);
          Double y2Val = Double.parseDouble(y2);
          return new Line2D.Double(x1Val, y1Val, x2Val, y2Val);
     }

     // Takes in a Line2D object and returns a String
     // representation of it
     private String lineToString(Line2D l){
          String lString =  l.getX1()+","+l.getY1()+","+
                           l.getX2()+","+l.getY2()+"\n";
          return lString;
     }}
```

5.a In the space below, fill in the `loadDrawingFromFile()` method for the `StreamPanel` class.

- Be sure to use a BufferedReader to read in the Line2D objects from the file represented by the `fileName` data member.
- Store each Line2D object in the `lines` data member *(Hint: you may find the `makeLine()` method useful for creating Line2D objects)*.
- Catch and print to the console the possible IOException thrown by opening a stream.
- Remember this is a comma delimited file. Also, assume the file is formatted properly and that all values in the file are valid doubles.

```
public void loadDrawingFromFile(){

}
```

5.b In the space provided below, fill in the `saveDrawingToFile()` method for the `StreamPanel` class. The method should save all the Line2D objects in the `lines` list to the file represented by the `fileName` data member. Also, remember to catch the possible `IOException` thrown by opening a stream.

Hint: you may find the lineToString() method useful for creating a String representation of each line.

```
public void saveDrawingToFile(){

}
```

5.c Imagine that you have a GUI called `DrawingFrame` that consists of a JFrame with only a StreamPanel added to it. You've also been given a file whose contents look like this:

```
50,100,50,200
50,100,400,100
50,200,400,200
400,100,400,50
400,200,400,250
400,50,550,150
400,250,550,150
```

Sketch what the resulting `DrawingFrame` GUI would look like if you ran the GUI and loaded this file into the StreamPanel. The GUI size is *600x600*.

Question 6 - Data Structures

We have used stacks in class, but one thing you might not have known is that stacks can actually be implemented by using linked lists. In this question, you will build a stack class using linked lists.

We provide a simple implementation of a linked list and an interface for stack here. You will use these in the implementation of your stack.

```
public class LinkedList {

     //Adds an Object o to the end of the Linked List
     public void add(Object o)
     {    /* Implementation hidden */   }

     //Adds an Object o at index position to the Linked List
     public void add(Object o, int index)
     {    /* Implementation hidden */   }

     //Returns the Object stored at the index position
     public Object get(int index)
     {    /* Implementation hidden */   }

     //Removes the Object stored at the index position
     public void remove(int index)
     {    /* Implementation hidden */   }

     //Returns the number of Objects in the Linked List
     public int size()
     {    /* Implementation hidden */   }

     //Removes all the Objects from the Linked List
     public void clear()
     {    /* Implementation hidden */   }
}
```

```
public interface Stack
{
     public boolean isEmpty();
     public void push( Object o );
     public Object pop();
     public void clear();
}
```

6.a Complete the implementation of `LLStack` given below. You may ignore the case of the `pop()` method being called on an empty stack.

```
public class LLStack implements Stack {

     private LinkedList ll;

     // Finish the rest of the implementation here





}
```

6.b Indicate which data structure is the most appropriate representation for each of these.

	Hash Table	Stack	Queue	Tree	ArrayList	Briefly explain your choice
Waiting list for a class at Sloan						The people at the front of the waiting list get taken off before the people afterwards
Counting the number of occurrences of each word in an article						This is an association between a word and the number of occurrences
The order that you get dressed and undressed when it's -40 (F or C)						The first article of clothing you put on is the last article of clothing you take off
The species in the Plant Kingdom						The species can be modeled as a hierarchy
The shortest path from MIT to many points in Boston						Your path can branch in many ways depending on where you are going