CS 307 – Midterm 2 – Fall 2001

Name____________________________________

Last 4 digits of SSN / Student ID ____________

Class Unique ID ___________________________

Instructions:
1. There are 5 questions on this test. Each is worth 20 points.
2. You will have 2 hours to complete the test.
3. You may not use a calculator.
4. When code is required, write Java code.
5. You may not use any methods from the Java Standard Library other than the ones specifically mentioned on each question.
6. The style guide is not in effect.
7. Please make your answers legible.

1. Java mechanics and short answer questions. Write the answer to each question in the space provided. If code results in an error indicate if it is a compile error or runtime error.

For questions A through C consider the following classes:

```java
public class Vehicle
{
    private int iMyWheels;

    public Vehicle(int w)
    {
        iMyWheels = w;
    }

    public void setWheels(int w)
    {
        iMyWheels = w;
    }

    public int getNumWheels()
    {
        return iMyWheels;
    }

    public String toString()
    {
        return "wheels: " + iMyWheels;
    }
}
```
public class Truck extends Vehicle
{
    private boolean bMyAllWheelDrive;

    public Truck(int w, boolean d)
    {
        super(w);
        bMyAllWheelDrive = d;
    }

    public void setWheels(int w)
    {
        if( w < 6 )
            super.setWheels(w);
        else
            bMyAllWheelDrive = false;
    }

    public String toString()
    {
        String result = super.toString();
        if( bMyAllWheelDrive )
            result +=  "," + getNumWheels() + " wheel drive";
        else
            result +=  ", 2 +" wheel drive";
        return result;
    }
}

A. Consider the following code which appears in a class other than Vehicle and Truck:

Truck t = new Truck(4, true);
System.out.println( t.getNumWheels() );

What is the output of the this of code?

____________________________________________________

B. Consider the following code which appears in a class other than Vehicle and Truck:

Truck t1 = new Truck(6, true);
t1.setWheels( 8 );
System.out.println( t1.toString() );

What is the output of the this of code?

____________________________________________________
C. Consider the following code which appears in a class other than Vehicle and Truck:

```java
Vehicle[] v = new Vehicle[2];
v[0] = new Vehicle(4);
v[1] = new Truck(6, true);
System.out.println( v[0].toString() + " " + v[1].toString() );
```

What is the output of the this of code?

```
v[0] = new Vehicle(4);
v[1] = new Truck(6, true);
System.out.println( v[0].toString() + " " + v[1].toString() );
```

D. Consider the following method:

```java
public int wand(int n)
{
    if( n == 0)
        return 2;
    else
        return 1 + wand(n/2) + wand(n/3);
}
```

What is the output of the following statement which appears in the same class as wand?

```java
System.out.println( wand(6) );
```
E. Consider the following method:

```java
public int stone(int n)
{
    if( n == 0)
        return 0;
    else
        return n * n + stone( n - 1 );
}
```

What is the output of the following statement which appears in the same class as stone?

```java
System.out.println( stone(5) );
```


F. Consider the following method:

```java
public int cloak(int n)
{
    if( n == 0)
        return 0;
    else
        return 2 + cloak( n - 3 );
}
```

What is the output of the following statement which appears in the same class as cloak?

```java
System.out.println( cloak(10) );
```


G. What is the sum of the integers from 1 to 5000?


H. What is the Big O number of executions necessary to insert an item into an arbitrary position in an array of N items while maintaining the relative order of the original items?
I. Consider the following code:

```java
for(int i = 0; i < N * 2; i++)
    for(int j = N - 10; j > 0; j = j / 2)
        { ... }
```

What is the Big O number of executions of this code in terms of N?

J. Consider a LinkedList class with the following methods:

```java
public LinkedList()
public void addFront(int x)
public void addBack(int x)
public void removeFront()
public void removeBack()
public int getFront()
public int getBack()
```

This linked list is designed to store integers. What is the output of the following code?

```java
LinkedList ls = new LinkedList();
ls.addFront(10);
ls.addBack(12);
ls.addFront(7);
ls.addBack(2);
ls.addFront(12);
ls.removeBack();
ls.addBack( ls.getFront() * 2 );
ls.removeFront();
ls.removeFront();
ls.removeFront();
ls.removeFront();
ls.addBack( ls.getFront() );
System.out.println( ls.getFront() );
```
2. An area on the earth's surface can be modeled with a matrix of integers. Each element of the matrix represents a small area. Many types of information could be stored about the earth's surface, but in this problem only the elevation of the area is modeled and stored. An integer value represents the elevation of the area. Here is an example:

```
0 1 2 3 4 5 6 7 8 9
0 1 1 2 3 4 2 1 1 1
1 2 2 3 2 2 1 1 1 1
2 4 4 3 2 2 1 1 1 1
3 4 4 3 1 1 2 2 1 1
4 5 4 3 1 1 2 2 2 3
5 5 4 3 3 3 2 2 2 3
6 6 4 4 4 4 1 1 1 3
```

The flow of water in the area may also be modeled by looking at the various elevations. Assume a drop of water is placed in a cell. In our model water can flow to a neighboring cell if that neighbor's elevation is less than or equal to the elevation of the current cell. Also assume water can only flow to cells north, south, east, and west, not diagonally. Here is an interesting question: can water from a given cell flow out of the area? In other words can it reach one of the cells on the border or edge of the matrix. Obviously any cells on the border drain. Look at another cell, 3, 5. The elevation there is 2. Water can flow off the matrix from this cell via many possible paths, one of which is (2,5), (1,5), (1,6), (1,7), (0,7) then off the matrix. Note water in cell 3,4 cannot flow off the board, it is part of a depression or low area surrounding by cells of higher elevation.

Write a method to determine if water from a given cell can drain from the area represented by a matrix of integers. Water can flow to cells to the north, south, east, or west with the same or lower elevation. Once the edge of the matrix is reached (any of the cells in the first or last row, or first or last column) water is assumed to have successfully drained.

```java
/* pre: world != null, the matrix is rectangular meaning each row
   has the same number of columns
   post: returns true if there is a path from the cell
         specified to the edge of the matrix
   */
public boolean drains(int[][] world, int row, int column)
```

Complete the method on the following page
/* pre: world != null, the matrix is rectangular meaning each row has the same number of columns
   post: returns true if there is a path from the cell specified to the edge of the matrix */
public boolean drains(int[][] world, int row, int column)
3. Image filtering. Images on computers are represented as a collection of pixels (picture elements). An image can be stored as a matrix of pixels. The most important piece of information about a pixel is its color. One way of changing or altering pictures is to filter them, that is change the colors of pixels according to some algorithm. Write a method to alter the color of each pixel in an image to be the average of the colors of the 8 surrounding pixels.

The following class is used to represent the pixels. Note, unlike the first midterm a color is represented with a single integer in this class.

```java
public class Pixel {
    // default constructor
    // pre: none
    // post getColor() = 0
    public Pixel() {
        // pre: isValidColor(color) = true
        // post: getColor() = color
        public Pixel(int color) {
            // pre: none
            // post: returns true if value represents a valid color
            public boolean isValidColor(int value) {
                // pre: none
                // post: returns the integer value of this Color
                public int getColor() {
                    }
    }

You may only use the method listed from the Pixel class

Write the following method:

/*pre: org != null, org is a rectangular matrix meaning all rows have the same number of columns.
   No elements of org are null.
   post: returns a matrix of Pixels where each Pixel's color is the integer average of the surrounding pixels of the original image, org. If a cell only has 3 or 5 neighbors its color is changed to the average of those cells.
*/
public Pixel[][] filter(Pixel[][] org)

Complete the method on the following page.
/*
 *pre: org != null, org is a rectangular matrix meaning all rows have the same number of columns.
 *   No elements of org are null.
 *post: returns a matrix of Pixels where each Pixel's color is the integer average of the surrounding pixels of the original image, org. If a cell only has 3 or 5 neighbors its color is changed to the average of those cells.
 */

public Pixel[][] filter(Pixel[][] org)
4. Linked Lists. Write an insert method for a sorted, null terminated, doubly linked list. Consider the following class:

```java
public class SortableNode
{
    public Comparable data;
    public SortableNode next;
    public SortableNode prev;
}
```

Here are portions of the SortedLinkedList class:

```java
public class SortedLinkedList
{
    private SortableNode head;
    private SortableNode tail;
    private int mySize;

    public int getSize()
    // other methods not shown
}
```

The list is kept in ascending order. This means for a nonempty list the first element is the "smallest" according to the definition of the classes that implement the Comparable interface.

If the list is empty, mySize == 0, head and tail are both equal to null. If the list has 1 element, mySize == 1, head and tail are pointing at the same node. The prev field in the first node and the next field in the last node are both equal to null for a non empty list.

For the pictorially minded:

An empty list:

```
head       tail
null       null
```

A list with 1 node:
The / character indicates null

```
head
/      tail

A SortableNode Object
prev   data   next

A Comparable Object
```
A list with 4 nodes:

![Diagram of a list with 4 nodes]

Note, every class that implements the Comparable interface must implement a compareTo method. This means the method compareTo may be used.

```java
/* pre: other != null, calling object and other are of the same object type
post: return an int < 0 if calling object is less than other object, return 0 if the two objects are equal, and return an int > 0 if the calling object is greater than the other object */
public int compareTo(Comparable other)
```

Complete the following method on the next page. You may not call any methods other than compareTo and the SortableNode constructor.

```java
/* pre: val != null, val is same type as other Comparables in list post: a new node containing val is inserted into the list. The list is in ascending order. getSize() = old getSize() + 1 */
public void insert(Comparable val)
```

Complete the following method on the next page.
public void insert(Comparable val)

/* pre: val != null, val is same type as other Comparables in list
post: a new node containing val is inserted into the list. The
     list is in ascending order. getSize() = old getSize() + 1 */
5. Using Data Types: For this question you will work with 4 classes, ArrayList and Collections from the Java standard library, and two user defined classes, Swimmer and Time:

Here is the Swimmer class with only the public method signatures shown:

```java
public class Swimmer {
    public String getName()
    public Time get50FreeTime()
    public boolean equals(Object other)
}
```

Here is the Time class with only the public method signatures shown:

```java
public class Time implements Comparable {
    public int getMinutes()
    public int getSeconds()
    public int getHundreths()
    public int compareTo( Comparable other )
    public boolean equals(Object other)
}
```

The compareTo returns a negative int if the calling object is less than other, a 0 if they are equal, or a positive int if the calling object is greater than the other object.

**You may only use the following methods from ArrayList:**

```java
public ArrayList() Constructs an empty list.

public int size() Returns the number of elements in this list. (not the capacity).

public Object get( int index ) Returns the Object at the specified index. Preconditions: 0 <= index < size()

public void set( int index, Object element) Replaces the element at the specified index with the parameter element. Preconditions: 0 <= index < size(), element != null

public boolean add(Object element) Adds element to the end of the list. Returns true if the list changed due to the add. Note for ArrayList this will always return true. Preconditions: element != null

public void add(int index, Object element) Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices). Preconditions: 0 <= index < size(), element != null
```
public Object get(int index)
Returns the Object at the specified index. Note the index must be valid.
Preconditions: 0 <= index < size()

You may also assume the following method exists in the Collections class and you may use it if you wish. You are not required to use this method and there are possible solutions that do not employ the sort method.

/* pre: vals != null,
   post: vals is sorted in ascending order */
public static void sort(Comparable[] vals)

You are to complete a method to find the four Swimmer objects with the fastest four times in the 50 free and return them in an ArrayList. The four Swimmer objects in the returned ArrayList do not need to be in any particular order. You must first write out your algorithm you will implement and then write the code for that algorithm on the next page.

/* pre: swimmers != null, swimmers.size() >= 4, swimmers contains only Swimmer objects and none of the references in swimmers refer to the same Swimmer object. (Every element in swimmers is unique.)
   post: returns an ArrayList object with size() = 4 and that contains the swimmers with the fastest 4 times in the 50 free. */
public ArrayList getRelay(ArrayList swimmers)

Write out your algorithm below:
/* pre: swimmers != null, swimmers.size() >= 4, swimmers contains only Swimmer objects and none of the references in swimmers refer to the same Swimmer object. (Every element in swimmers is unique.)
   post: returns an ArrayList object with size() = 4 and that contains the swimmers with the fastest 4 times in the 50 free.
   */
public ArrayList getRelay(ArrayList swimmers)
   // write out your method in Java below