Test 1 Review: CompSci 100e

During the test, you may consult your four (4) sheets of notes and no other resources. You may not use any computers, calculators, cell phones, or other human beings. Your answers may refer to any program text supplied in lectures or assignments.

In writing code you do not need to worry about specifying the proper import statements. Assume that all libraries and packages we’ve discussed are imported in any code you write.

PROBLEM 1:  (Short Ones (12 points))

A. Give two distinct advantages of using an ArrayList (e.g. ArrayList<String> list) versus an array (e.g. String[] a).

B. What is printed as a result of the following code excerpt?

```java
int j = 12;
int i = j;
i = i - 4;

System.out.println(i);
System.out.println(j);
```

C. What is printed as a result of the following code excerpt?

```java
ArrayList<String> list1 = new ArrayList<String>();
ArrayList<String> list2 = list1;
list1.add("Baldwin");
list1.add("pass");

list2.add("luck");

System.out.println(list1);
System.out.println(list2);
```

D. Consider the following program.

```java
public class Pattern {
    public static void print(int n) {
        for (int i = -n; i <= n; i++) {
            for (int j = -n; j <= n; j++) {
                if (i == j)
                    System.out.print("A ");
                else if (i == -j)
```
System.out.print("B ");
else
    System.out.print(". ");
}
System.out.println();
}
}

public static void main(String [] args) {
    // Call print method with 1 as the argument
}
}

I. [2pts] Add a line to the main method above to call the print method with 1 as the argument.
II. [4pts] What is the result of calling print above?
III. [3pts] What is the result of calling print with 2 as the argument?

E. [6pts] For each of the following object-oriented programming terms, summarize the distinction between the two terms. Your answer should be brief.
   I. Comparing objects via == vs. .equals
   II. Classes that implement Comparable vs. Comparator
   III. class vs. object
   IV. the constructor vs. any other class method

F. (4) Suppose that b[] is an array of 100 elements, with all entries initialized to 0, and a[] is an array of N elements, each of which is an integer between 0 and 99. What is the effect of the following loop? (select all that apply)
   for (int j = 0; j < N; j++)
   b[a[j]]++;
   I. Sets b[0] to 0, b[1] to 1, b[2] to 2, etc.
   II. Sets b[0] to the number of 0s in a[], b[1] to the number of 1s in a[], etc.
   III. Sets b[0] to a[0], b[1] to a[0] + a[1], b[2] to a[0] + a[1] + a[2], etc.
   IV. Sets all entries of b[] to 1.
   V. Out-of-bounds array access (i.e. ArrayIndexOutOfBoundsException).
   VI. None of the above.

PROBLEM 2: (Maximum impact: 8 points)

A. [4pts] Complete the function allDifferent that returns true only if all three of the given values have different values. For example, a call to allDifferent(3,128,255) should return true; while the values allDifferent(128, 256, 128) should return false. Write allDifferent below - including the method header.
B. [4pts] Complete the function max3 that takes three numbers and returns the one with the largest value. Your implementation may not use any conditionals or loops and **must** call the built-in Java library method `Math.max` described below at least once and use its result in determining the result of this function.

```java
public class Math {
    /**
     * Returns the greater of two int values.
     */
    public static int max(int a, int b) { // Implementation omitted
        // Rest of class omitted
    }

    public int max3(int a, int b, int c) {
    }
}
```

**PROBLEM 3 : (Digits! (8 points))**

Write a method `oddDigits` below that returns whether a given integer has any digits that have odd values. Examples are below.

- `oddDigits(2468) → false` because 2, 4, 6, & 8 are even
- `oddDigits(111) → true` because 1 is odd
- `oddDigits(4096) → true` because 9 is odd

**PROBLEM 4 : (Close to Home (10 points))**

In this problem, you will write a method `closestToOrigin` that given arrays of x-coordinates and y-coordinates corresponding to points, returns the point that is the minimum distance away from the origin (0,0).

For example, given the arrays

```java
closestToOrigin({1, 0, 6, 3, 5}, {10, -12, 4, 4, 5})
```

should return the Point (3,4). The class `Point` is defined as follows:

```java
public class Point {
    public int x;
    public int y;

    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    public double distanceFrom(Point p){
        return Math.sqrt( (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y) );
    }
}
```
Complete closestToOrigin below.

```
public static Point closestToOrigin (int[] xVals, int[] yVals)
{
```

**PROBLEM 5 : (Apart (10 points))**

The input file is a text file that contains the information for a particular universe. The first value is an integer $N$ which represents the number of particles. Each line after that has two values indicating the the $x$- and $y$- coordinates of each particle.

Given an initialized Scanner for reading from the file, complete farthestApart so that it prints out the two particles which are the greatest distance apart. You are given a working distance function below.

For example, given the following file:

```
3
0.0 0.0
-1.0 -1.0
10.0 20.0
```

You should print

```
(-1.0, -1.0) and (10.0, 20.0) are farthest apart
```

Notes:

- In the event of ties, you can print any maximum pair.
- Order of the points printed does not matter.
- Do not worry about formatting your numbers (i.e. worrying about the number of trailing zeroes or significant digits).

```
public class Distance {
    // returns Euclidean distance between (x1, y1) and (x2, y2)
    public static double distance(double x1, double y1, double x2, double y2) {
        double dx = x2 - x1;
        double dy = y2 - y1;

        return Math.sqrt( (dx*dx) + (dy*dy) );
    }

    public static void farthestApart(Scanner in)
    {
```

**PROBLEM 6 : (Comparisons)**

The ABC organization has many chapters in every state of the United States. In order to keep track of all the member information an ABCChapter class has been created. An ABCChapter has three kinds of information, an array of all its member names, the state and the region of the state. We would like to be
able to compare ABCChapters and thus to implement Comparable. Some of its methods are shown below and on the next page.

Assume that each member is a string with ?Lastname Firstname? and that the members in each chapter are stored in myMembers in alphabetical order. For example one ABC Chapter with four members might be:

State = "NC"
Region = "Eastern"
MemberList = "Astrachan Owen", "Duvall Robert" "Forbes Jeff" "Rodger Susan"

Below is the class.

```java
public class ABCChapter implements Comparable<ABCChapter>{
    private String[] myMembers;
    private String myState;
    private String myRegion;

    public ABCChapter(String[] list, String state, String region) {
        myMembers = new String[list.length];
        System.arraycopy(list, 0, myMembers, 0, list.length);
        myState = state;
        myRegion = region;
    }

    /**
     * Return value that meets criteria of compareTo conventions.
     * @param chap is the ABCChapter to which this is compared
     * @return appropriate value less than zero, zero, or greater than zero
     */
    public int compareTo(ABCChapter chap) { // TODO implement this method
    }

    /**
     * Return true if this ABCChapter is the same as the parameter
     * @param o is the ABCChapter to which this one is compared
     * @return true if o is equal to this ABCChapter
     */
    public boolean equals(Object o) {
        ABCChapter chap = (ABCChapter) o;
        // TODO return correct value
    }

    /**
     * Return a good value for this ABCChapter to be used in hashing.
     */
    public int hashCode() { // TODO return a hash value
    }
}
```

A. [6pts] Implement the `equals` method for the ABCChapter class. Two ABCChapters are equal if they have the same state, the same region, and all the members are the same. Assume the members are stored in alphabetical order in `myMembers`.

```java
public boolean equals(Object o) {
    ABCChapter chap = (ABCChapter) o;
}
```

B. [6pts] Implement the `compareTo` method for the ABCChapter class. An ABCChapter is less than another ABCChapter if they have fewer members. If both chapters have the same number of members, then a chapter is less than another chapter if its state comes before it in alphabetical order. If the
states are the same then a chapter is less than another chapter if its region comes before the other
region in alphabetical order. If both the regions and the state are the same, then one chapter is less
than another chapter if the list of member names comes before the list of the other chapter member
names in alphabetical order.

public int compareTo(ABCChapter chap) {

C. [4pts] Implement the hashCode method for the ABCChapter class and explain why you think it is a
good hashCode function.

public int hashCode(){

PROBLEM 7: (Credit Crunch (12 points))

Credit card numbers can be validated using what’s called a checksum algorithm. The algorithm basically
works as follows, where the left-most digit has index 0 (as it does for strings) and the algorithm works from
left to right examining each digit of the credit card.

Accumulate a sum based on adding a value obtained from each digit of the credit card as follows, where each
\( k^{th} \) digit is examined starting with \( k = 0 \).

- If \( k \) is even, add the \( k^{th} \) digit to the sum.
- If \( k \) is odd, multiply the \( k^{th} \) digit by two. If the result is \( \geq 10 \), subtract 9. Add this computed value
to the sum.

If the resulting sum is divisible by 10, the credit card number passes the checksum test, otherwise the card
number isn’t valid. Credit card numbers are often 16-digits long, but they can be of any length.

Here’s an example for the card number 2543-2109-8765-4321 showing that it is a valid card number.

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>credit card number</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>value added</td>
<td>2</td>
<td>10-9</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>18-9</td>
<td>8</td>
<td>14-9</td>
<td>6</td>
<td>10-9</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>total</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>13</td>
<td>15</td>
<td>17</td>
<td>17</td>
<td>26</td>
<td>34</td>
<td>39</td>
<td>45</td>
<td>46</td>
<td>50</td>
<td>56</td>
<td>58</td>
<td>60</td>
</tr>
</tbody>
</table>

Since this is valid, the call isValid("2543210987654321") evaluates to true. However, the call
isValid("1234") evaluates to false since the computed sum is \( 1 + 4 + 3 + 8 = 16 \) which isn’t divisible
by 10.

Write the boolean-valued function isValid whose header is given on the next page.

In writing isValid you might want to use the function Integer.parseInt that converts a String to its int
equivalent, e.g., Integer.parseInt("9") evaluates to 9 and Integer.parseInt("17") evaluates to 17.

/**
 * Returns true if credit-card number’s checksum is valid
 * @param ccard contains only characters ‘0’, ’1’, ... ’8’, ’9’
 * @return true if ccard’s checksum is valid, otherwise returns false
 */
public boolean isValid(String ccard)
{

}

PROBLEM 8: (Like MemberCheck)
A health club chain allows its members to visit any of its many health club locations an unlimited number of
times per day. The only constraining rule is, a customer can only visit one health club location per day,
even though he or she may return to that location an unlimited number of times for the rest of that day.
Although the honor system has always worked quite well, the club wants to run some tests to see how many
people really follow the rules. You are to write a method that takes the entrance log files from all of the
different clubs (all logging the same day) and return a sorted list of the people who are not honest and went
to more than one health club location in the same day.
The entrances for each club are listed on a line. For example in the following file, there are three clubs.

JOHN JOHN FRED PEG
PEG GEORGE
GEORGE DAVID GEORGE

In Club 0, John visited twice, while Fred and Peg visited once. Peg and George each visited Club 1 once.
George also visited Club 2 twice, and David visited Club 2 once.

A. Complete the method, `attendeeList` below so that it returns a List of Sets corresponding to the
members who visited each club.

For the above file, the three element list below should be returned.

{[FRED, JOHN, PEG], [GEORGE, PEG], [DAVID, GEORGE]}

```java
List<Set<String>> attendeeList(Scanner in) {
}
```

B. Write a method, `whosDishonest` that takes the entrance log files data structure from `attendeeList` the
previous part and return a sorted list of the people who are not honest and went to more than one
health club location in the same day.

For the file on the previous page, the two element list `{GEORGE, PEG}` should be returned.

```java
List<String> whosDishonest(List<Set<String>> attendees) {
}
```

C. Briefly describe the big-Oh of the following call to your methods in terms of \( n \), the number of clubs,
and \( m \), the number of people who visit each club per day. Briefly justify your answer

```java
Scanner in = getScanner(); // open log file
List<String> list = whosDishonest(attendeeList(in)); // Give big-Oh for this line
```

**PROBLEM 9 : ( Sets (16 points))**

You are given the data from Dook University of the clubs and members of each club in the following format.
The data for each club is on a line. Each line has the name of all the members in the club, with members
separated by a colon.
The sample data below shows the members of three clubs.

Jeff Forbes:Hillary Rodham Clinton:Mary Lou Retton
Susan Rodger:Oprah Winfrey:Cay Horstmann:Mary Lou Retton:Owen Astrachan
Owen Astrachan:Oprah Winfrey:Mary Lou Retton

A. [10pts] Complete the Clubs method `allMembers` that given a Scanner as an argument that is initialized
to read from a valid data file, returns an ArrayList of all the people from Dook University in a club.
For example, using the datafile given above, the ArrayList would contain: Jeff Forbes, Hillary Rodham
Clinton, Mary Lou Retton, Susan Rodger, Oprah Winfrey, Cay Horstmann, and Owen Astrachan.
You may write and use helper functions if you find it helpful.
public class Clubs {
    public ArrayList<String> allMembers(Scanner in) {

B. [6pts] Describe how you would change allMembers to return an ArrayList of all the people who are in more than one club.

In the previous example, you would return the list containing: Mary Lou Retton, Owen Astrachan, and Oprah Winfrey.

You should be specific as possible in describing your changes. For example, you could specify exactly the lines of code that would change and what the new lines would be.

PROBLEM 10 : (Map)

public class FreqCompare implements Comparator<String> {
    private HashMap<String, Integer> myMap;

    public FreqCompare(String[] a) {
        // To be filled in..
    }

    public int compare(String a, String b) {
        int afreq = myMap.get(a);
        int bfreq = myMap.get(b);
        int diff = bfreq - afreq;
        if (diff != 0)
            return diff;
        return a.compareTo(b);
    }
}

A. Complete the constructor for FreqCompare below, so that the map is correctly filled in. that is, myMap should map from entries in the original array to the number of times they occur.

    public FreqCompare(String[] a) {

B. Describe how the ordering done by the Comparator will change if the line

    int diff = bfreq-afreq;

is changed to the following. Be brief and precise, not thorough.

    int diff = afreq-bfreq;

That is, given the following code:

    String a[] = // OMITTED
    Arrays.sort(a, new FreqCompare(a));

how would the contents of a change.
PROBLEM 11: (Over hill, Over dale, Overlap (12 points))

In this problem there is a lengthy explanation of the problem followed by three questions. You are modeling requests for one classroom. The requests are specified by the start-time and end-time for each class meeting in the room, e.g., (1, 3) means a two-hour class starting at one o'clock and (2, 5) means a three-hour class starting at two o'clock (ending at five o'clock). In this problem all times are after noon and before midnight. Class requests are considered intervals with a start time and an end time.

We want to keep rooms occupied and busy, but busy is the first priority. This means we want to schedule as many classes/requests as possible in the room. Consider the requests below.

(1,2) (1,3) (3,4) (1,4) (4,5) (3,5) (2,5) (6,7)

We could schedule the classes as follows. In both scenarios three classes are scheduled and the room is idle for one hour.

(1,2) (2,5) (6,7)

or we could schedule these classes:

(1,3) (3,5) (6,7)

However, if we schedule the following classes we get four classes using the room although there are two idle hours.

(1,2) (3,4) (4,5) (6,7)

It’s not possible to schedule more than four classes given the requests shown (though other schedules with four classes are possible too.)

You’ll write code to manipulate the interval requests. You’ll also write code to produce the list of class requests that keeps the room most busy, i.e., that schedules the most classes in the room. To do this you’ll implement the following algorithm in Java.

1. Sort the intervals by end time, breaking ties by the length of the interval (longer intervals are larger). The lowest end time comes first in the sorted array — see the sorted list above that starts with (1,2) and ends with (6,7) for an example.

2. Schedule the first (least) interval by adding it to an ArrayList being built that will contain the answer.

3. Now consider each of the remaining intervals in sorted order, call the interval being considered the current interval.

   (a) If the current interval overlaps with the last interval added to the ArrayList, skip it.

   (b) Otherwise, if the current interval doesn’t overlap, add the current interval to the ArrayList (it’s now the last-added interval).

The definition for the class Interval you’ll use in this problem is provided on the next page. You’ll implement several methods using this class.

```
private static class Interval implements Comparable<Interval>{
    private int myStart;
    private int myEnd;
```
public Interval(int start, int end){
    myStart = start;
    myEnd = end;
}
/**
 * Returns value < 0 if this interval less than o,
 * returns value > 0 if this interval greater than o,
 * and returns 0 if this interval is equal to o.
 */
public int compareTo(Interval o) {
    int diff = myEnd - o.myEnd;
    // complete code here in Part B
}
/**
 * Returns true if and only if this interval overlaps Interval i.
 * @param i is the interval considered for overlap with this one
 * @return true if and only if the intervals overlap
 */
public boolean overLaps(Interval i){
    if (i.myEnd <= myStart || myEnd <= i.myStart) return false;
    return true;
}
public String toString(){
    return "["+myStart+","+myEnd+"]";
}

A. [4pts] Write the method idleTime which returns the total time between all intervals in a sorted, non-overlapping array of intervals. For example, consider these arrays:

{ (1,3) (3,5) (6,7) }
{ (1,2) (3,4) (4,5) (6,7) }

The first array has one hour of idle time between (3,5) and (6,7). The second array has two hours of idle time.

/**
 * Assuming list is sorted, and intervals don't overlap
 * returns the number of hours not in any interval in list,
 * only hours between the start of list[0] and the end of
* list[list.length-1] are considered.
* @param list is the sorted list of non-overlapping intervals
* @return the number of idle time, time not in any interval
*/
public int idleTime(Interval[] list) {
}

B. [4pts] Implement the compareTo method in the class Interval so that intervals are compared as specified earlier in this problem: an Interval a is less than an interval b if and only if a ends before b ends or they end at the same time and the length of interval a is less than the length of interval b.

private static class Interval implements Comparable<Interval> {
    private int myStart;
    private int myEnd;

    /**
     * Returns value < 0 if this interval less than o,
     * returns value > 0 if this interval greater than o,
     * and returns 0 if this interval is equal to o.
     */
    public int compareTo(Interval o) {
        int diff = myEnd - o.myEnd;
        return diff;
    }
}

C. [4pts] Write method mostBusy which returns an array of intervals which contains the maximal number of non-overlapping intervals from the parameter list. Assume the Interval class compareTo method is implemented correctly (so the sort call below works). Use the algorithm at the beginning of this problem to create a maximal array of non-overlapping intervals. The ArrayList that is part of the algorithmic description is initialized in the starter-code provided, and the ArrayList is converted to an array in the return statement. Add code as necessary to implement the algorithm.

/**
 * Returns array containing maximal number of non-overlapping from list
 * @param list is an array of intervals
 * @return array containing maximal number of non-overlapping intervals
 */
public Interval[] mostBusy(Interval[] list) {
    ArrayList<Interval> maximal = new ArrayList<Interval>();
    Arrays.sort(list);

    return (Interval[]) maximal.toArray(new Interval[0]);
}