COMP 110 FINAL EXAM Fall 2000 Last Name first _____

Circle Section 1 2 3 4 5 6

Open portfolio.
Open mind.
Open book.

In this exam we are mainly interested in your approach,
your general method of doing the problems.

You need not complete the entire exam in all its details.
Considerable partial credit will be given for a good start on a problem,
so it pays to show all of your work.
If you do any rough work, show it on the back side of the previous page.

Note carefully what is asked:
an algorithm can be in flowchart, data flow, pseudo-code or flowBlock form,
a method (function or routine) must have slots, parameters, passed properly,
a program should be complete, written in Java preferably, with data declarations.
ReUse!!! Feel free to use any previously created procedures, don't recopy them!
For speed, you could use short names for boxes, sub-programs, etc.
For speed, you need not echo inputs, or prompt carefully.

Remember: Many, First may be worst, Begin BIG, ReUse, Package properly, Picture it.

Please do your work on these given pages.

The problems are approximately equally weighted,
but they are not all of equal difficulty.

NOTE: you need do only 8 of the 10 problems; circle below the numbers of the two problems which you do not wish to have graded; otherwise we will omit the problems having the lowest scores.

We wish you success.

1. Quick
2. Leap
3. Plot
4. Money
5. Parse
6. Analysis
7. Sort
8. Trace
9. Min
10. Face

1. Quick Questions:

Differences

Indicate briefly one significant difference among the following pairs of concepts;
Do not just define each concept, and do not give examples; compare it to the other.

a. string, char

b. while, for

c. function, routine

d. object, class

e. variable (box), parameter (slot)

Indicate what each of the following loop statement produces:

f. `for (int i = 0; i < 4;  i++)  System.out.print (i + " ");`

g. `for (int i = 0; i >= 0; i++)  System.out.print (i + " ");`

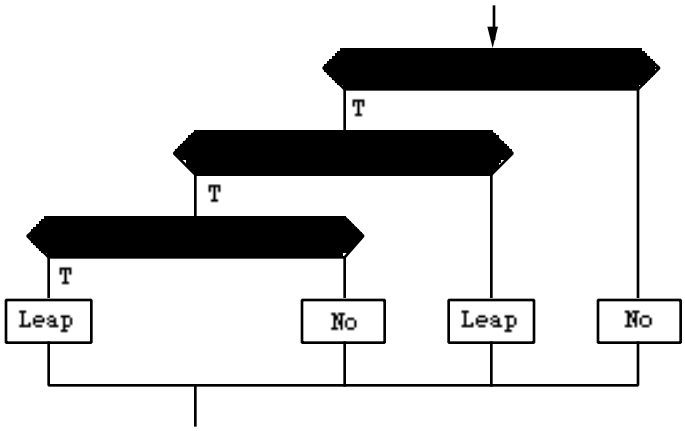
h. `for (int i = 0; i >= 0; i--)  System.out.print (i + " ");`

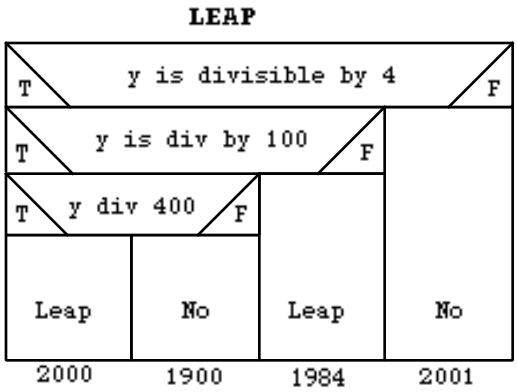
i. `for (int i = 0; i != 0; i++)  System.out.print (i + " ");`

j. `for (int i = 0; i < 4 ; i++); System.out.print (i + " ");`

2. Leap Again

The given diagrams show one way of many to determine if a given year y is a leap year.





The following three logical variables (boxes) and formulas involve any year y and whether it is divisible by some constant.

```
   boolean yD4, yD100, yD400, leap;
   int y;  // year

   yD4   = ( (y % 4) == 0);  // year y is divisible by   4
   yD100 = (y % 100) == 0 ;  // year y is divisible by 100
   yD400 = (y % 400  == 0);  // year y is divisible by 400
```

First, use these to code the given diagram exactly;
do not change the structure of this nested Choice form.
Then package this code into a proper function method.
Finally, write an expression for leap as one boolean condition.

```
class ManyLeaps {

   public static void main (String args[]) {
      System.out.println ( "2000" + isLeap (2000) );
      System.out.println ( "1900" + isLeap (1900) );
      System.out.println ( "1950" + isLeap (1984) );
      System.out.println ( "2001" + isLeap (2001) );
   }//end Routine main

   public static boolean isLeap (int y) {
      boolean yD4, yD100, yD400, leap;
   // Does determine if a year is a leap year

      yD4   = ( (y % 4) == 0);  // year y is divisible by   4
      yD100 = (y % 100) == 0 ;  // year y is divisible by 100
      yD400 = (y % 400  == 0);  // year y is divisible by 400

      if (yD4)                  if (yD4) {
         if (yD100)                if (yD100) {
            if (yD400)                if (yD400) {
               leap = true;              leap = true;
            else                      }else{
               leap = false;             leap = false;
                                      }//end if
         else                      }else{
            leap = true;              leap = true;
                                   }//end if
      else                      }else{
         leap = false;             leap = false;
                                }//end if

      return leap;

   }//end Function isLeap

}//end Class ManyLeaps
```

3. Plot Rule of 72

A function doubleTime (percent) returns the integer number of years at which an amount doubles when compounded at the given yearly percent rate. I.e. doubleTime(5) will return a value of 15.

You are not to define it, but you are to use it to plot a bar graph which demonstrates the "Rule of 72", (the product of years and percent is approximately 72). For example, for each rate from 1 to 99, a bar is constructed of length year*percent, which is approximately of length 72. The deviation from this length shows the approximation visually.

You may assume that you have a routine outRow (many, str) which prints out a string str many times. Avoid long output lines (use ourRow and fors instead).
You need not be concerned about "packaging" this code as a method.

```
%   ---------+---------+---------+---------+---------+---------+---------+--------+
1   **********************************************************************
2   **********************************************************************
3   **********************************************************************
4   **********************************************************************
5   **********************************************************************
6   **********************************************************************
7   **********************************************************************************
etc
```

```
class Rule72 {

   public static void main (String args[]) {
   // Shows accuracy of the "Rule of 72"
   // rate * time approximates 72

      System.out.print ("   ");
      outRow (8, "---------+");

      for (int rate = 1; rate < 7; rate++) {
         System.out.print (rate + " ");
         outRow ( rate*doubletime (rate), "*");
      }//endFor
   }//endRoutine main

   public static void outRow (int many, String str) {
   // Draws a row of many strings (ReUsed from previous)
      for (int i = 0; i <= many; i++)
         System.out.print (str);
      System.out.println ();
   }//end Routine outRow

// public static int doubletime (int percent) {
// Don't do it: Use it! ReUse it just like outRow above

}//End Class Rule72
```

4. Money Class

For the following Money class, create the following methods:

a.plus (b) which does return a new Money object of the sum of two amounts
a.lessThan(b) which does indicate whether Money amount a is less than b.
a.round () which does round off to the next nearest dollar value.

These are used in the following main routine.
Then trace the main routine indicating the output.

```
class Money {

   private int dollars;
   private int cents  ;

   public Money (int d, int c) {
      dollars = d;
      cents   = c;
   }//end Constructor Money

   public int toCents() {
      return 100*dollars + cents;
   }//end Function toCents

   public boolean isValid() {
   // Does tell if this is valid
      return (cents >= 0) && (cents < 100);
   }//end Function isValid

   public static void main (String args[]) {
      Money a,b,c;
      a = new Money (1234, 56);
      b = new Money (9876, 54);
      c = a.plus (b);
      if (c.isValid()) c.round();
      System.out.println (c.dollars);
      System.out.println (a.lessThan (b));
   }//end routine main

}//end Class Money
```

Solution:

```
   public void round () {
      if (cents > 50)
         dollars ++;
      cents = 0;
   }//end routine round

   public boolean lessThan (Money that) {
      return (this.toCents() < that.toCents());
   }//end boolean function lessThan


   public Money plus (Money b) {
   // Does result in this added to amount b
      int sum = this.toCents() + b.toCents() ;
      return new Money (sum / 100, sum % 100);
   }//end Function plus
```

5. Parse Name

Given a name as a string in directory form (written with last name first followed by first name separated by a comma and a space), write code which outputs the name in the proper form, first name first.
For example, the string **"Onymous, Ann"** would print **"Ann Onymous".**
Then package this as a function method named properName that returns a string in the proper form.

```
class Parsename {

   public static void main (String args[]) {
      String name = "Onymous, Ann";
      System.out.println (properName(name) );
   }//end Routine main

   public static String properName (String s) {
      int len = s.length();
      int pos = s.indexOf(',');
      String last = s.substring (0, pos);
      String first = s.substring( pos+2, len);
      return (first + " " + last);
   }//end Routine outName

}// end class Parsename
```

8. Trace Convoluted Array

Trace the following code whch operates on the given two arrays a,b,
of type int and size n.

Package this program as a method con (a,b, n) in Java.

```
a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 0;

b[0] = 4;
b[1] = 5;
b[2] = 0;
b[3] = 0;

n = 3;
for (int j = 0; j <= n; j++ ) {
   int s = 0;
   for (int i = 0; i <= j; i++)
       s = s + a[i] * b[j-i];
   System.out.println (s);
}
```

Trace (in 2D) yields 4, 13, 22, 15.

6. Analyse Account

For the given Account class, compare the following two main routines (for correctness, generality, speed, style, accuracy, purity, etc.). Write your analysis on the facing page (not on the back of this one). The routines find the time when two accounts grow to the same amount.

```
import java.io.*;
class Account {
   private String identity ;
   private double balance  ;
   private double rate ;

/* Constructor*/
   public Account (String i, double b, double r ) {
      identity = i ;
      balance = b ;
      rate = r ;
   }//End Constructor Account

/* Function*/
   public double theBalance () {
      return balance;
   }//EndFunction theBalance

/* Routine*/
   pubic void compoundedBy (int n) {
      while (n > 0) {
         balance = balance * rate + balance ;
         n--;
      }//endWhile
   }//endRoutine compoundedBy

}//EndClass Account
```

```
public static void main (String[] args) throws IOException {
   Account first, second;
   int years = 20;
   while (true) {
      first  = new Account ("Jack", 10000.00, 0.04);
      second = new Account ("Jill",  5000.00, 0.06);
      first.compoundedBy  (years);
      second.compoundedBy (years);
   if ( first.theBalance() < second.theBalance() ) break;
      years ++;
   }//endWhile
   System.out.println ("The duration is " + years );
}//EndRoutine main
```

```
public static void main (String args[]) throws IOException {
   Account first  = new Account ("Jack", 10000.00, 0.04);
   Account second = new Account ("Jill",  5000.00, 0.06);
   int years = 0;
   while (first.balance >= second.balance) {  //
      first.compoundedBy  (1);
      second.compoundedBy (1);
      years ++;
   }//endWhile
   System.out.println ("The duration is " + years );
}//EndRoutine main
```

7. BreakUp swapSort

Most of the sorts considered so far involved the sort calling another method (SelectSort called maxPosn, and CountSort called countGreater). This packaging also makes the sorts clearer, for each method involves only one loop rather than nested loops.

SwapSort, however, was not broken up in such a way. Now you break up the original "slow" swapSort by calling a method swapOnIncrease(b,m) which is used as follows in JJ. Do swapOnIncrease in Java preferably.

```
Routine swapSort (A, n) is public
   Slot A ofType int[] -- array
   Slot n ofType int   -- size
-- Does simple swap sort the slow way
   Box  i ofType int
   Set i = 0
   Repeat
   ExitOn (i == n-1)

      Call swapOnIncrease with (A,n)

      Inc i by 1
   EndRepeat -- of i
EndRoutine swapSort
```

```
class SwapSorting {

   public static void main (String args[] ) {
      int arr[] = {1,9,6,8,3};
      int n = 5;
      swapSort (arr,n);
      for (int i=0; i<n; i++)
         System.out.println (arr[i]);
   }//end routine main

   public static void swapOnIncrease (int[] b, int m) {
      for( int j=0; j < m-1; j++)  {
         if ( b[j] < b[j+1] )   {
            int temp;
            temp   = b[ j ];
            b[ j ] = b[j+1];
            b[j+1] = temp;
         } // end if
       } // end for j
   }//end routine swapOnIncrease

   public static void swapSort (int A[], int n) {
      for( int i=0; i < n-1; i++ ) {
         swapOnIncrease (A,n);
       } // end for i
   } // end of swap sort method (short eh?)

}// end class swapSorting
```

9. Many Mins in Array

Create one of the following methods to operate on an array named arr of n integer percents, (ranging from 0 to 100), having many repeated values.

MinPositions, which outputs the positions of all of the possibly many minimum values.
or
MinCounts, which returns the number of times that the minimum appears.

For the other method specify it's signature (just the header, declaration part);
i.e. show the definition without the body part.

Pass parameters properly. Use Java. ReUse.

```
class ManyMins {

   public static void minPositions (int[] arr, int n) {
   // Does output positions of all min values
      int min = minArray (arr, n); //ReUse!!
      for (int p = 0; p < n; p++)
      if (min == arr[p])
         System.out.println (p);
   }//end Routine minPositions

   public static int minCounts (int[] arr, int n) {
   // Does return count of all min values
      int min = minArray (arr, n);
      int count = 0;
      for (int p = 0; p < n; p++)
      if (min == arr[p])
         count ++;
      return count;
   }//end Routine minCounts

   public static int minArray (int[] arr, int n) {
      int min = 100;
      for (int p = 0; p < n; p++)
         if (min > arr[p])
             min = arr[p];
      return min;
   }//end function minArray

   public static void main (String[] args) {
      int arr[] = {2,3,4,2,2,2,3};
      int m = 7;
      System.out.println ("min occurrences = " + minCounts(arr,m) );
      minPositions (arr, m );
   }//end routine main

}//end Class manyMins
```

Problem 10. Face Lift

Given a happy face, shown as a 2D array of pixels (480 by 640) show part of the code which converts the mouth from sad into glad.