Arrays
Bubble Sort
Binary Search
&
Recursion

This is the last unit and WILL NOT be covered in strict order

/* PROBLEM: Sort a list of integers.

SUBPROBLEM: How do we store the integers.
Answer: In an Array.

-- Think of an Array as a list of objects all of the same type.

int [] list = new int [5];

int list = new int [5];  <-- indicates primitive type integer
int [] list              <-- indicates it is an
                         ARRAY of integers
int [] list = new        <-- Name we choose to give the Array
int [] list = new int [5]  <-- An array is an OBJECT.
                          It is created with
                          the operator "new" which
                          tells the Java Interpreter to
                          allocate memory (from the
                          Heap) at run time.
int [] list = new int [5];  <-- Finally it is 5 integers that
                          we need space allocated for.

NOTICE: list is a variable which contains the address of the 5 integers.
list[0]  <-- is the first integer
         in the array. Computers
         start counting at 0.
list[4]  <-- is the last, i.e., 5th integer
         in the array
/* Illustration  Recall from unit 2: Problem: find the average of 10 integers.
   With arrays, the program can "remember" all 10 integers.

Declaring/defining an array
Accessing the member of the array

The declaration:  int[] addend = new int[HOW_MANY];
is telling the computer to create HOW_MANY == 10 variable-boxes (which are indexed) as
   addend[0],  addend[1],  addend[2],   ...   ,  addend[9]

NOTE:  an array is an OBJECT.  Therefore we create the actual array using the"new".

NOTICE:  the array members start at index 0 and go to HOW_MANY - 1.
*/

// file == AverageArray.java

import utils.SavitchIn;
class AverageArray
{
    public static void main (String[] args)
    {
        final int HOW_MANY = 10;
        int count, sum = 0;

        int[] addend = new int[HOW_MANY];

        System.out.println
            ("Type " + HOW_MANY + " integers, one per line!");

        for (count = 0; count < HOW_MANY; count++)
        {
            System.out.print("ADDEND #" + count + ": ");
            addend[count] = SavitchIn.readLineInt();
        }

        for (count = 0; count < HOW_MANY; count++)
        {
            sum = sum + addend[count];
        }

        System.out.println ("The truncated average is ... " + sum/HOW_MANY);

        System.out.println("The numbers in reverse order are: ");

            for ( count = HOW_MANY-1; count >= 0; count-- )
                System.out.print( addend[count] + " ");
        System.out.println();
    }
}
// A more Oopish way

// Parameter Passing an array to a method

import utils.SavitchIn;

class AverageArrayOop
{
    double averageThem ( int [] arrayOfInt )
    {
        int actualSize = arrayOfInt.length;
        int counter, sum = 0;
        for (counter = 0; counter < actualSize; counter++)
            { sum = sum + arrayOfInt[counter]; }
        return (double)sum/actualSize;
    }

    void showInReverse ( int [] arrayOfInt )
    {
        int actualSize = arrayOfInt.length;
        for (int i = arrayOfInt.length - 1; i > 0;  i-- )
            System.out.print( arrayOfInt[i] + " ");
        System.out.println();
    }

    public static void main (String[] args)
    {
        AverageArrayOop A = new AverageArrayOop();
        int[] example1 = {1,2,3};
        System.out.println( A.averageThem( example1 ) );
        int[] example2 = {-1,2,-3,4,-5,6,-7,8,-9,10};
        System.out.println( A.averageThem( example2 ) );

        final int HOW_MANY = 10;
        int count,          // for loop index
            sum = 0;        // to hold the sum

        int[] addend = new int[HOW_MANY];

        System.out.println
            ("Type " + HOW_MANY + " integers, one per line!");
        for (count = 0; count < HOW_MANY; count++)
            { System.out.print("ADDEND #" + count + ": ");
                addend[count] = SavitchIn.readLineInt();
            }
        System.out.println( A.averageThem( addend ) );
        System.out.print(".. Here they are reversed: ");
        A.showInReverse (addend);
    }
}
Back to the Sorting Problem

**SUB-PROBLEM:** How do we actually sort the list?
There are many methods to sort. We present only one of them.

**ANALYSIS:**
Find the smallest
put it in the front of the list.
Find the second smallest
put it second from the front.
..keep doing this....
OBSERVE: This is a LOOP

**SIDE-ISSUE:** Part of the problem is that a computer can compare only 2 integers at a time.

Example: What is the smallest in the list: 23, 8, 17, 4, 61
Most humans can look at hat list and "spot" the smallest. The computer "can't".

THEREFORE: There will be a loop INSIDE the above loop.

**SUB-PROBLEM:** Find the smallest and put it in front.
**ANALYSIS:** Here is one way to do it - see actual algorithm.

**NOTE:** The Bubble Sort is usually improved by having it check if the array is in order.
If so the loop can be stopped.

```java
import e50utils.*;

class BubbleSrtDemo
{
    private static final int MAX_ARRAY_SIZE = 100;
    private static final int ACTUAL_SIZE = 5;

    int[] list = new int[MAX_ARRAY_SIZE]; // elements init'd to 0
    static boolean showDetails = true;
}*/
```
public class BubbleSortDemo {
    public BubbleSortDemo(int currentSize, boolean showTheDetails) {
        this(currentSize);
        showDetails = showTheDetails;
    }

    public BubbleSortDemo(int currentSize) {
        int i;
        System.out.println("Here is the Initial Zero Array");
        showArray(list, currentSize);

        for (i = 0; i < currentSize; i++) // load the array
            list[i] = Integer.parseInt(e50Input.readWord());
    }

    static void showArray(int[] a, int size) {
        for (int index = 0; index < size; index++)
            System.out.print(e50Format.rAlign(a[index], 5) + " ");
        System.out.println();
    }
}

BubblesSortDemo demo = new BubblesSortDemo(100, true);
BubblesSortDemo demo = new BubblesSortDemo(100);
BubblesSortDemo demo = new BubblesSortDemo(100, false);
BubblesSortDemo demo = new BubblesSortDemo(100);
static void bubblesort (int a[], int size)
{
    int i, j, x;
    for (i = 1; i < size; i++)
    {
        for (j = size-1; j >= i; j--)
        {
            if (a [j - 1] > a[j])
            {
                x = a[j - 1];
                a[j - 1] = a[j];
                a[j] = x;
            }
            if (showDetails)
            {
                System.out.print("\n\tHere is j = " + j + " through inside loop\n\t");
                showArray( a, size);
            }
        } //end of inside loop
    }
    if (showDetails)
    {
        System.out.println("\nHere is i = " + i + " through outside loop");
        showArray( a, size);
        e50Input.pauseLine();
    }
} //end of outside loop

public static void main (String[] args)
{
    BubbleSrtDemo bsd = new BubbleSrtDemo( ACTUAL_SIZE );
    System.out.println("Here is the filled Array");
    showArray( bsd.list, ACTUAL_SIZE);
    e50Input.pauseLine();
    bubblesort (bsd.list, ACTUAL_SIZE);
}
import e50utils.*;

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

      int [] myList = { 20, 10, 0, -5, 100, -10};

      BubbleSrtDemo x = new BubbleSrtDemo(3, false);
      System.out.println( "Here is the array twice." );
      x.showArray( x.list, 3);
      BubbleSrtDemo.showArray( x.list, 3);

      x.bubblesort( x.list, 3 );
      System.out.println( "Here it is sorted." );
      BubbleSrtDemo.showArray( x.list, 3);

      System.out.println( "\nHere is another list." );
      BubbleSrtDemo.showArray( myList, 6 );
      BubbleSrtDemo.bubblesort( myList, 6 );
      System.out.println( "Here it is sorted." );
      BubbleSrtDemo.showArray( myList, 6 );

   }//endOfmain()
}//endOfClass
PROBLEM: Create a class to make Histograms.
(recall GraphMaker.java)

<table>
<thead>
<tr>
<th>Category</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>England</td>
<td>*********</td>
</tr>
<tr>
<td>Spain</td>
<td>********</td>
</tr>
<tr>
<td>France</td>
<td>**********</td>
</tr>
</tbody>
</table>

Run demo: UseHistogram.java

This is a very vague problem.
We have to decide what we want to use this class for.

As usual, we need to analyze the fields we will need and the methods we will need.

Here is a start.

Variable Analysis

I am thinking, that the class needs
- a list of names --- String variables
- a list of integers --- integer variables.

Method Analysis

Some useful methods might be
- Increment the count of a category
- Search for a category
- return the count for a category
- return the number of the category

There are many other useful methods - most of which are not implemented.

REMARK: This class has two "parallel" arrays.
An alternate method, would be to create a subclass which has two fields a string and an integer. And then the Histogram could have just one array of objects. We will demonstrate this later in the notes.
/ Histogram.java - allows one to construct a histogram
// with up to 30 different String categories
//
// Author: Max Kamin, modified by HH Leitner

import utils.*;

class Histogram {
    private String[] categories = new String[30];
    private String nameOfHistogram;
    private final int SIZE;
    private int[] frequencies;
    private int current; // Next category to return

    public Histogram(String name) {
        nameOfHistogram = name;
        System.out.println("Input your categories, and a blank line when done");
        int i = 0;
        String s;
        System.out.print("Input a category: ");
        while (!s = SavitchIn.readLine().equals("")) {
            categories[i] = s;
            i++;
            System.out.print("Input a category: ");
        }
        SIZE = i;
        frequencies = new int[SIZE];
    }

    private int findCat(String cat) {
        // Find location of cat, if any
        int i = 0;
        while (i < SIZE && !categories[i].equals(cat)) i++;
        return i;
    }
}
public void incrCategory (String cat) {
    // cat is one of the categories, hopefully
    // increment the associated frequency.
    int i = findCat (cat);
    if (i != SIZE ) frequencies[i]++;
    else System.out.println("Inappropriate category!");
}

public int categoryFreq (String cat) {
    // cat is one of the categories
    // return the associated frequency.
    return frequencies[findCat(cat)];
}

public int numberOfCategories () {
    return SIZE;
}

public void reset () {
    current = 0;
}

public String nextCategory () {
    current++;
    return categories[current-1];
}

public String getName() {
    return nameOfHistogram;
}

}//end of class
import utils.*;

public class UseHistogram
{
    public static void main (String[] args)
    {
        Histogram myHistogram
        = new Histogram("Hysterical/Historical Survey");
        int freq;
        String category;

        // Read categories up to end-of-file and
        // increment their frequencies
        System.out.print("Input some category
(or nothing to end the program): ");
        category = SavitchIn.readLine();
        while (category != "")
        {
            myHistogram.incrCategory(category);
            System.out.print("Input a category
(or nothing to end the program): ");
            category = SavitchIn.readLine();
        }

        // Output the histogram
        System.out.println(myHistogram.getName());
        System.out.println();
        System.out.println("Category	Count");
        System.out.println();
        myHistogram.reset();
        int lines = myHistogram.numberOfCategories();
        for (int i = 0; i < lines; i++)
        {
            // Print category and then print a number of asterisks
            // corresponding to the frequency
            category = myHistogram.nextCategory();
            freq = myHistogram.categoryFreq(category);
            System.out.print(category + "	" + freq + "\t\t" + System.out.println();
            for (int j=1; j<=freq; j++) System.out.print('*');
            System.out.println();
        }
    }
}
/* file BinarySearch.java

PROBLEM: Search a list to see if a particular integer is in it.

Pre-Analysis:
Start at the beginning and go through each element of the array.

This is fine for small lists.
However, today the databases people want to search are so large that this becomes an efficient method.

Assume that the list is in "order".
Then we can do a BINARY search.
This is far more efficient then going through in order.

ANALYSIS
Jump to the middle of the list.
The item is either
in the front half
or in the back half.

Jump to the middle of the correct half
....repeat the above algorithm

OBSERVE: The way we designate the front half and the back half is by the appropriate indices.

Example
Left-indice   =   0  <-- Array of size = 100
Right-indice  =  99  <-- goes from 0 to 99

Middle-indice = (99+0)/2 = 49  <-- integer division

    Left half is
    Left-indice = 0
    Right-indice = Middle-1 = 48

    Right half is
    Left-indice = Middle + 1 = 50
    Right-indice = 100

Question: What stops the loop?
Answer: examine the code.

*/

import e50utils.*;
class BinarySearch
{
    private static final int MAX_ARRAY_SIZE = 100;

    /** performs a binary search for item in array a[left..right]
     * Returns array index where item found, or -1 if not found
     */
    static int binary (int a[], int item, int left, int right)
    {
        int middle; // middle of array A
        do
        {
            middle = (left + right) / 2;
            if (item > a[middle])
                left = middle + 1;
            else right = middle-1;
        } while ((left <= right) && (a[middle] != item));

        if (a[middle] == item)
            return middle;
        else return -1;
    }

    static void showList( int a[], int size )
    {
        for ( int i = 0; i < size; i++ )
        {
            System.out.print(
                e50Format.rAlign( a[i], 5 )
            );
            if ( i % 10 == 9 )
                System.out.println();
        }
    }
}
public static void main (String[] args) {
    int i, itemLoc;
    char response;

    int[] list = new int[MAX_ARRAY_SIZE]; // elements init'd to 0

    e50FileInput f = new e50FileInput( "RandomIntegers.data" );

    for ( i = 0; i < MAX_ARRAY_SIZE; i++ )
        list[i] = Integer.parseInt((f.readWord()));

    System.out.println( "Here is the unsorted list." );
    showList (list,MAX_ARRAY_SIZE);

    e50Input.pauseLine();

    BubbleSrt.bubblesort (list, MAX_ARRAY_SIZE);
    System.out.println( "\nHere is array sorted into non-descending order:"");
    showList (list,MAX_ARRAY_SIZE);

    e50Input.pauseLine();

    do
    {
        System.out.print ("\nWhich value shall I search for? ");
        i = Integer.parseInt( e50Input.readWord() );

        itemLoc = binary (list, i, 0, MAX_ARRAY_SIZE - 1);

        if (itemLoc >= 0)
            System.out.println ("Found " + i + " at location " + itemLoc);
        else
            System.out.println ("Sorry -- we couldn't find "
                               + i + " in the array!" );

        System.out.print ("Want to search for 
                        another value? [yes or no] ");
        response = e50Input.readChar();
        e50Input.flushLine();
    } while (response != 'N' && response != 'n');
}
/* file == MakeFileIntegers.java

DEMO: Making a file of random integers for use in other programs.

Explanation:
   e50FileOutput f = new e50FileOutput( "RandomIntegers.data" );

   e50FileOutput <-- A Class in the PACKAGE e50utils
     This is loaded by the line
     import e50utils.*;

   e50FileOutput f <-- "f" is a variable of this type. 
     Think of it as a  
     "file" variable.

   = new e50FileOutput( . . ) <-- Creating the actual file "stream"
     "RandomIntegers.data" <-- The actual file being connected to

NOTE: a file has "two" names,
     The operating system name:  RandomIntegers.data
     The program variable name:  f
 */

import e50utils.*;;

class MakeFileIntegers
{
   public static void main (String[] args)
   {
      e50FileOutput f
           = new e50FileOutput( "RandomIntegers.data" );

      for ( int i = 0; i < 100; i++ )
      {
         f.outF.print( (int)(1000 * Math.random()) + " ");
         if ( i % 10 == 9 )
             f.outF.println();
      }
      f.outF.close();
   }
}
In the game of NIM, two players (one of the players is the computer) start with a pile of 11 matches between them. Each player, in turn, must take either 1, 2, or 3 matches from the pile. The player who is forced to take the last match LOSES the game. Pretty simple!

The following PROGRAM "learns" how to play a very simple game of NIM through experience with a human user.

**Strategy:** "Record each winning and losing move"
and get a running total for each possible move.

**More Details:** Each moves consists in knowing the following
- How Many Matches There Were
- How Many Matches Were Taken Away
- And I want to Remember if this move led to a win

**Sub-Problem:** We need a data structure to represent the possible moves
**Solution:** An 11 by 3 TABLE == a 2-dimensional array (run program Lnim2 to see Table)
- Because there are 11 possibilities for how many matches there are, and there are 3 possibilities for how many matches are taken away.

**Sub-Problem:** We need to represent winning versus losing
**Solution:** Add 1 for a move that leads to a win
Subtract 1 for a move that leads to a lose

**OBJECTification.** When I think about the 11 by 3 Table, I also think of some operations I will need to perform on it. Therefore, I want to make the table into an object with the associated methods.

**BUT:** I am not going to extend the Table-Class.
I am going to include it in the Lnim program as another variable-field.
is05:/spring02/unit4 % cat Table.java
// Table class is part of the Lnim program
// Author: Henry H. Leitner
// Last modified: 12/18/01

import utils.*;

class Table
{
    int [][] array;

    Table (int numberOfRows, int numberOfColumns)
    {
        array = new int[numberOfRows] [numberOfColumns];
    }

    void initialize( int aValue)
    {
        for (int row = 0; row < array.length; row++ )
            for (int col = 0; col < array[row].length; col++ )
                array[row][col] = aValue;
    }

    void output()
    {
        for (int i = 0; i < array.length; i++ )
        {
            for (int j = 0; j < array[i].length; j++ )
                System.out.print(array[i][j] + " ");
            System.out.println();
        }
    }

    void setValue(int row, int col, int aValue)
    {
        array[row][col] = aValue;
    }

    void displayHistory()
    {
        for (int row = 1; row < array.length; row++)
            for (int col = 2; col < array[row].length; col++)
            {
                TxtGrph.position (2*row+2, 4*col+4);
                System.out.print
                    (OutFmt.RJust (OutFmt.fmt(array[row][col]), 2));
            }
    }
}
void display()
{
    int numberOfColumns = array[0].length-1;
    TxtGrph.eraseAll();         // first, clear the screen
    for (int i = 0; i < array.length-1; i++)
    {
        TxtGrph.position (2*i+3, numberOfColumns-1);
        System.out.print ("+---+---+---+---+---+---+---+---+---+---+");
        TxtGrph.position (2*i+4, numberOfColumns-1);
        System.out.print ("|   |   |   |   |   |   |   |   |   |   |"Mathf);
    }
    TxtGrph.position (2*array.length+1, numberOfColumns-1);
    System.out.print ("+---+---+---+---+---+---+---+---+---+---+");
    TxtGrph.setReverse();
    for (int i = 2; i <= numberOfColumns; i++)
    {Txt
        TxtGrph.position (2*array.length+2, 4*i+3);
        System.out.print (OutFmt.RJust(OutFmt.fmt(i), 3));
    }
    for (int i = 1; i <= 3; i++)
    {Txt
        TxtGrph.position (2*i+2, 52);
        System.out.print (OutFmt.RJust(OutFmt.fmt(i), 2));
    }
    TxtGrph.setNormal();
    displayHistory();
    TxtGrph.position(20,1);
}

void reinforce (Table winning,Table losing)
    // Here we update the reward/punishment table named HISTORY
{
    for (int row = 1; row < array.length; row++)
        for (int col = 2; col < array[row].length; col++)
            array [row][col] += winning.array [row][col] - losing.array [row][col];
    // we rewarded every move made by the winning player,
    // and punished every move made by the losing player.
}
int chooseOneToThree (int left)
{
    int i, j;

    if (    array[1][left]== array[2][left] 
        && array[2][left]== array[3][left] )
        return (int) Math.random()*3 + 1;
    i = 1;              // start by assuming that T[1]
    // contains the largest entry!
    for (j = 2; j <= 3; j++)
        if (array[i][left] < array[j][left] ||
            (array[i][left] == array[j][left] && Math.random() > 0.5))
            i = j;

    return i;        // convert subscript to # of matches to take
}

int takeSomeMatches (int left)
{
    int choice;

    do
        choice = chooseOneToThree (left);
    while (choice >= left);

    return choice;
}
/** Lnim.java  
* unit 4 lecture notes  
* "learns" how to play the very simple game of NIM  
* through experience with a human user.  
*  
* @author: Dr. Henry H. Leitner  
* @version: Last Modified January, 2002  
*/

import utils.*;  
import e50utils.e50Graph;

public class Lnim  
{  
    static final int SCORE_ROW = 2;  
    static final int A_SCORE_COL = 10;  
    static final int B_SCORE_COL = 40;  
    static final int GAME_INTERACTION_ROW = 13;  
    static final int PLAY_AGAIN_PROMPT_ROW = 16;  
    static final int PLAY_AGAIN_PROMPT_COL = 40;  
    static final int TOTAL_MATCHES = 11;  
    // number of matches to play with

    static Table aMoves  = new Table(4, TOTAL_MATCHES+1);  
    static Table bMoves  = new Table(4, TOTAL_MATCHES+1);  
    static Table history = new Table(4, TOTAL_MATCHES+1);  

    static int aScore=0, bScore=0, matchesLeft=0;
static void play1GameOfNim()
{
    int m;
    boolean game_over;

    TxtGrph.position(GAME_INTERACTION_ROW, 1);
    game_over = false;  // We haven't started playing yet!
    matchesLeft = TOTAL_MATCHES;
    while ( !game_over )
    {
        // Let player A go first
        m = history.takeSomeMatches (matchesLeft);
        aMoves.setValue(m, matchesLeft, 1);  // record that move.
        matchesLeft -= m;  // remove M matches
        System.out.println ("A takes " + m + " matches, leaving " + matchesLeft);

        if (matchesLeft == 1)  // did player A win??
        {
            history.reinforce (aMoves, bMoves);  // yes; remember
            aScore++;  // and tally the win.
            game_over = true;
        }
        else
        {
            do
            {
                System.out.print ("How many matches do you want? ");
                m = SavitchIn.readLineInt();
            } while (m < 1 || m > 3 || m >= matchesLeft);
            bMoves.setValue(m, matchesLeft, 1);  // record B's move.
            matchesLeft -= m;  // remove M matches
            System.out.println ("B takes " + m + " matches, leaving " + matchesLeft);

            if (matchesLeft == 1)  // did player B win??
            {
                history.reinforce (bMoves, aMoves);  // yes; remember
                // the experience
                bScore++;  // and tally the win.
                game_over = true;
            }
        }
    }
}
public static void main (String[] args)
{
    e50Graph.scrollRegion(GAME_INTERACTION_ROW, 24);

    char response;
    history.display(); // Initialize everything
    do
    {
        aMoves.initialize(0);
        bMoves.initialize(0);

        TxtGrph.position(GAME_INTERACTION_ROW, 1);
        TxtGrph.eraseDown();

        play1GameOfNim();
        TxtGrph.position(SCORE_ROW, A_SCORE_COL);
        System.out.print("A score: "+aScore);
        TxtGrph.position(SCORE_ROW, B_SCORE_COL);
        System.out.print("B score: "+bScore);
        TxtGrph.beep();
        history.displayHistory();
        TxtGrph.position(PLAY_AGAIN_PROMPT_ROW, PLAY_AGAIN_PROMPT_COL);
        System.out.print("Do you want to play again? ");
        response = SavitchIn.readLineNonwhiteChar();
    } while (response !="N" & response !="n");

    e50Graph.scrollRegion(1, 24);
}
/* file == SwapComposers.java  --- see accompanying file StringNumber.java

DEMONSTRATES sorting an array of objects
  + a few advanced notions.

Uses the class:  StringNumber.java
  which is a simple object containing
  a string and an integer variable.

The file StringNumber.java is in the same directory as this file
  so we don't need any "import".
  (Technicality: actually its just a link to the file
  but that is irrelevent for the discussion.)

This reads the file: composers.text
  which has
    110 lines in file
    each line has
      date of birth
      name (may be spaces and more than 1 word)

Digression: Sorting an Array of objects is built into Java.
  see SwapComposers2.java and StringNumber2.java

*/

import e50utils.*;

class SwapComposers
{
  public static void bubblesort (StringNumber a[], int size)
  {
    int i, j;
    StringNumber x;
    
    for (i = 1; i < size; i++)
      for (j = size-1; j >= i; j--)
        {
          if ( a[j].comesBefore( a[j-1] ) )
            {
              x = a[j - 1];
              a[j - 1] = a[j];
              a[j] = x;
            }
        }
  }
}
public static void swap( StringNumber[] a, int i, int j )
        {
            StringNumber temp;
            temp = a[i]; a[i] = a[j]; a[j] = temp;
        } //not used in this example

public static void main (String args[])
        {
            System.out.println(
                "First a demo of accessing the class StringNumber" );
            System.out.println( "... and an array thereof." );

            StringNumber [] l = { new StringNumber("fred",1),
                                      new StringNumber("Wilma",2),
                                      new StringNumber("Barney",3 )};

            for ( int k = 0; k < l.length; k++ )
                    System.out.println( l[k].display() );

            System.out.println( "Testing the boolean predicate comesBefore.");
            System.out.println( l[1].comesBefore( l[0] ) );
            System.out.println( l[1].comesBefore( l[2] ) );

            System.out.println( "\n\ntHere are the composers in the file");
            e50Input.pauseLine();

            StringNumber [] composer = new StringNumber[110];

            e50FileInput foo = new e50FileInput( "composers.text" );

            int j = 0; String temp1, temp2;
            temp1 = foo.readWord();
            temp2 = foo.readLine().trim();
            while( ! foo.EOF )
                    {
                composer[j] =
                        new StringNumber(temp2,e50Convert.toInt(temp1));
                        // composer[j] = new StringNumber();
                        // composer[j].i = e50Convert.toInt( temp1 );
                        // composer[j].s = temp2;
                        temp1 = foo.readWord();
                        temp2 = foo.readLine().trim();
                        System.out.println( composer[j++].display() );
                    }
System.out.println(  
    "\n\tWe will sort the first 7 of them"");

e50Input.pauseLine();

for ( int k = 0; k < 7; k++ )
    System.out.println( k + ":"  
                        + composer[k].display() );
System.out.println( "109:" + composer[109].display() );

e50Input.pauseLine();

bubblesort( composer, 7 );
for ( int k = 0; k < 7; k++ )
    System.out.println( k + ":" + composer[k].display() );

} // endOfmain()
} // endOfClass

/* REMARKS
Recall: methods don't change the value of the items sent to them.
Why can the method bubblesort change some of the variables.

swap( StringNumber[] a, int i, int j )  <-- a, i, j
    cannot be changed.
    a                  <-- a is a variable.
    It contains the ADDRESS
    of the array of objects.
    We can go to that address and change values there.

    a[i], a[j]  <-- are two of the values we are changing.
    They each happen to also be addresses.
*/
/* file == StringNumber.java

This class is used in the class SwapComposers.java

This is just a class for the purpose of Demonstrations -
for something other than the primitive types.

It doesn't have a main() method (though it could).

It has two instance variables
  a String
  a Number

If has 4 Constructors.
  The various possible initializations

It has a boolean method to compare two items.
  this.comesBefore(that)
  is either true or false.
  It compares Integer only.....

It has 2 instance methods
  show
  display

Digression: all objects can be printed out with .toString()
but you won't like what you get.
*/
class StringNumber
{
    String s;
    int i;

    //......4 Constructors
    StringNumber()
    {
        i = -99;
        s = "Adam";
    }

    StringNumber( int comingIn )
    {
        i = comingIn;
        s = "Adam";
    }

    StringNumber ( String comingIn )
    {
        i = -99;
        s = comingIn;
    }

    StringNumber ( String sIn, int iIn )
    {
        i = iIn;
        s = sIn;
    }

    String show ()
    { return s + i; }

    String display()
    { return \"\t\t\" + i + \"\t\" + s; }

    boolean comesBefore ( StringNumber sn )
    { return ( this.i < sn.i ); }
}
}//end of class
/* file == AfterBArrays.java

ILLUSTRATION of some manipulations with Arrays.
An array of booleans
An array of characters
Converting a String to an array of characters.
Passing arrays to methods.
Methods which return arrays.

Recall the AfterB.java problem.
A secret password is hidden inside of a longer string of integers.

We will redo this with Arrays.
This is not necessarily better - it is only meant to demonstrate some uses of Arrays.

Note however, that the previous solution did have a mistake in it.
It did not consider what happens if the message itself has a 'b' in it. So we could use arrays to do more analysis of the message.

DEMO 1: First we will make an array of booleans to tell us where the possible letters are

So we want a method which receives a string and returns an array of booleans.

DEMO 2: We will convert the string to an array of capital characters.
Strings are "immutable". That is, you cannot change and of its letters.
By converting it to an array of characters we can modify the letters.

return s.toLowerCase().toCharArray();

s                             <-- This is String
s.toLowerCase()               <-- A String method which returns a different String
.toCharArray                  <-- A String method which returns an Array of characters.

OBSERVE: .toCharArray is operating on (s.toLowerCase())
not on the String s.

DEMO 3:
We will write a method which receives two arrays of the same size. The first an array of characters and the second an array of booleans. Stepping through, we will use the boolean array to decide where to CHANGE the character Array.

Remark: An Array of booleans is automatically initialized to "false". So we need only change the appropriate elements to true.

```java
import e50utils.*;
class AfterBArrays {
    static final String TEST_MESSAGE =
        "VbfhNblSDbutaBbiXBbeBeJBrAB ab=EHB "
        + "szBfublhbywbiLbnlbgpAb "
        + "DbrtBuzCbbekbbcBerObr";

    static final String PW1 = "VbfhNblSDbutaBbiXBbeBeJBr";
    static final String PW2 = "kbBxblgUBuudbbOBbttyBeRNBr";
    static final String PW3 = "TBBfblcbaObbGBboBerBr";

    static String deCode( String s ) {
        String temp = "";
        for ( int i = 0; i < s.length() - 1; i++ )
            if ( s.charAt(i) == 'b' || s.charAt(i) == 'B' )
                temp = temp + s.charAt(i+1);
        return temp;
    }
}
```
static void deCode ( char [] a, boolean [] b )
{
    System.out.println( "We will attempt to decode the" + " message. " );
    if ( a.length != b.length )
        System.out.println("They are not the same length.");

    for ( int i = 0; i < a.length - 1; i++ )
    {
        if ( ! b[i] ) a[i+1] = ' ';
        else if ( b[i+1] ) a[i+2] = ' ';
    }
    a[0] = ' '; //first letter
}

static boolean[] makeBooleanArray( String s )
{
    String temp = s.toUpperCase();
    int l = s.length();
    boolean [] b = new boolean [l];

    for ( int i = 0; i < l; i++ )
        if ( temp.charAt(i) == 'B' ) b[i] = true;
    return b;
}

static void displayBArray( boolean [] b )
{
    System.out.println("\t\t\tHere is the\n\t\t\tbooleans");
    for ( int i = 0; i < b.length; i++ )
    {
        System.out.print( b[i] );
        if ( i % 8 == 7 ) System.out.println();
    }
    System.out.println();
e50Input.pauseLine();
    System.out.println("\n\n");
}
static char[] s2ch ( String s )
{
    return s.toLowerCase().toCharArray();
}

public static void main (String args[])
{
    String s = PW3;
    System.out.println( "Here is this message decoded:\t\t" + deCode( PW3 ) );
    System.out.println( "Which you may notice is " + "not actually a word.\n\n" );
e50Input.pauseLine();

    boolean [] trueB = makeBooleanArray( s );
    displayBArray( trueB );

    char [] a = s2ch( s );
    System.out.println( a );

    deCode( a, trueB );
    System.out.println( a );
}
Advanced but useful

**PROBLEM:** There is a file with words and numbers in it. Write a program to sum up the numbers in the file.

A `StreamTokenizer` parses an inputstream into "tokens" so that we can "read" the tokens one at a time. The tokens are separated by whitespace." In Parsing, it knows to skip over end-of-lines.

Notice the last token is processed after the loop. This is the EOF token.

```
import java.io.*;

class StrmTokFile
{
    public static void main (String args[])
        throws FileNotFoundException, IOException
    {
        double sum = 0.0;

        int tokenType;

        StreamTokenizer inF = new StreamTokenizer( 
            new FileReader( "forStrmTok.data" ) );

        tokenType = inF.nextToken();

        while ( tokenType != StreamTokenizer.TT_EOF )
        {
            System.out.print( tokenType + "\t");
            switch( tokenType )
            {
                case inF.TT_EOF : System.out.println( "EOF" );
                    break;

                case inF.TT_NUMBER :
                    sum += inF.nval;
                    System.out.println( inF.nval + "\t" + sum );
                    break;

                case inF.TT_WORD :
                    System.out.println( inF.sval );
                    break;
            }

            tokenType = inF.nextToken();
        }

        System.out.println( tokenType );
    } //endOfmain()
} //endOfClass
```
Consider the line
StreamTokenizer inF = new StreamTokenizer(
    new FileReader( "forStrmTok.data" ) );

We are opening the file with FileReader.
StreamTokenizer is **wrapping around** the file and adding extra functionality.

Basically, we are viewing the stream in the
file forStrmTok.data as a sequence of tokens separated by white space.
Whitespace means, the space character, the tab character,
the newling character, and other items of that ilk.
Hence the tokens correspond to what would we think of as words.

Example:  Aaa 1 is 2
has 4 tokens.

In addition, StreamTokenizer allows us to know
which are actual words and which are numbers.

Here is part of the class definition

```java
public class StreamTokenizer extends Object {
    //some constants
    public static final int TT_EOF ...
    public static final int TT_EOL ...
    public static final int TT_NUMBER ...
    public static final int TT_WORD ...
    //some variables
    public int ttype;
    public String sval;
    public double nval;
    //constructor
    public StreamTokenizer ( Reader r )
    {
    }
    //method
    public int nextToken()
    {
    }
}
```

StreamTokenizer processes each token by
storing an integer in variable ttype
which indicates its type
and
storing a number in variable nval
and
storing a word in variable sval
A function which calls itself is recursive. As a definition, it is simple to say whether or not something is recursive. The question is, what problems are "nicely" solved recursively

**RECURSION** and **BACKTRACKING** are natural learn to recognize them.

**PROBLEM:** You are in a room with 2 doors. When you choose one and go through, you enter another room with two doors. Again, you choose one and go through - again you are in a room with two doors. You keep on until you get to a room with no doors or until you get outside. If you get outside, you are done. If you get to a room with no doors you BACKTRACK to a previous room and choose a new door.
main(...)  
{  
    pick_a_door();  
    try_and_get_out();  
}  

try_and_get_out()  
{  
    if ( room_leads_out() )  
        leave();  
    else if ( room_has_doors() )  
    {  
        pick_a_door();  
        try_and_get_out();  
        {** recur **}  
    }  
    else  
    {  
        do { {** back track **}  
            go_back_a_room();  
        } while ( all-doors_already_tried() );  
        pick_a_door();  
        try_and_get_out();  
    }  
}
/* file == Factorial.java  see demo: java FactorialStep 8

PROBLEM: Write a method to calculate the factorial of any positive integer.

Factorial of 5 = 5 * 4 * 3 * 2 * 1
Factorial of \( n \) = \( n \) * \((n-1)\) * \((n-2)\) * ... * 2 * 1

DEMO: We will give two solutions to this - iterative and recursive.

PROGRAMMING NOTE: The problem did not state what to do if the input is 0 or a negative number. The two versions below handle this by returning 1 for such input. This is not necessarily correct.
   What it means is that the problem was not completely stated.
   Technically, the program should be idiot-proofed to inform the user not to try improper input.

MATH NOTE: The factorial of 0 is 1. But the factorial of negative integers is not defined.
For those interested, look up The Gamma Function in a math book.

ANALYSIS 1
   The factorial can be iterated in a loop starting with \( n \) and decreasing by 1 each time.
   (see the method for details.)

ANALYSIS 2.
   Observe that
   \[ \text{Fact}(5) = 5 * (4*3*2*1) \] <-- definition of factorial
   = 5 * \text{Fact}(4) \] <-- Recognizing the recursive nature.

DIGRESSION: Using the ternary operator
*/
class Factorial
{
    static int factRecur (int n)
    {
        return ( n <= 0 ? 1 : n * factRecur(n-1) );
    }

    static int factIterate ( int n )
    {
        int temp = 1;
        while ( n > 1 )
        {
            temp = temp * n--;
        }
        return temp;
    }

    public static void main(String args[])
    {
        for (int i = -3 ; i < 10; i++ )
            System.out.println( factRecur(i) + "\t\t" + factIterate(i) );
    }

    /* OUTPUT
    is03:~/unit3 % java Factorial
    1 1
    1 1
    1 1
    1 1
    2 2
    6 6
    24 24
    120 120
    720 720
    5040 5040
    40320 40320
    362880 362880
    */
/* file == NeedBaseCase.java

WARNING: In Recursion, there must be a BASECASE to stop the program from calling itself forever --- i.e. INFINITE LOOP.
*/

import e50utils.*;

class NeedBaseCase
{
  void with_base_case ( int n )
  {
    if ( n > 5 )
      System.out.println( "ending i hope");
    else
      {
        System.out.println("calling with: " + (n+1) );
        with_base_case( n + 1 );
      }
    System.out.println( "finishing call: " + n ) ;
  }
  void no_base_case ( int n )
  {
    if ( n >10) System.exit(0); //added as emergency exit
    System.out.println( "called with: " + n );
    no_base_case( n+1 );
    System.out.println( "??THIS LINE NEVER IS EXECUTED ??");
  }

  public static void main (String args[])
  {
    NeedBaseCase r = new NeedBaseCase();
    r.with_base_case(1);
    e50Input.pauseLine();
    r.no_base_case(1);
  }
}
/*PROBLEM: input a positive integer and write it out in reverse order.

ANALYSIS: First write out the last digit
           (which happens to be  number % 10)
           THEN reverse the rest of the number
           (which happens to be  number / 10)
 */

// file == ReverseInteger.java

import e50utils.*;

class ReverseInteger
{
    void flip ( int number )
    {
        System.out.print ( number % 10 );
        if ( number/10 != 0 )
            flip( number/10 );
    }

    public static void main (String args[])
    {
        ReverseInteger r  = new ReverseInteger();
        System.out.print ( "Please input a positive integer: " );
        int n = e50Convert.toInt( e50Input.readLine() );
        r.flip( n );
        System.out.println();
    }
}
// file == ReverseLine.java

/*PROBLEM: reverse a line of characters typed at the keyboard.

ANALYSIS  read a char
           if not end of line
           then
               reverse the rest of the line
               write out char (the one we read at the top)

COMPARE this to reversing an integer:
       how are they different?
       how are they the same?
*/

import e50utils.*;
class ReverseLine {
    void flip () {
        char ch;
        if ( (ch = e50Input.readChar()) != '\n' )
            {
                flip();
                System.out.print( ch );
            }
    }

    public static void main (String args[])
    {
        System.out.println( "Type a line of text." );
        ReverseLine r = new ReverseLine();
        r.flip();
        System.out.println();
    }
}
/* file = RecursiveE.java

QUESTION: How do you think functions like, sin(x) and exp(x) are actually done on the computer?

Here is a recursive method to calculate exp(x).

MATH-DIGRESSION: Where does the algorithm come from? Recall from your old calculus course the following definition of exp(x):

\[ e^x = \lim_{n \to \infty} \left( 1 + \frac{x}{n} \right)^n \]

ILLUSTRATION: This also shows the difference in calculating with doubles versus floats.

DIGRESSION: The answer we get is not to within the prescribed accuracy. We will not go into the details of how to "improve" this. This is an active field of research.

*/

import e50utils.*;

class RecursiveE
{
    double PRECISION = 0.01;

    RecursiveE () {};
    RecursiveE ( double p ) { PRECISION = p; }

    double doubleE ( double x)
    {
        double r;

        if (Math.abs (x) < PRECISION) return 1 + x;
        else r = doubleE (x/2) ;
        return r * r;
    }
}
// a float version - just to compare.
float floatE ( float x)
{
    float r;

    if (Math.abs (x) < PRECISION) return 1 + x;
    else r = floatE (x/2) ;
    return r * r;
}

class RecursiveE {
    float floatE ( float x)
    {
        float r;

        if (Math.abs (x) < PRECISION) return 1 + x;
        else r = floatE (x/2) ;
        return r * r;
    }

    public static void main (String args[]) {
        RecursiveE r = new RecursiveE();

        System.out.println( "Here is the value of E "
            + "stored in the Math class: "
            + Math.E );
        System.out.println( "Here is the value of E "
            + "calculated as a double: "
            + r.doubleE(1) );
        System.out.println( "Here is the value of E "
            + "calculated as a float: "
            + r.floatE(1) );
    }
}
PROBLEM: KNIGHT TOUR:

Given a chessboard with a knight placed on it in some arbitrary location, find a knight's tour of the board. That is find a set of moves so that the Knight visits each square once and only once (and of course visits every square).

ANALYSIS:

try and move the knight
if a move is possible then
  do the move
  check if he board is filled
  if not then try and move the knight (recursion)
  if yes then done.
else
  back track a move if possible
  if not possible then quit.

/* Here is the pseudo-code for the recursive function try-and-move */

{ repeat select next possible move (of the eight)
  if move is acceptable (i.e. not off board or already taken)
  { do the move
    if not full board full
    { try (*recursion*)
      if not success backtrack
    }
  }
  until success or no more possible moves

This is the standard recursive solution.
Can we improve on this?

**HUMAN INSIGHT**

With a small amount of human insight, we can vastly improve the program to find a knight's tour of the chessboard.

----- will be discussed in class -----

Here are two runs of the above program and an improved version.

```
$ run knight
Enter starting X & Y coordinates: 1 1
Enter boardsize - maximum is 30, minimum is 3
5
Knight's tour for a 5 by 5 board
Starting knight position is (1,1)

Completed knight's tour:

```
```
    1  6  15  10  21
  14  9  20  5  16
  19  2  7  22  11
  8  13  24  17  4
 25  18  3  12  23
```
```

Number of recursive calls is: 8840
CPU time in milliseconds is: 15190
```

```
$ run hknight
Enter starting X & Y coordinates: 1 1
Enter boardsize - maximum is 30, minimum is 3
5
Knight's tour for a 5 by 5 board
Starting knight position is (1,1)

Completed knight's tour using heuristic search:

```
    1  22  11  16  7
  12  17  8  21  10
  25  2  23  6  15
  18  13  4  9  20
   3  24  19  14  5
```
```

Number of recursive calls is: 25
CPU time in milliseconds is: 100