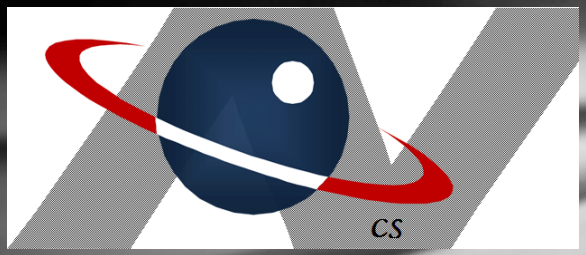
O(N) CS LESSONS

Lesson 9 – Matrices



By John B. Owen

Table of Contents

Objectives

- In several previous lessons, you learned about arrays and how they can handle multiple instances of the same data type.
- The arrays you learned about were one-dimensional, and could represent a single row of elements.
- In this lesson you will learn about ***matrices*** (plural for ***matrix***) which can represent several rows of data in a 2-dimensional fashion.

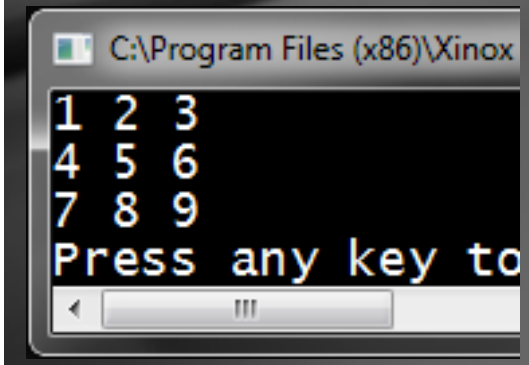
What is a matrix?

- A matrix is an array of arrays, a one-dimensional array of one-dimensional arrays.
- A matrix has rows and columns, like a spreadsheet.
- Often a matrix is referred to as a ***2D array*** (two-dimensional array)
- It is even possible to have 3D, 4D, or 5D arrays…whatever you need!

Matrix example

- Here you see a very simple example of a 3X3 matrix – 3 rows, and 3 columns in each row.

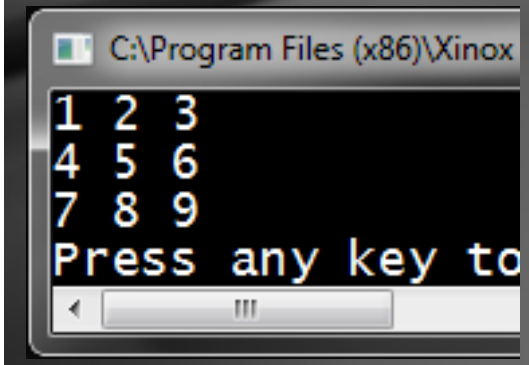
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    out.print(grid[0][0]+" ");
    out.print(grid[0][1]+" ");
    out.println(grid[0][2]);
    out.print(grid[1][0]+" ");
    out.print(grid[1][1]+" ");
    out.println(grid[1][2]);
    out.print(grid[2][0]+" ");
    out.print(grid[2][1]+" ");
    out.println(grid[2][2]);
}
}
```



Matrix example

- First notice how the array is created.
  - The data type is followed by TWO pairs of square brackets, then the identifier, then the elements.

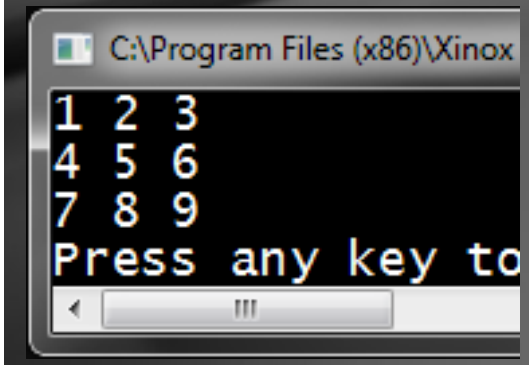
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    out.print(grid[0][0]+" ");
    out.print(grid[0][1]+" ");
    out.println(grid[0][2]);
    out.print(grid[1][0]+" ");
    out.print(grid[1][1]+" ");
    out.println(grid[1][2]);
    out.print(grid[2][0]+" ");
    out.print(grid[2][1]+" ");
    out.println(grid[2][2]);
}
}
```



Matrix example

- The elements are arranged inside nested brackets.
  - The outside pair of brackets encompasses the entire grid.

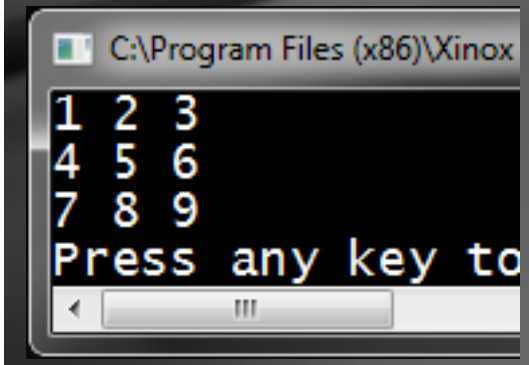
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    out.print(grid[0][0]+" ");
    out.print(grid[0][1]+" ");
    out.println(grid[0][2]);
    out.print(grid[1][0]+" ");
    out.print(grid[1][1]+" ");
    out.println(grid[1][2]);
    out.print(grid[2][0]+" ");
    out.print(grid[2][1]+" ");
    out.println(grid[2][2]);
}
}
```



Matrix example

- The inside pairs of brackets contain each row of the grid, with commas separating each row.

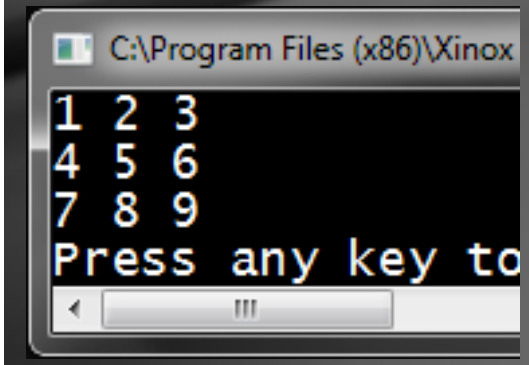
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };

    out.print(grid[0][0]+" ");
    out.print(grid[0][1]+" ");
    out.println(grid[0][2]);
    out.print(grid[1][0]+" ");
    out.print(grid[1][1]+" ");
    out.println(grid[1][2]);
    out.print(grid[2][0]+" ");
    out.print(grid[2][1]+" ");
    out.println(grid[2][2]);
}
}
```



Matrix example

- The values inside the inner sets of brackets are the column elements of each row, also separated by commas.

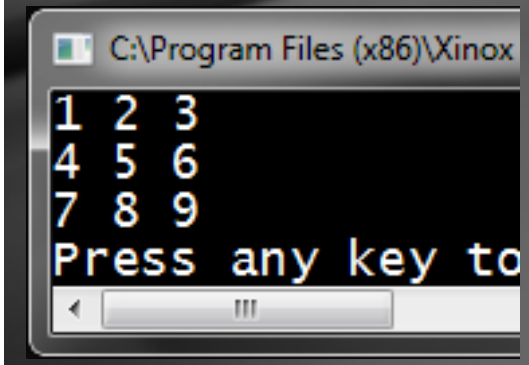
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    out.print(grid[0][0]+" ");
    out.print(grid[0][1]+" ");
    out.println(grid[0][2]);
    out.print(grid[1][0]+" ");
    out.print(grid[1][1]+" ");
    out.println(grid[1][2]);
    out.print(grid[2][0]+" ");
    out.print(grid[2][1]+" ");
    out.println(grid[2][2]);
}
}
```



Matrix example

- Finally, the contents of the grid are output using TWO index values…
  - The first index represents the ***row***
  - The second index represents the ***column***.

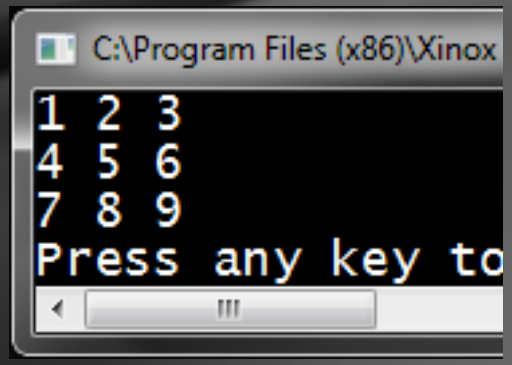
```
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    out.print(grid[0][0]+" ");
    out.print(grid[0][1]+" ");
    out.println(grid[0][2]);
    out.print(grid[1][0]+" ");
    out.print(grid[1][1]+" ");
    out.println(grid[1][2]);
    out.print(grid[2][0]+" ");
    out.print(grid[2][1]+" ");
    out.println(grid[2][2]);
}
}
```



Output using nested loops

- Just as arrays use loops to process, matrices often use nested loops, as seen below.
  - The outside loop is the ***row*** loop, and the inside one the ***column*** loop

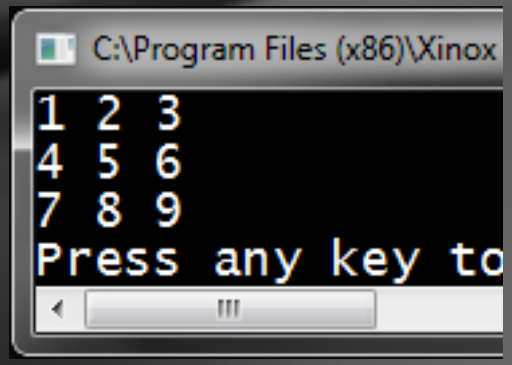
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    for(int r=0;r<grid.length;r++)
    {
        for(int c=0;c<grid[r].length;c++)
            out.print(grid[r][c]+" ");
        out.println();
    }
}
```



grid.length vs grid[r].length

- The ***length*** command for any array simply provides the number of elements in that array.
- The ***grid.length*** command below measures the "length" of the **grid** array, which in reality counts how many **rows** are in the matrix, in this case 3.

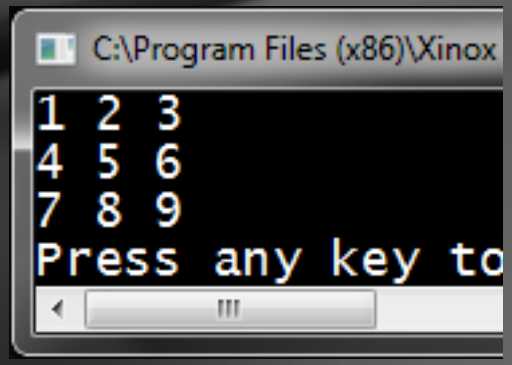
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    for(int r=0;r<grid.length;r++)
    {
        for(int c=0;c<grid[r].length;c++)
            out.print(grid[r][c]+" ");
        out.println();
    }
}
```



grid.length vs grid[r].length

- The ***grid[r].length*** command measures the "length" of the each row in the grid array, or the number of column elements in each row, again 3 in each case here.

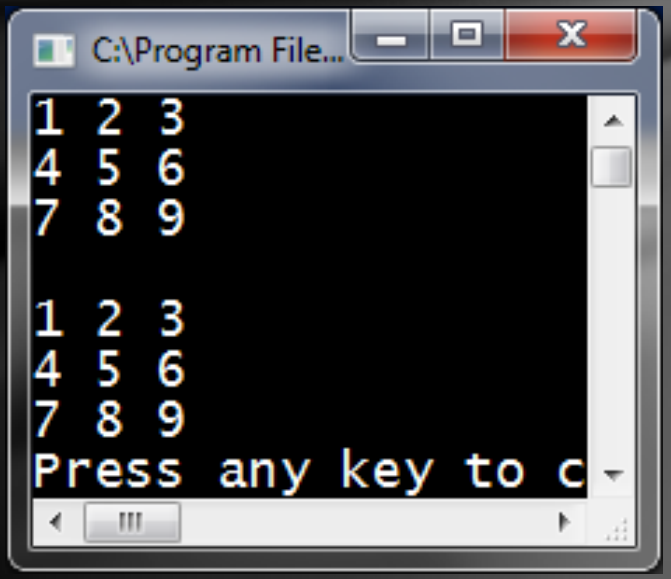
```
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    for(int r=0;r<grid.length;r++)
    {
        for(int c=0;c<grid[r].length;c++)
            out.print(grid[r][c]+" ");
        out.println();
    }
}
```



Output with for each nested loops

- You can use the ***for each*** loops in a nested fashion to output this matrix.
- Check it out.

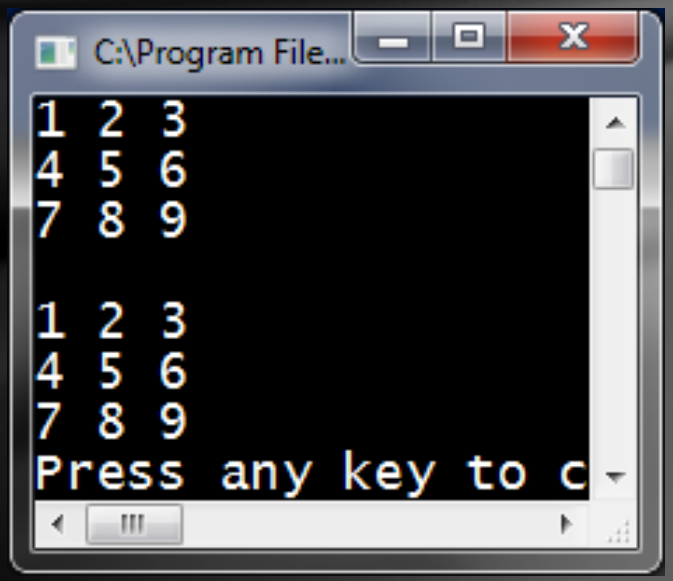
```
es.java ×
 */
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    for(int r=0;r<grid.length;r++)
    {
        for(int c=0;c<grid[r].length;c++)
            out.print(grid[r][c]+" ");
        out.println();
    }
    out.println();
    for(int [] row:grid)
    {
        for(int col:row)
            out.print(col+" ");
        out.println();
    }
}
```



Output with for each nested loops

- Notice carefully that the data type for the first loop is an array of the data type, and for the second loop it is the data type itself.

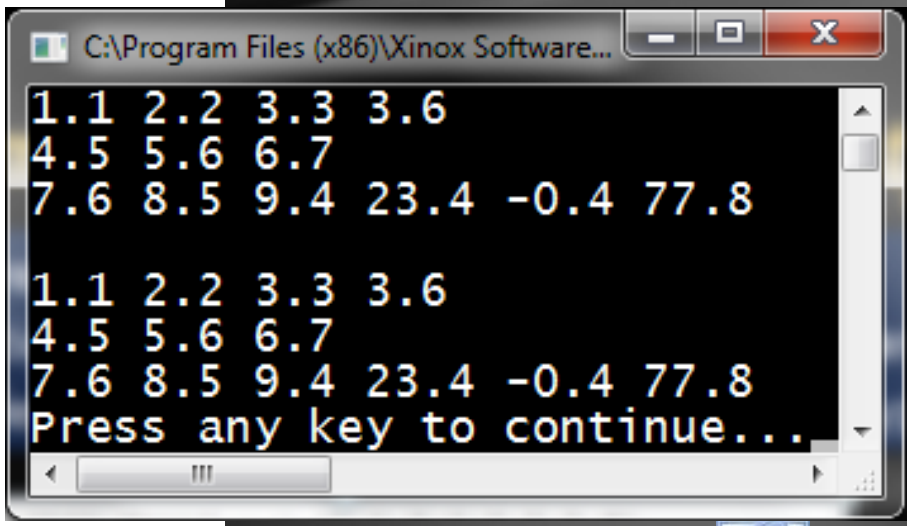
```
es.java ×
 */
public static void main (String [] args)
    throws IOException
{
    int [][] grid = { {1,2,3},
                      {4,5,6},
                      {7,8,9} };
    for(int r=0;r<grid.length;r++)
    {
        for(int c=0;c<grid[r].length;c++)
            out.print(grid[r][c]+" ");
        out.println();
    }
    out.println();
    for(int [] row:grid)
    {
        for(int col:row)
            out.print(col+" ");
        out.println();
    }
}
```



double matrix

- Here is an example of a matrix with type ***double***.
- Notice carefully that the number of columns in each row are different, which is possible for any data type, although most of the time the rows are all the same length.

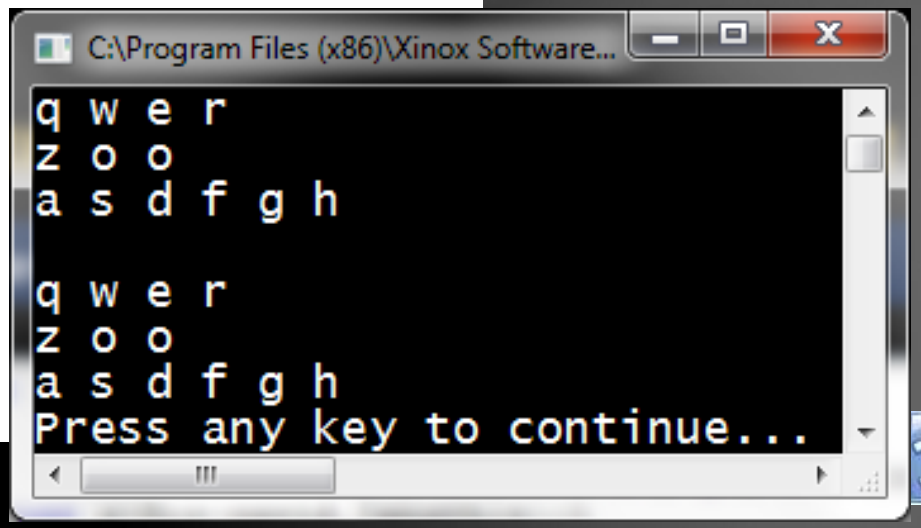
```
double [][] dgrid = { {1.1,2.2,3.3,3.6},
                      {4.5,5.6,6.7},
                      {7.6,8.5,9.4,23.4,-0.4,77.8} };
for(int r=0;r<dgrid.length;r++)
{
    for(int c=0;c<dgrid[r].length;c++)
        out.print(dgrid[r][c]+" ");
    out.println();
}
out.println();
for(double [] row:dgrid)
{
    for(double col:row)
        out.print(col+" ");
    out.println();
}
```



char matrix

- Here is a char matrix.
- Notice the only changes involve the data type. The matrix construction and output processes are identical.

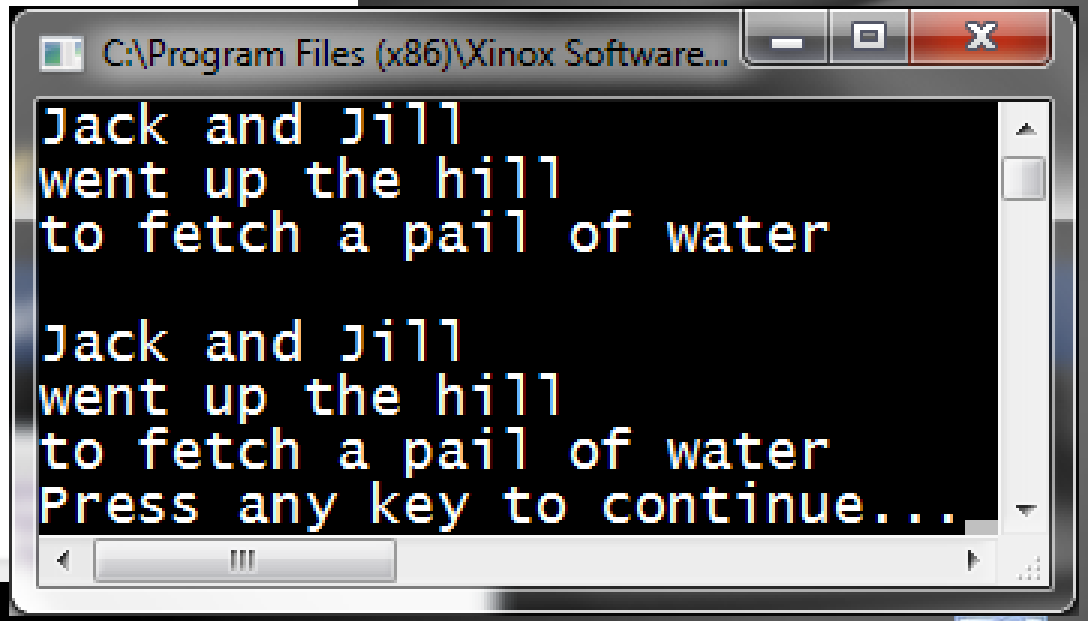
```
char [][] cgrid = { {'q','w','e','r'},
                    {'z','o','o'},
                    {'a','s','d','f','g','h'} };
for(int r=0;r<cgrid.length;r++)
{
    for(int c=0;c<cgrid[r].length;c++)
        out.print(cgrid[r][c]+" ");
    out.println();
}
out.println();
for(char [] row:cgrid)
{
    for(char col:row)
        out.print(col+" ");
    out.println();
}
```



String matrix

- Here is a String matrix.

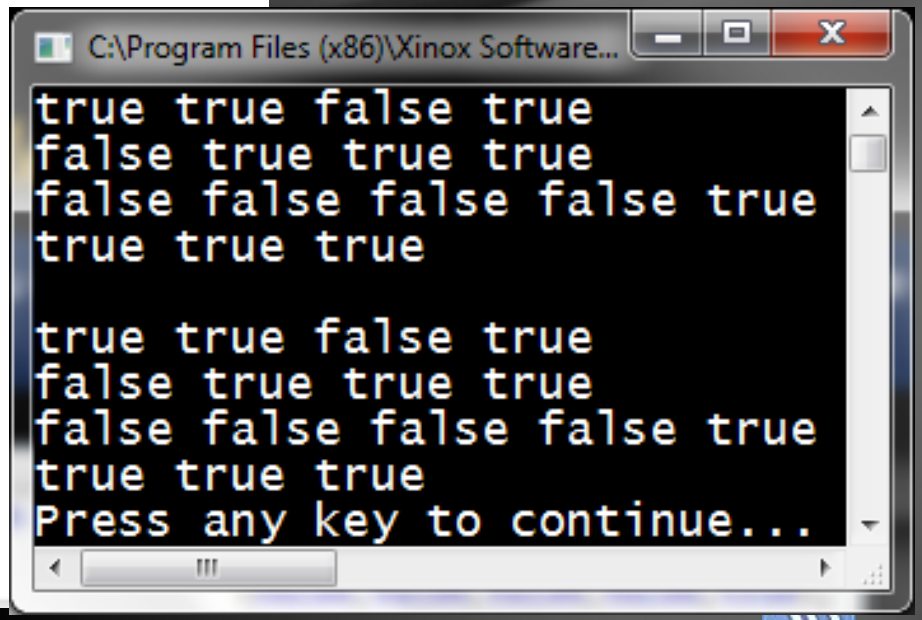
```
String [][] sgrid = { {"Jack","and","Jill"},
                      {"went","up","the","hill"},
                      {"to","fetch","a","pail","of","water"} };
for(int r=0;r<sgrid.length;r++)
{
    for(int c=0;c<sgrid[r].length;c++)
        out.print(sgrid[r][c]+" ");
    out.println();
}
out.println();
for(String [] row:sgrid)
{
    for(String col:row)
        out.print(col+" ");
    out.println();
}
```



boolean matrix

- You can even have a boolean matrix if the situation requires it.

```
boolean [][] bgrid = { {true,true,false,true},
                       {false,true,true,true},
                       {false,false,false,false,true},
                       {true,true,true} };
for(int r=0;r<bgrid.length;r++)
{
    for(int c=0;c<bgrid[r].length;c++)
        out.print(bgrid[r][c]+" ");
    out.println();
}
out.println();
for(boolean [] row:bgrid)
{
    for(boolean col:row)
        out.print(col+" ");
    out.println();
}
```



Creating "new" matrices

- The "new" process you learned with arrays also works with matrices, as you can see below.
- Default values are automatically loaded into each new array

s.java

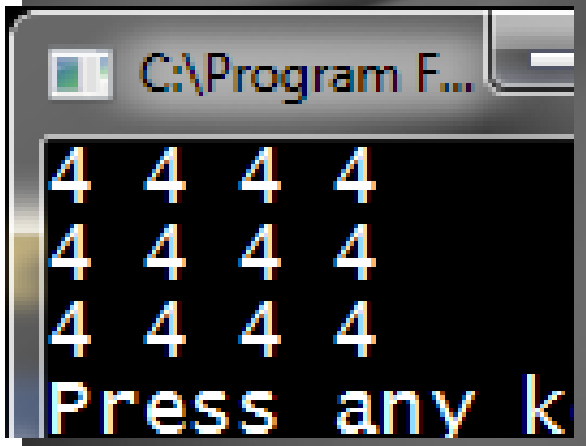
```
public static void main (String [] args)
    throws IOException
{
    int [][] igrid = new int[3][4];
    double [][] dgrid = new double[4][4];
    char [][] cgrid = new char[5][3];
    String [][] sgrid = new String[5][4];
    boolean [][] bgrid = new boolean[2][2];
    for(int [] row:igrid){
        for(int col:row) out.print(col+" ");
        out.println();}
    for(double [] row:dgrid){
        for(double col:row) out.print(col+" ");
        out.println();}
    for(char [] row:cgrid){
        for(char col:row) out.print(col+"-");
        out.println();}
    for(String [] row:sgrid){
        for(String col:row) out.print(col+" ");
        out.println();}
    for(boolean [] row:bgrid){
        for(boolean col:row) out.print(col+" ");
        out.println();}
```

```
0 0 0 0
0 0 0 0
0 0 0 0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
 - - -
 - - -
 - - -
 - - -
 - - -
null null null null
null null null null
null null null null
null null null null
null null null null
false false
false false
Press any key to continue...
```

Filling matrices

- Although there is no direct way to fill matrices as there was with arrays, a loop can be used to fill each row, which IS an array, using the ***Arrays.fill*** method.

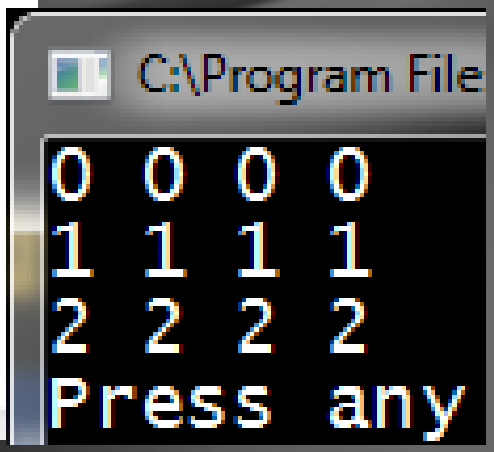
```
public static void main (String [] args)
    throws IOException
{
    int [][] igrid = new int[3][4];
    for(int r=0;r<igrid.length;r++)
        Arrays.fill(igrid[r],4);
    for(int [] row:igrid){
        for(int col:row) out.print(col+" ");
        out.println();}
```



Filling matrices

- You could also fill each row with a different value, like the row number itself.

```
public static void main (String [] args)
    throws IOException
{
    int [][] igrid = new int[3][4];
    for(int r=0;r<igrid.length;r++)
        Arrays.fill(igrid[r],r);
    for(int [] row:igrid){
        for(int col:row) out.print(col+" ");
        out.println();}
```



Loading new arrays, example #1

- To load an array with outside data, a keyboard input process can be used, as shown below.

```
How many rows do you need?==>4
How many columns do you need?==>2
Enter element for row 0, column 0==>1
Enter element for row 0, column 1==>2
Enter element for row 1, column 0==>4
Enter element for row 1, column 1==>6
Enter element for row 2, column 0==>3
Enter element for row 2, column 1==>7
Enter element for row 3, column 0==>8
Enter element for row 3, column 1==>5
1 2
4 6
3 7
8 5
Press any key to continue...
```

es.java × Arrays.html

```
public static void main (String [] args)
    throws IOException
{
    Scanner kb = new Scanner(in);
    out.print("How many rows do you need?==>");
    int rows = kb.nextInt();
    out.print("How many columns do you need?==>");
    int cols = kb.nextInt();
    int[][]grid = new int[rows][cols];
    for(int r=0;r<rows;r++)
        for(int c=0;c<cols;c++)
        {
            out.printf("Enter element for row %d, column %d==>",
            r,c);
            grid[r][c]=kb.nextInt();
        }
    for(int [] row:grid){
        for(int col:row) out.print(col+" ");
        out.println();}
```

Loading new arrays, example #2

- Here is an example of data file input.
- The first two values indicate the number of rows and columns, followed by the matrix elements.

s.java | nums.in

```
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("nums.in"));
    int rows = f.nextInt();
    int cols = f.nextInt();
    int[][]grid = new int[rows][cols];
    for(int r=0;r<rows;r++)
        for(int c=0;c<cols;c++)
            grid[r][c]=f.nextInt();
    for(int [] row:grid){
        for(int col:row) out.print(col+" ");
        out.println();}
```

Matrices.java | nums.in

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

C:\Program Files (x...

```
3 5 4 6 7
1 2 3 4 6
8 6 4 3 5
9 9 8 8 5
Press any key to con
```

Matrix processing – Example #1

- Now that we know how to construct and load data into matrices, let's look at some classic processing techniques.
- This one totals all the values in the matrix, using a method with a traditional nested ***for loop*** process.

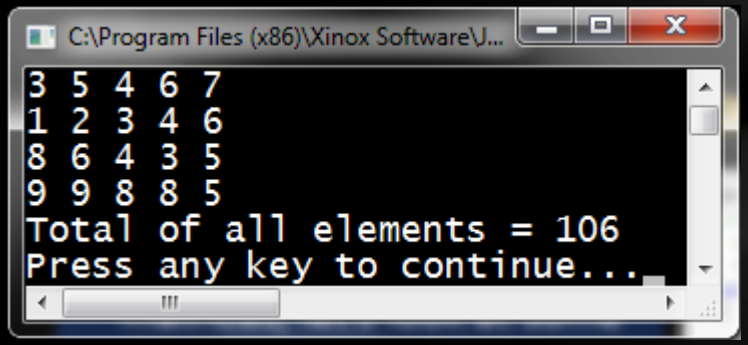
```
C:\Program Files (x86)\Xinox Software\J...
3 5 4 6 7
1 2 3 4 6
8 6 4 3 5
9 9 8 8 5
Total of all elements = 106
Press any key to continue...
```

ces.java × nums.in

```
/**Method to total all grid elements
 *using traditional nested for loops
 */
public static int gridTotalA(int[][]g)
{
    int sum = 0;
    for(int r=0;r<g.length;r++)
        for(int c=0;c<g[r].length;c++)
            sum+=g[r][c];
    return sum;
}
/**main method
 */
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("nums.in"));
    int rows = f.nextInt();
    int cols = f.nextInt();
    int[][]grid = new int[rows][cols];
    for(int r=0;r<rows;r++)
        for(int c=0;c<cols;c++)
            grid[r][c]=f.nextInt();
    for(int [] row:grid){
        for(int col:row) out.print(col+" ");
        out.println();}
    out.printf("Total of all elements = %d\n",
        gridTotalA(grid));
```

Matrix processing – Example #2

- This example also shows a summation method using the nested ***for each*** loop process

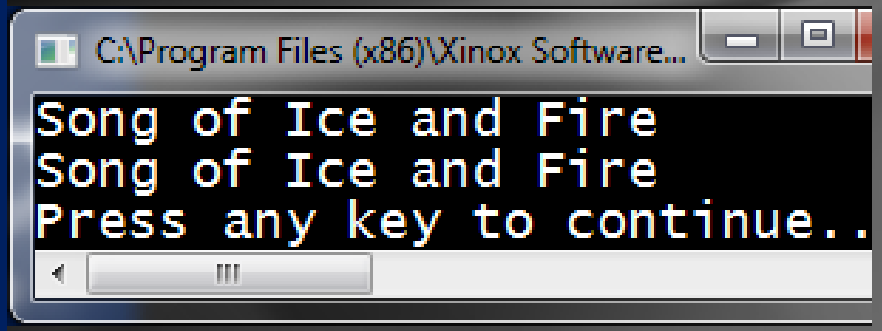



```
.java ×  nums.in
  /**Method to total all grid elements
   *using nested for each loops
   */
public static int gridTotalB(int[][]g)
{
    int sum = 0;
    for(int [] row:g)
        for(int col:row) sum+=col;
    return sum;
}
/**main method
 */
public static void main (String [] args)
    throws IOException
{
    Scanner f = new Scanner(new File("nums.in"));
    int rows = f.nextInt();
    int cols = f.nextInt();
    int[][]grid = new int[rows][cols];
    for(int r=0;r<rows;r++)
        for(int c=0;c<cols;c++)
            grid[r][c]=f.nextInt();
    for(int [] row:grid){
        for(int col:row) out.print(col+" ");
        out.println();}
    out.printf("Total of all elements = %d\n",
        gridTotalB(grid));
```

Matrix processing – Example #3

- Here is an unusual example in that the structure shown is a 1D array of Strings.
- However, since a String is in a sense an array of characters, this actually a **2D array of characters**.
- The first two outputs are a review of the two types of loops that can be used to output an array.
- The next slide shows the 2D nature of the structure.

```
throws IOException

String [] list = {"Song","of","Ice","and","Fire"};
for(int x=0;x<list.length;x++)
    out.print(list[x]+" ");
out.println();
for(String s:list)
    out.print(s+" ");
out.println();
```



Matrix processing – Example #3

- The two-dimensional character array becomes evident in this example, with each word treated as a row of characters, separated in each output…

```
S-o-n-g-
o-f-
I-c-e-
a-n-d-
F-i-r-e-

S-o-n-g-
o-f-
I-c-e-
a-n-d-
F-i-r-e-

g-n-o-S-
f-o-
e-c-I-
d-n-a-
e-r-i-F-
Press any ke
```

.java × nums.in

```
public static void main (String [] args)
    throws IOException
{
    String [] list = {"Song","of","Ice","and","Fire"};
    for(int x=0;x<list.length;x++)
    {   for(int y = 0;y<list[x].length();y++)
        out.print(list[x].charAt(y)+"-");
    out.println();
    }
    out.println();
    for(String s:list)
    {
        char [] a = s.toCharArray();
        for(char c:a)
            out.print(c+"-");
        out.println();
    }
    out.println();
    for(int x=0;x<list.length;x++)
    {   for(int y = list[x].length()-1;y>=0;y--)
            out.print(list[x].charAt(y)+"-");
        out.println();
    }
```

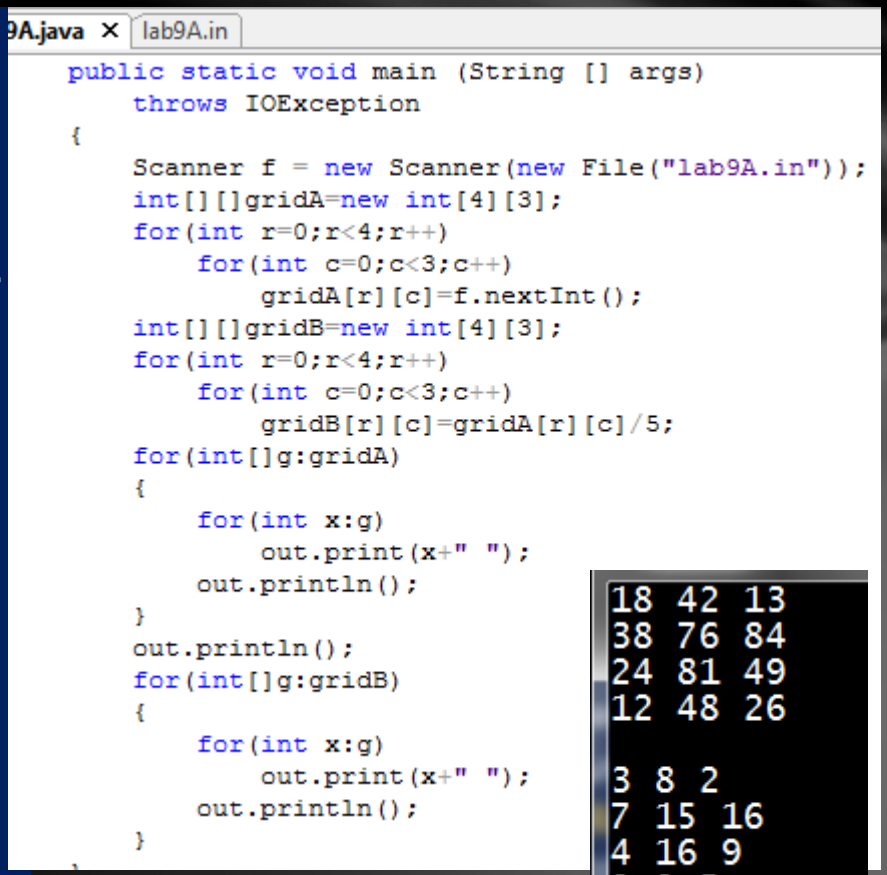
- …with an interesting twist at the end…check it out!

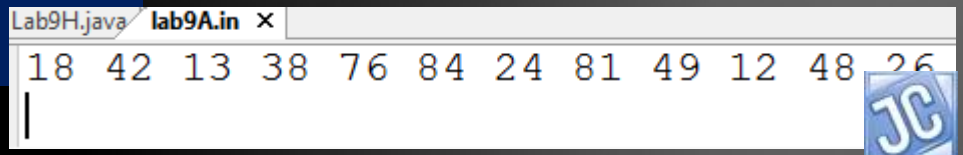
Lesson Summary

- In this lesson, you learned about constructing, loading, and processing 2-dimensional arrays.
- You used the nested loop processes learned earlier.
- Now it is time to practice with several examples.

Lab 9A

- WAP to manage two grids.
- Read 12 numbers from a file ("***lab9A.in***") into a 4 X 3 TWO DIMENSIONAL array, or matrix (4 rows, 3 columns). Call this ***gridA***.
- Create another same size matrix ***gridB***, divide each element in ***gridA*** by five and store the result in the ***gridB***.
- Output both grids as shown.
- A suggested solution is shown.





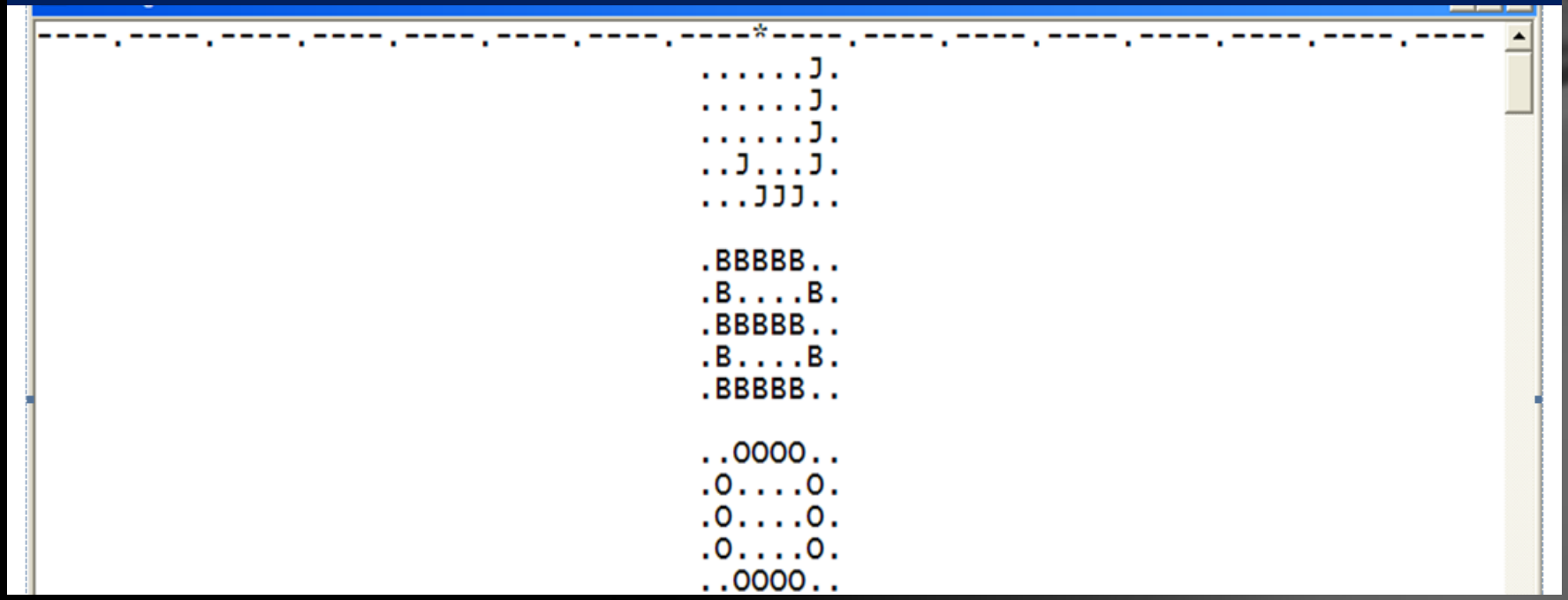
Lab 9B

- WAP to load your three initials into three different matrixes of characters, each measuring 5 rows by 8 columns.
- You must use loops at every possible opportunity.
- The basic code for the letter 'J' might look like this.
- The required final output is shown on the next slide.

```
public static void main (String [] args)
    throws IOException
{
    char [][] let1 = new char[5][8];
    for(int r=0;r<5;r++)
        Arrays.fill(let1[r],'.');
    let1[3][2]='J';
    for(int c=3;c<=5;c++)
        let1[4][c]='J';
    for(int r=0;r<=3;r++)
        let1[r][6]='J';
    for(char[]ca:let1)
    {
        for(char c:ca)
            out.print(c);
        out.println();
    }
}
```

Lab 9B(cont)

- The final output shows all three letters centered with a metric line above the first initial. The output for the initials JBO is shown.



Lab 9C

- Adapt the work you did for Lab 9B to output the three initials in horizontal fashion, as shown below, again with the metric line to verify centering.
- *Hint: Use one row loop, with three column loops nested inside it.*

```
                        ......J.   .BBBBB..   ..OOOO..
                        ......J.   .B....B.   .O....O.
                        ......J.   .BBBBB..   .O....O.
                        ..J...J.   .B....B.   .O....O.
                        ...JJJ..   .BBBBB..   ..OOOO..
Press any key to continue...
```

Lab 9D

- WAP to read four numbers (A, B, C, D) from a data file ("***lab9d.in***"). Use the numbers to create an A X B integer matrix stored in a two-dimensional array such that **each element is the sum of its index values**.
- For the numbers 5 8 4 5, the matrix will be a 5 X 8 grid. In the matrix, each element will be the sum of its index values, so the value in row `0`, column `0` will be `0+0`, or `0`. Row `4`, column `3` will contain the number `7 (4+3)`.
- Once the array is constructed and loaded, generate the four outputs shown below.
  - The values of C and D (`4` and `5` in the sample) are used for reports `2` and `3`.
  - (continued on next slide)

Lab 9D(cont)

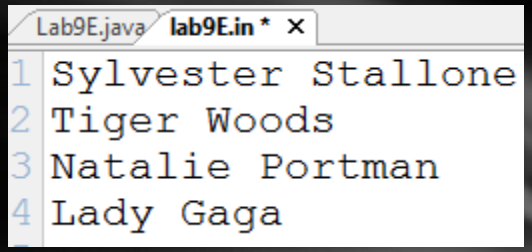
- Output 1: print the A by B matrix contents
- Output 2: print the elements in row C
- Output 3: print the elements in col D
- Output 4: calculate and print the total of all the elements in the matrix

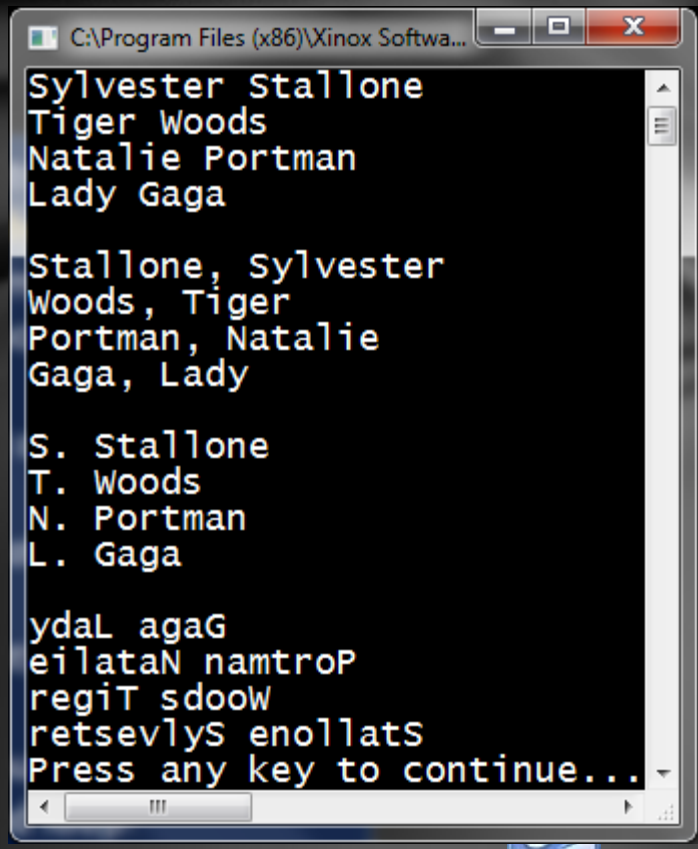
```
Contents of the matrix
-----------------------
  0  1  2  3  4  5  6  7
  1  2  3  4  5  6  7  8
  2  3  4  5  6  7  8  9
  3  4  5  6  7  8  9 10
  4  5  6  7  8  9 10 11

Contents of Row 4
-----------------------
  4  5  6  7  8  9 10 11

Contents of Column 5
-----------------------
  5  6  7  8  9

Matrix total  = 220
-----------------------
Press any key to continue...
```

Lab 9E

WAP to create a 4 X 2 matrix of Strings. Read from a data file ("***lab9E.in***")into the array the first and last name of several famous people and produce each of the following outputs.

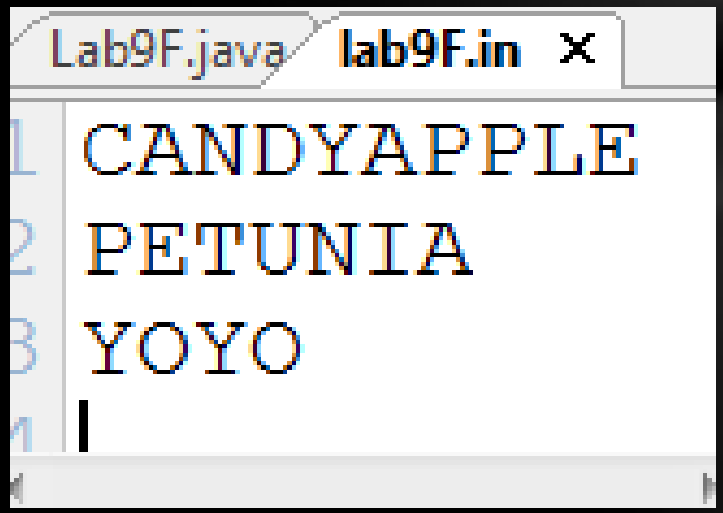
- Output 1: print the list of names (first name first)
- Ouput 2: print the list of names (last name first)
- Output 3: print the first initial and last name of each person
- Output 4: see the output to figure it out…*hint: triple-nested loop!*

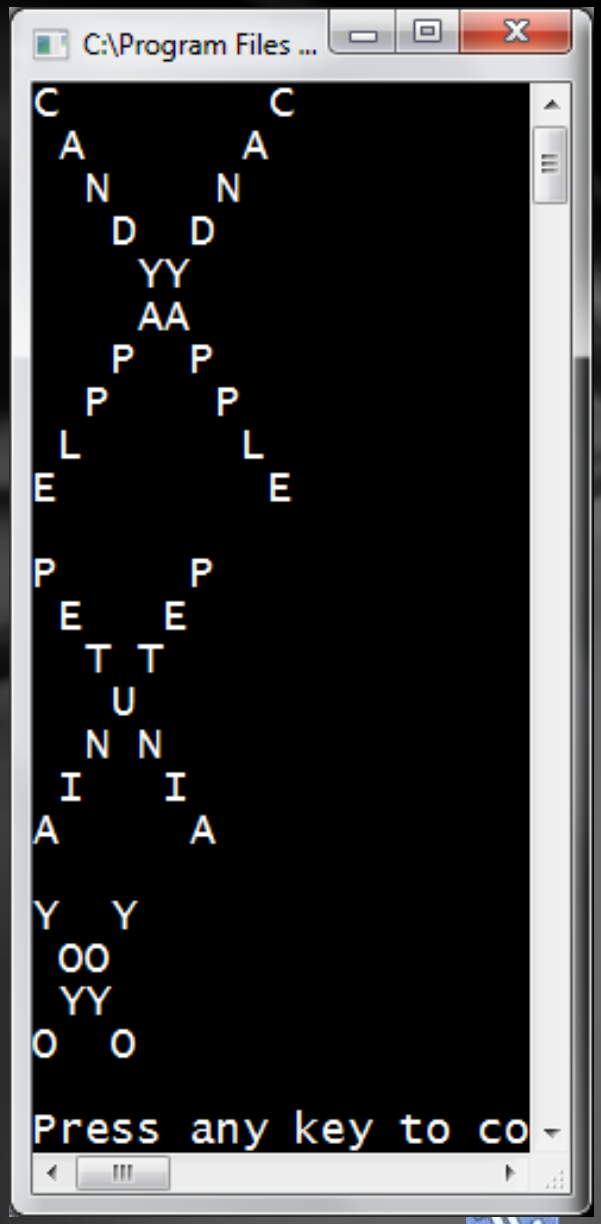




Lab 9F

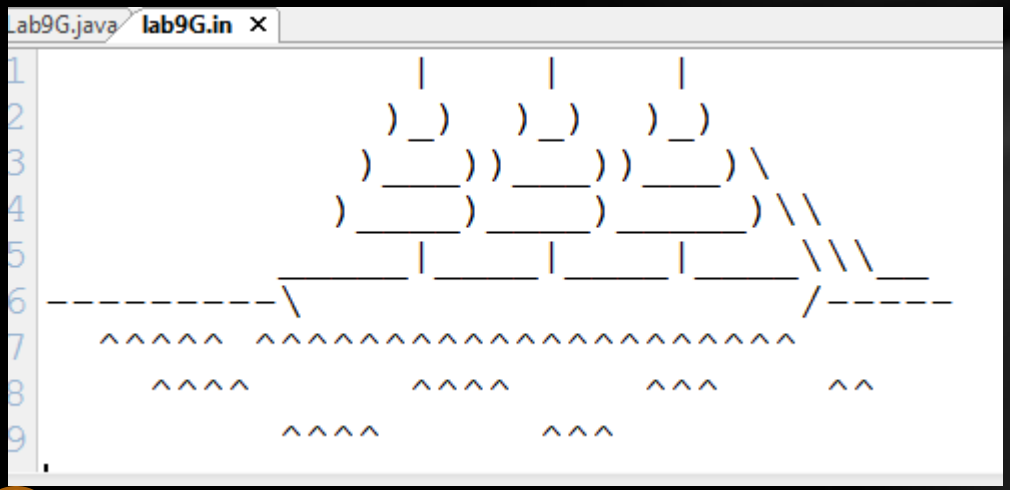
WAP to input several Strings from a data file ("***lab9F.in***") (use the ***while hasNext*** file input technique), create an X-diagonal formation in a character matrix for each String, and output as shown here.

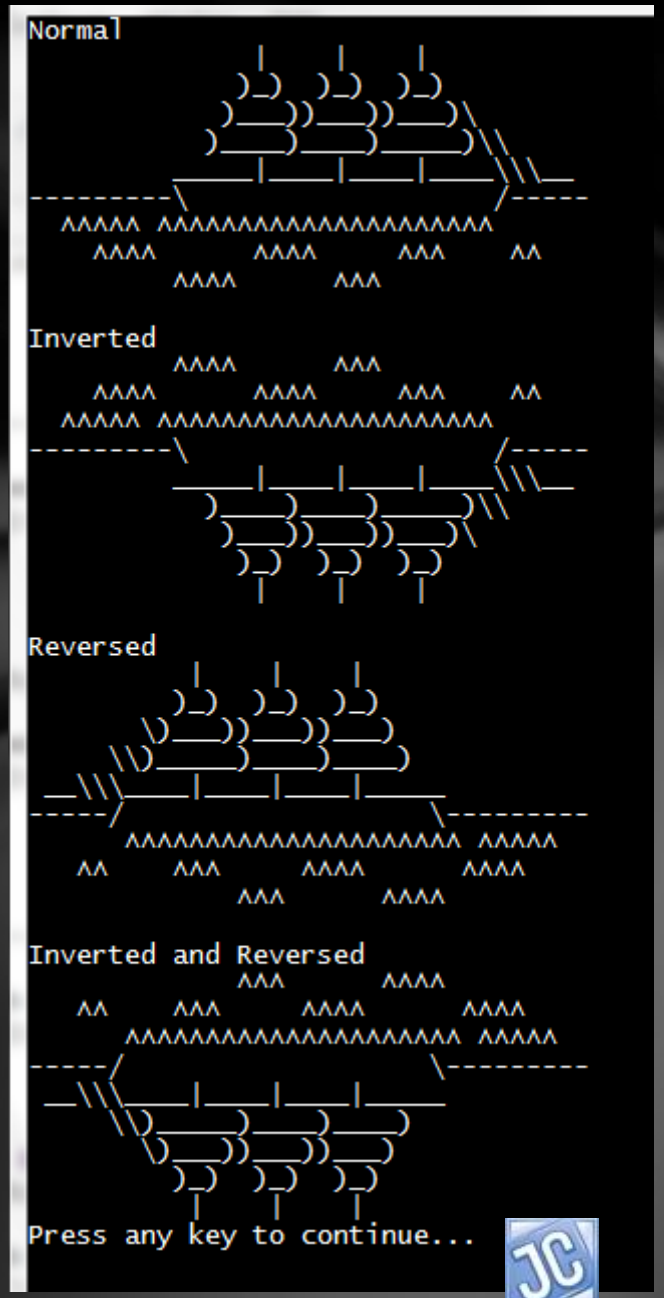




Lab 9G

WAP to load an interesting ASCII picture (max size 25 rows, 40 columns) into a character matrix, and output the picture 4 different ways: normal, inverted, reversed, both inverted and reversed.





Lab 9H

- And now for ***the classic*** matrix algorithm…***multiplying matrices***! This is a process usually first learned in Algebra II, involving a very specific set of rules.

- It is easily accomplished on the graphing calculator because someone wrote a program to do that. *You will now write that program!*

- If you don't know the process, ask your algebra teacher, or look it up!

Lab 9H

- Your job is to write a method that receives two integer matrices as parameters, and returns a new integer matrix containing the product of the two parameter matrices.

- It will be guaranteed in the precondition to the method that the two matrices passed as parameters will be "compatible" as far as the size requirements (columns of the first must match rows of the second).

Lab 9H (cont)

- Due to the fairly difficult nature of this algorithm, most of the solution is shown below and on the next slide. All that is left for you to do is to figure out the calculation for the triple nested loop process you see below that is required to actually "do the math"! Good luck!

```
Lab9H.j
1  imp
2  imp
3  imp
4  /**
5   *I
6   */
7  pub
8  {
9      /**Matrix multiplication method
0       *Precondition: Both matrix parameters are "compatible",
1       *i.e., the number of columns in agrid matches the number
2       *of rows in bgrid, a requirement for matrix multiplication
3       *The resulting matrix will have the number of rows for the
4       *agrid matrix and the number of columns of the bgrid matrix.
5       */
6      public static int[][] matMult(int[][]agrid,int[][]bgrid)
7      {
8          int rows = agrid.length;
9          int cols = bgrid[0].length;
0          int[][]cgrid=new int[rows][cols];
1          for(int r=0;r<rows;r++)
2              for(int c=0;c<cols;c++)
3                  for(int x=0;x<agrid[0].length;x++)
4                      cgrid[r][c]+=
5          return cgrid;
6      }
7      /**main method
8       */
       public static void main (String [] args)
           throws IOException
```

Lab9H.java | lab9H.in

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

```
|3|2|1|4|
|4|5|7|2|
|6|9|5|1|

TIMES

|5|9|4|
|6|2|8|
|7|1|6|
|8|7|2|

EQUALS

|66|60|42|
|115|67|102|
|127|84|128|
Press any key
```

Lab 9H (cont)

- Here is most of the main method.

Lab9H.java | lab9H.in

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

```
|3|2|1|4|
|4|5|7|2|
|6|9|5|1|

TIMES

|5|9|4|
|6|2|8|
|7|1|6|
|8|7|2|

EQUALS

|66|60|42|
|115|67|102|
|127|84|128|
Press any key
```

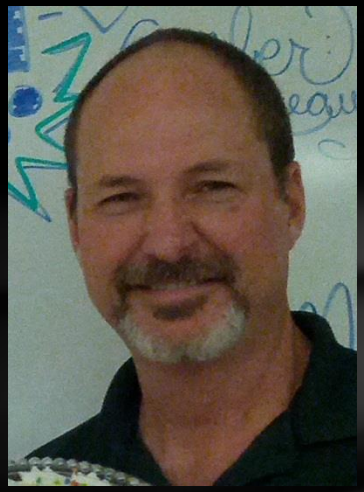
Lab9H.java | lab9H.in

```
26      }
27      /**main method
28       */
29      public static void main (String [] args)
30          throws IOException
31      {
32          Scanner f = new Scanner(new File("lab9H.in"));
            int [][]alphaGrid = new int[f.nextInt()][f.nextInt()];
            for(int r=0;r<alphaGrid.length;r++)
                for(int c=0;c<alphaGrid[0].length;c++)
                    alphaGrid[r][c]=f.nextInt();
            int [][]bravoGrid = new int[f.nextInt()][f.nextInt()];
            for(int r=0;r<bravoGrid.length;r++)
                for(int c=0;c<bravoGrid[0].length;c++)
                    bravoGrid[r][c]=f.nextInt();
            int [][] charlieGrid = matMult(alphaGrid,bravoGrid);
            for(int r=0;r<alphaGrid.length;r++)
            {   out.print("|");
                for(int c=0;c<alphaGrid[0].length;c++)
                    out.print(alphaGrid[r][c]+"|");
                out.println();
            }
            out.println("\nTIMES\n");
            for(int r=0;r<bravoGrid.length;r++)
            {   out.print("|");
                for(int c=0;c<bravoGrid[0].length;c++)
                    out.print(bravoGrid[r][c]+"|");
                out.println();
            }
            out.println("\nEQUALS\n");
```

CONGRATULATIONS!

- You now know how to work with 2D arrays.
- *Lesson 10 will begin the OOP adventure, the journey into the world of Object Oriented Programming.*

Thank you, and have fun!



To order supplementary materials for all the lessons in this package, such as lab solutions, quizzes, tests, and unit reviews, visit the O(N)CS Lessons website, or contact me at

John B. Owen
captainjbo@gmail.com