# STUDENT OUTLINE

## Lesson 21 – Two-Dimensional Arrays

**INTRODUCTION:**

Two-dimensional arrays allow the programmer to solve problems involving rows and columns. Many data processing problems involve rows and columns, such as an airplane reservation system or the mathematical modeling of bacteria growth. A classic problem involving two-dimensional arrays is the bacteria simulation program presented in the lab exercise, *Life.* After surveying the syntax and unique aspects of these larger data structures, such information will be applied to more challenging lab exercises.

The key topics for this lesson are:

A. Two-Dimensional Arrays
B. Passing Two-Dimensional Arrays to Methods
C. Two-Dimensional Array Algorithms

**VOCABULARY:**

| | |
|---|---|
| MATRIX | ROW |
| COLUMN | LENGTH |

**DISCUSSION:**

A. Two-Dimensional Arrays

1. Often the data a program uses comes from a two dimensional situation. For example, maps are two-dimensional (or more), the layout of a printed page is two-dimensional, a computer-generated image (such as on your computer's screen) is two dimensional, and so on.

   For these situations, a Java programmer can use *a two-dimensional array*. This allows for the creation of table-like data structures with a row and column format. The first subscript will define a row of a table with the second subscript defining a column of a table. Here is an example program including a diagram of the array.

   Program 21-1

```
class ArrayExample
{
  public static void main (String[] args)
  {
    int[][] table = new int[3][4];
    int  row, col;

    for (row = 0; row < 3; row++)
      for (col = 0; col < 4; col++)
        table[row][col] = row + col;
  }
}
```


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

**table**

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 |
| 1 | 1 | 2 | 3 | 4 |
| 2 | 2 | 3 | 4 | 5 |

2. Two-dimensional arrays are objects. A variable such as `table` is a reference to a 2D array object. The declaration

```
int[][] table;
```

says that `table` can hold a reference to a 2D array of integers. Without any further initialization, it will start out holding **`null`**.

3. The declaration

```
int[][]table = new int[3][4];
```

says that `table` can hold a reference to a 2D array of integers, creates an array object of 3 rows and 4 columns, and puts the reference in `table`. All the elements of the array are initialized to zero.

4. The declaration

```
int[][] table = { {0,0,0,0},
                  {0,0,0,0},
                  {0,0,0,0} };
```

does exactly the same thing as the previous declaration (and would not ordinarily be used.)

5. The declaration

```
int[][]table = { {0,1,2,3},
                 {1,2,3,4},
                 {2,3,4,5} };
```

creates an array of the same dimensions (same number of rows and columns) as the previous array and initializes the elements to the same values in each cell.

6. If no initializer is provided for an array, then when the array is created it is automatically filled with the appropriate value: zero for numbers, **`false`** for **`boolean`**, and **`null`** for objects.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

7. Just as with one-dimensional arrays, the row and column numbering of a 2-D array begin at subscript location zero (0). The 3 rows of the table are referenced from rows 0...2. Likewise, the 4 columns of the table are referenced from columns 0...3.

8. This particular two-dimensional array `table` is filled with the sums of `row` and `col`, which is accomplished by Program 21-2. To access each location of the matrix, specify the row coordinate first, then the column:

   ```
   table[row][col]
   ```

   Each subscript must have its own square brackets.

9. The length of a 2D array is the number of *rows* it has. The row index will run from 0 to length-1. The number of rows in `table` are given by the value `table.length`.

   Each row of a 2D array has its own length. To get the number of columns in `table`, use any of the following:

   ```
   table[0].length
   table[1].length
   table[2].length.
   ```

   There is actually no rule that says that all the rows of an array must have the same length, and some advanced applications of arrays use varying-sized rows. But if you use the **`new`** operator to create an array in the manner described above, you'll always get an array with equal-sized rows.

10. The routine that assigned values to the array used the specific numbers of rows and columns. That is fine for this particular program, but a better definition would work for an array of any two dimensions.

    <u>Program 21-2</u>

    ```
    class ArrayExample2
    {
      public static void main (String[] args)
      {
        int[][] table = new int[3][4];
        int  row, col;

        for (row = 0; row < table.length; row++)
          for (col = 0; col < table[row].length; col++)
            table[row][col] = row + col;
      }
    }
    ```

    In Program 21-2, the limits of the **`for`** loops have been redefined using `table.length` and `table[row].length` so that they work with *any* two-dimensional array of **`int`**s with any number of rows and columns .


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

B. Passing Two-Dimensional Arrays to Methods

1. The following program will illustrate parameter passing of an array. The purpose of this program is to read a text file containing integer data, store it in a 2-D array, and print it out. The contents of the text file "*data.txt*" is shown first:

"*data.txt*"

```
17   3   2  13
 5  10  11   8
 9   6   7  12
 4  15  14   1
```

Program 21-3

```
// A program to illustrate 2D array parameter passing

import chn.util.*;
import apcslib.*;

class Test2D
{
  public void printTable (int[][] pTable)
  {
    for (int row = 0; row < pTable.length; row++)
    {
      for (int col = 0; col < pTable[row].length; col++)
        System.out.print(Format.right(pTable[row][col], 4));
      System.out.println();
    }
  }

  public void loadTable (int[][] lTable)
  {
    FileInput inFile = new FileInput("data.txt");

    for (int row = 0; row < lTable.length; row++)
      for (int col = 0; col < lTable[row].length; col++)
        lTable[row][col] = inFile.readInt();
  }

  public static void main (String[] args)
  {
    final int MAX = 4;

    int[][] grid = new int[MAX][MAX];

    Test2D test = new Test2D();

    test.loadTable(grid);
    test.printTable(grid);
  }
}
```

2. The `loadTable` and `printTable` methods each use a reference parameter, (**int**`[][] lTable` and **int**`[][] pTable` respectively). The local identifiers `lTable` and `pTable` serve as aliases for the actual parameter `grid` passed to the methods.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

3. When a program is running and it tries to access an element of an array, the Java virtual machine checks that the array element actually exists. This is called *bounds checking*. If your program tries to access an array element that does not exist, the Java virtual machine will generate an `ArrayIndexOutOfBoundsException` exception. Ordinarily, this will halt your program.

C. Two-Dimensional Array Algorithms

1. The most common 2-D array algorithms will involve processing the entire grid, usually row-by-row or column-by-column.

2. Problem-solving on a matrix could involve processing:

   a. one row
   b. one column
   c. one cell
   d. adjacent cells in various different directions

3. In the next lesson we will look at a 2-D recursive solution to a rather difficult problem.

**SUMMARY/ REVIEW:** Two-dimensional arrays will be applied to two interesting problems. The simulation of life in a petri dish of bacteria will require a two-dimensional array representation. The second and third lab exercises are different versions of the "Knight's Tour" problem, an interesting and demanding chess movement problem.

**ASSIGNMENT:**

Lab Exercise, L.A.21.1, *Life*
Lab Exercise, L.A.21.2, *Knight's Tour 1*
Lab Exercise, L.A.21.3, *Knight's Tour 2*


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

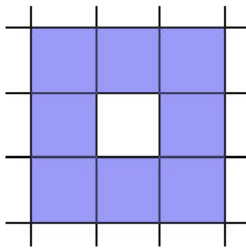
# LAB EXERCISE

## Life

**Background:**

The "Game of Life"[1] is a computer simulation of the life and death events of a population of organisms. This program will determine the life, death, and survival of, for example, bacteria from one generation to the next, assuming the starting grid of bacteria is generation zero. The rules for the creation of the next generation are as follows:

1. A "neighbor" of a cell is defined as any cell touching that cell, for example the eight blue cells in the diagram are the neighbors of the cell in the middle.



2. Every empty cell with three living neighbors will come to life in the next generation (a "birth").

3. Any cell with one or zero neighbors will die of loneliness, while any cell with four or more neighbors will die from overcrowding (a "death").

4. Any cell with two or three neighbors will live into the next generation (no change).

5. All births and deaths occur simultaneously.

**Assignment:**

1. Write a program that solves the game of Life. The size of the grid will be a square 20 x 20.

2. The original grid of bacteria will be supplied to your program from a text file.

   a. The first line will contain the number (`N`) of bacteria locations in the file.

   b. What follows are `N` pairs of data, one pair of numbers per line.

   c. The first value of each line indicates the row location while the second value on the line indicates the column location.

   d. The data file values are given as: `1 <= Row <= 20` and `1 <= Col <= 20`.

---

[1] The Game of Life was invented by John H. Conway of Princeton University. It was first described to a wide audience by Martin Gardner in his "Mathematical Games" column in *Scientific American*, October, 1970.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

3. After your program has initialized the grid with generation 0, your program must allow Life to proceed for 5 generations.

4. Display the final results on the screen and determine the following statistical information:

    a. The number of living cells in the entire board.

    b. The number of living cells in row 10.

    c. The number of living cells in column 10.

**<u>Instructions:</u>**

1. A sample run output is given below. Note, these <u>are</u> the correct answers if you use the provided data file "*life100.txt*".

```
       12345678901234567890

 1           **
 2          * *
 3           *        **
 4                  **  *
 5                  ** **
 6                 * * **
 7               * **
 8               *
 9            ***  **
10          ** ** ** **
11          ******
12           ** **** **
13            *  **  ***
14           *  * *** **
15            **     ** *
16           **      **  *
17          * *   **     *
18          * *   ***    *
19          *    *  * **
20

Number in Row 10 ---> 8

Number in Column 10 ---> 5

Number of living organisms ---> 88
```


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

# LAB EXERCISE

## Knight's Tour 1

**Background:**

The Swiss mathematician Leonhard Euler (1707 – 1783) proposed a problem regarding the chess piece called the knight. The challenge that Euler proposed is to move the knight around an empty chessboard, touching each of the 64 squares once and only once. You may start the knight at any position on the board and move it using its standard L-shaped moves (two over in one direction, over one in a perpendicular direction). Try it on this empty grid. Number any position as 1 and then visit as many squares as possible, numbering as you go:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |

More difficult than it seems!

**Assignment:**

Your task in this lab is to write a program that will move a knight around an empty chess board, leaving behind a trail of increasing integers, ranging from 1 to, hopefully, 64. Here are the specifications for your assignment:

1. The knight will start in row 1, column 1.

2. The program will mark squares as they are visited, ranging from 1-64.

3. The program will continue until a complete tour is accomplished (all 64 squares) or the program gets stuck with nowhere to go.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

4. The program will print the results, looking something like this:

```
      1    2    3    4    5    6    7    8

 1    1    0   21    0    0   14   23   12
 2   20    0    6    9   22   11    0    0
 3    7    2   19   36   15   46   13   24
 4    0    5    8   47   10   37    0   45
 5    0   18    3   16   35   44   25   38
 6    4   31   34    0   42   39   28    0
 7    0    0   17   32   29   26   43   40
 8    0   33   30    0    0   41    0   27

47 squares were visited
```

5. Use the `Random` class described previously in Lesson 18 to generate the necessary random numbers.

6. Here are two suggestions to solve this lab.

Suggestion 1: Here is an idea on how to deal with the 8 different possible moves. If we analyze the possible moves we can break each move down into a horizontal and vertical component.

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  | **8** |  | **1** |  |  |  |
| 3 |  | **7** |  |  |  | **2** |  |  |
| 4 |  |  |  | **K** |  |  |  |  |
| 5 |  | **6** |  |  |  | **3** |  |  |
| 6 |  |  | **5** |  | **4** |  |  |  |
| 7 |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |

Here are the moves analyzed as horizontal and vertical components:

| move | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| horizontal | +1 | +2 | +2 | +1 | -1 | -2 | -2 | -1 |
| vertical | -2 | -1 | +1 | +2 | +2 | +1 | -1 | -2 |


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

If you stored the above data in 2 arrays called horizontal and vertical, it would be possible to move the knight to the next square using a statement like:

```
row = row + vertical[moveNumber];
col = col + horizontal[moveNumber];
```

This kind of approach will simplify your program.

Suggestion 2: Declare the board as a 9 x 9 grid. This will allow you to work with rows 1..8 and column 1..8. Row 0 and column 0 will not be used in this approach.

7. Turn in your source code and a run output as described above.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

# LAB EXERCISE

## Knight's Tour 2

**Background:**

Rather than trying a random approach to solve for the next move we will develop an algorithm that uses some information and logic about each square. As you played the game you should have noticed that the edges were more difficult to visit and the corners the most difficult of all. If we analyze each square we will notice that some are more accessible than others. Here is an analysis of the accessibility of each square.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |
| 2 | 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 3 | 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 4 | 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 5 | 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 6 | 4 | 6 | 8 | 8 | 8 | 8 | 6 | 4 |
| 7 | 3 | 4 | 6 | 6 | 6 | 6 | 4 | 3 |
| 8 | 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |

A square with an accessibility of 8 means that it can be approached from 8 different other squares. A corner square is rated at 2, while the edges are rated at 3 or 4. It makes sense to try and visit squares with lower accessibility values first, leaving the more accessible middle squares for later in the algorithm.

**Assignment:**

1. Write a revised version of the Knight's Tour program using the accessibility strategy. In determining the next move, the knight should move to the square with the lowest accessibility value. In the case of a tie you may move the knight to any of the tied squares.

2. The original accessibility information is stored in an 8-line text file called *access.txt*. Each line consists of the 8 accessibility numbers for that line, separated by blank spaces, terminated with the enter key. You should read this data file to set up your starting accessibility data table. It would be a good idea to test your initialization of the accessibility table.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)

3. To ensure a greater degree of success, as the knight moves around the board you should reduce the accessibility numbers in the appropriate squares. For example, if `location[4][5]` is visited, then the 8 squares that can be reached from `location[4][5]` should have their accessibility values reduced by 1.

4. Proper decomposition of the problem into single-purpose methods will be of great importance in this lab exercise.

5. Turn in your source code and a run output with the highest number of visited squares.


© ICT 2003, www.ict.org, All Rights Reserved
Use permitted only by licensees in accordance
with license terms (http://www.ict.org/javalicense.pdf)