COMP102

Part 1

Data Types and Structures

Lecture 2

Multi-Dimensional Arrays

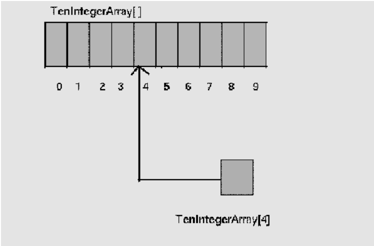
Array structures can be defined over any class of object instances

A further development of arrays is - **multi-dimensional** arrays.

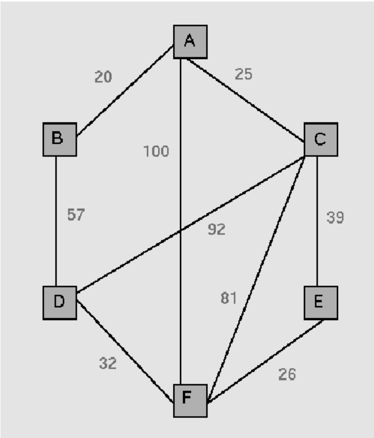
The form of arrays so far, has been **1-dimensional**

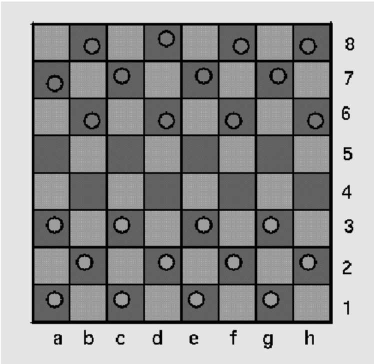
A 10 element array of integers, is realised by

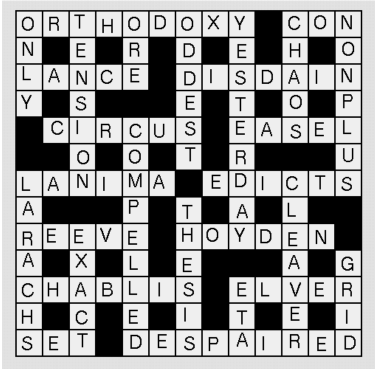
int [] TenIntegerArray = **new int**[10]



It is often more natural to think of data in an ordered collection by using 2 or more **indices**:







As a final example, recall the Giant Java example from COMP101. If we wished to represent a given letter using characters other than '*' and ' ' then a sequence of printed Strings is not very suitable.

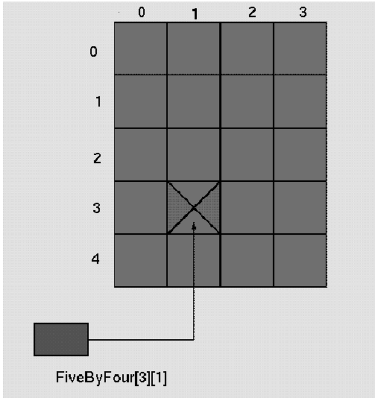
Furthermore it is not clear how to use such a form to print characters from **left to right**

Multi-dimensional Arrays in Java

Java allows arrays having 2,3,4 ... dimensions using a syntax that is similar for declaring arrays with 1 dimension:

An array of 5 rows and 4 columns, each entry of which is integer

int [] [] FiveByFour = **new int** [5] [4];



For this lecture we will consider examples involving constrained multi-dimensional arrays.

Consider the example declaration

int [] [] FiveByFour = **new int** [5] [4];

In general (for a constrained array) this would look like:

<Type> [] [] [] ... NameofArray=
new <Type> [LEN_1] [LEN_2] [LEN_3] $\cdots$;

Returning to the examples presented declarations for these would be,

int [] [] RoadMap = new int [6] [6];
String [] [] DraughtsBoard = new String [8] [8];
char [] [] CrossWordGrid = new char [13] [13];

Logically one can view,

1. A 1-dimensional array as an ordered collection of ‘cells’ (Primitive types or Objects).
2. A 2-dimensional array as an ordered collection of 1-dimensional arrays.
3. A 3-dimensional array as an ordered collection of 2-dimensional arrays

. . .

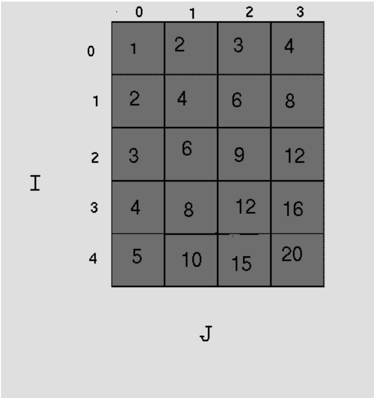
k. A **K**-dimensional array as an ordered collection of (K-1)-dimensional arrays.

Just as an aggregate of values can be used to set the starting values of a 1-dimensional array the same can be done with a multi-dimensional array.

For the example 2-dimensional integer array, if the element **FiveByFour[i][j]** is to hold $(i+1)*(j+1)$

```
int [][] FiveByFour = { {1,2,3,4},
                        {2,4,6,8},
                        {3,6,9,12},
                        {4,8,12,16},
                        {5,10,15,20} };
```

resulting in



Similarly for a 3-dimensional array, **FourByThreeByTwo** of integers with the element **[i][j][k]** initiated to the value $(i+1)*(j+1)*(k+1)$,


```
int [][][] FourByThreeByTwo = { {{1,2},
                                 {2,4},
                                 {3,6} },
                                {{2,4},
                                 {4,8},
                                 {6,12}},
                                {{3,6},
                                 {6,12},
                                 {9,18}},
                                {{4,8},
                                 {8,16},
                                 {12,24}}
                              };
```


Summary

1. Multi-dimensional array objects provide a convenient mechanism for modelling information in which references to a specific datum are naturally defined in terms of two, three, or more index values.
2. There is no limit to the number of dimensions that can be defined for a multi-dimensional array, although examples in which more than three are necessary are rare.
3. Multi-dimensional array objects can be viewed as simply a ‘special’ case of (1-dimensional) arrays of an instance of a compound Object, i.e. one in which the Object itself is an array (single or multi-dimensional).