DEPARTMENT OF COMPUTER SCIENCE

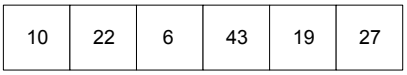
UCL

COMP1007
Arrays

Arrays

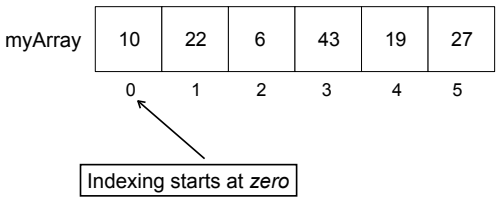
- A normal variable holds 1 value:

- An array variable holds a *sequence* of values:



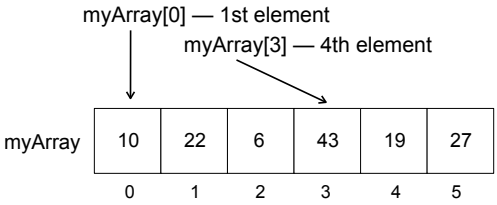
Naming arrays

- An array has a single name, so the *elements* are numbered or *indexed*.



Indexing

- An array element is accessed using the name and index number, like this:



- The square brackets are the *index operator*.

Using indexing

- We can fetch the value of an array element:

```
int n = myArray[2];
```

- Or assign to an array element:

```
myArray[3] = 10;
```


Why?

- Arrays allow a sequence of values of the same type to be stored using *one* variable.
- We don't have to name lots of variables.
- The array can be as big as we like (within limits).
- Elements can be accessed using loops.

Example:

- Add up a collection of numbers in array:

```
int sum = 0;
for (int n = 0 ; n < 6 ; n++)
{
   sum += anArray[n];
}
```


Enhanced for

enhanced for is new to Java 5

- Can also use the enhanced for loop:

```
int sum = 0;
for (int n : anArray)
{
  sum += n;
}
```

for each element n in anArray ...

int n — declares loop variable
: anArray — array to *iterate* over

Declaring an Array

```
int[] myArray = new int[6];
```

Type array of int — Create an array of 6 ints

- Note the two parts:
  - Declare the variable
  - Create the array

Size v. Index

- int[] myArray = new int[6];

myArray	10	22	6	43	19	27
	0	1	2	3	4	5

- This gives 6 elements indexed from *0 to 5*.

More declarations

```
double[] values = new double[100];
String[] name = new String[50];
boolean[] marks = new boolean[5000];

int n = <some expression>;
long[] numbers = new long[n];
```


Initialisation

```
int[] anArray = new int[10];
```

- The variable anArray is initialised,
- but what about the array elements?

- They are initialised to default values (e.g., 0,0.0, *null*)

Initialising array elements

- Either write a loop and assign to each element,
- or use an array initialisation expression:

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

or

```
int[] array = new int[]{1,2,3,4,5};
```

(Create an array of size 5, with each element initialised.)

Array length

- Arrays are actually *objects*.
- And they know their own size:

```
int[] n = new int[10];
…
int size = n.length;
// size == 10
```


Safer loops

```
int sum = 0;
for (int n = 0 ; n < anArray.length ; n++)
{
   sum += anArray[n];
}
```

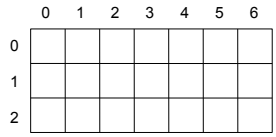
- No longer need to have a ‘magic number’.
- Instead ask array for its size.

But in Java 5 can do this:

```
int sum = 0;
for (int n : anArray)
{
  sum += n;
}
```


2 dimensions

- int[][] twoD = new int[3][7];



- 3 rows by 7 columns

2D indexing

- Assign to row 1 column 3

```
twoD[1][3] = 10;
```

- Fetch row 0 column 6

```
int x = twoD[0][6];
```

(Don’t forget we index from zero!)

2D loop (v1)

```
int sum = 0;
for (int row = 0 ; row < 3 ; row++)
{
    for (int col = 0 ; col < 7 ; col++)
    {
       sum += twoD[row][col];
    }
}
```

2D loop (v2)

```
int sum = 0;
for (int row = 0 ; row < twoD.length ; row++)
{
    for (int col = 0 ; col < twoD[row].length ; col++)
    {
       sum += twoD[row][col];
    }
}
```

twoD.length gives number of rows

twoD[row].length gives number of columns in row

2D loop (v3) — enhanced for

```
int sum = 0;
for (int[] row : twoD)
{
   for (int n : row)
   {
      sum += n;
   }
}
```

for each row
  for each value in row
    add to sum

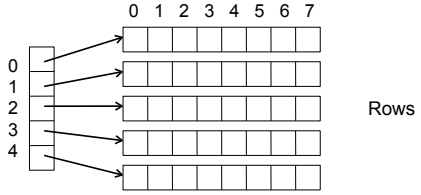
Data Structure

- An array is a *data structure.*
- A data structure is a collection of values, organised in a particular way.
  - An array is a *sequence* of values.
  - Can be accessed in sequence (loop).
  - Or randomly (index any element).
- An array is a basic data structure built into the language.
  - Special syntax.
  - But also see library classes Array and Arrays.

Questions?

2D Arrays – The Truth!

- A 2D array is really an array of arrays!

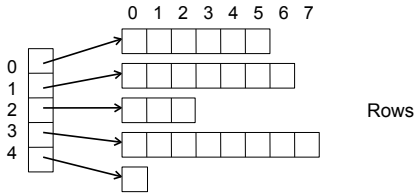


Array Fun…

```
int[] oned1 = new int[20]; // 1D array
int[][] twod = new int[10][]; // 2D array

twod[0] = oned1; // Add array as row

int i = twod[0][2]; // Can  now index

int[] oned2 = new int[50]; //  Another 1D array

twod[1] = oned2; // New row different length
twod[1][45] = 10;
int[] n = twod[1]; // Get row from array (slice array)
```

Ragged Array

- Different length rows, to avoid wasting space.



N-Dimensional Arrays

- In principle arrays can have as many dimensions as you want:

```
double[][][] d = new double[10][20][30];
```

- In practice, rarely need more than 3 dimensions.
- Large arrays also use lots of memory.

Array Summary

- Arrays allow collections of values to be stored in a single variable.
- New syntax with square brackets is used.
- Loops are used to work with arrays.

Containers

- Arrays are “built-in” to the Java language and directly supported by the syntax.
- There are also *Container classes*, providing data structure objects.
  - Also called *Collection classes*.
- Containers have different properties, variously optimised for convenience, speed and memory use.

Containers (2)

- The Java Collections Framework provides various container classes:
  - ArrayList
  - HashMap, HashSet
  - TreeSet, TreeMap
  - LinkedList
  - and others

ArrayList (Generic class)

```
import java.util.ArrayList;  // Note

ArrayList<String> a = new ArrayList<String>();
String s1 = "hello";
a.add(s1);
a.add(“world”);
String s = a.get(1);
```

ArrayList v. arrays

- Arrays are “manual”, ArrayList is more automated.
- ArrayList allows elements to be added and deleted from any position.
- ArrayLists change size automatically.

Non-generic ArrayList

- Can also use ArrayList like this:

```
import java.util.ArrayList;

ArrayList a = new ArrayList();
String s1 = "hello";
a.add(s1);
a.add(“world”);
String s = (String)a.get(1);
```

Cast expression to specify type of object returned.

Containers, Objects and Primitives

- A container class can store objects only.
  - Not primitive types.
- However, this can be done:

```
ArrayList<Integer> a = new ArrayList<Integer>();
a.add(1); // Adding a primitive value
...
int n = a.get(0); // Getting a primitve value
```


Auto-boxing/unboxing

- For each primitive type, there is a matching class:
  - Integer, Long, Double, Float, Boolean, etc.
  - objects of theses classes represent the values.
- Where possible the compiler will automatically add code to convert representations.
  - int -> Integer (boxing)
  - Integer -> int (unboxing)

Summary

- Container classes provide higher level abstractions for dealing with collections.
- But require more knowledge to be used effectively.
- By the end of 1008 you will know how to construct your own containers.