ARRAYS
CITS1001
Scope of this lecture

- Arrays
- Constructing arrays
- Arrays of 2 or more dimensions
Fixed-size collections

- ArrayList is used when the size of a collection is not known in advance

- But sometimes the maximum collection size can be predetermined.

- A special fixed-size collection type is available: an array.

- Unlike the flexible List collections, arrays can store object references or primitive-type values.

- Arrays use a special syntax
Arrays

- An *array* is an *indexed* sequence of variables of the same type.

```
```

- The *array* is called `a` – its *elements* are called `a[0]`, `a[1]`, `a[2]` and so on; each element is a separate variable.
Use of arrays

• Arrays are used when we have large numbers of identical objects that we want to operate on as a collection
  • A collection of student marks that we want to analyse
  • A collection of temperatures that we want to average
  • A collection of names that we want to sort

Warning: In Java, the indexing of an array starts at 0, so the first element of the array \texttt{a} is \texttt{a[0]}
Declaring arrays

- An array is declared using similar syntax as other variables:
  ```java
  int[] a;
  ```
  - Declares a to be a variable representing an array of ints
  ```java
  double[] temps;
  ```
  - Declares temps to be a variable representing an array of doubles
  ```java
  String[] names;
  ```
  - Declares names to be a variable representing an array of Strings
  ```java
  Student[] marks;
  ```
  - Declares marks to be a variable representing an array of Student objects
Creating Arrays

- An array is an *object* in a Java program
- Therefore the declaration simply creates a *reference* to refer to the array, but does not create the array itself
- Hence, the declaration
  ```java
  int[] a;
  ```
  creates a shoebox called *a*, big enough to hold an *object reference*, and currently set to the special value *null*

Only space for the array reference has been created, not the array itself
Creating arrays II

- In order to actually create the array, we need to use the keyword `new` (just like creating any other object)

  ```java
  int[] a;
  a = new int[7];
  ```

An object containing 7 variables of type `int` is created.
Creating arrays III

The seven variables do not have individual names, but are referred to by the *array name*, and an *index*
Referencing array elements

Array elements can be used in just the same way as any other variable of that type

\[
\begin{align*}
a[4] &= 15; \\
a[2] &= 7; \\
a[6] &= a[0] + 17;
\end{align*}
\]
Indexing array elements

- The power of arrays comes from the fact that the index can be a variable or expression

```java
int x = 3;
a[x] = 5;
a[7-x] = 44;
```
private int max(int[] a)
{
    int max;
    max = a[0];
    for (int ai : a)
    {
        if (ai > max)
        {
            max = ai;
        }
    }
    return max;
}
Arrays of objects

• An array can have any type as its base type, either primitive type or reference type

• When using an array of primitive type, there are two steps involved
  • Declare the variable to refer to the array
  • Create the space for the array elements using `new`

• When using an array of reference type, there are three steps involved
  • Declare the variable to refer to the array
  • Create the space for the array elements using `new`
  • Populate the array with objects by repeatedly using `new` in a loop
Arrays of Students

```java
public class Student {
    private String studentNumber;
    private int mark;

    public Student(String studentNumber, int mark) {
        this.studentNumber = studentNumber;
        this.mark = mark;
    }

    public String getStudentNumber() {
        return studentNumber;
    }

    public int getMark() {
        return mark;
    }
}
```

A skeleton version of a possible Student class in a student records system
Creating a unit list

```
Student[] unitList;  // Declare

unitList = new Student[numStudents];  // Create

unitList[0] = new Student("042371X",64);
unitList[1] = new Student("0499731",72);
unitList[2] = new Student("0400127",55);
...  // Populate
unitList[numStudents-1] = new Student("0401332",85);
```
The three steps
Using arrays of objects

- Using arrays of objects just requires remembering that each element of the array is (a reference to) an object

```java
public Student topStudent(Student[] unitList) {
    Student top;
    top = unitList[0];
    for (int i=1; i<unitList.length; i++) {
        if (unitList[i].getMark() > top.getMark()) {
            top = unitList[i];
        }
    }
    return top;
}
```
private Student top(Student[] unitList) {
    Student top = unitList[0];
    for (Student si : marks) {
        if (si.getMark() > top.getMark()) {
            top = si;
        }
    }
    return top;
}
Method signatures

- Method signatures

  \[\text{int max(int[]} a)\]

- and

  \[\text{Student topStudent(Student[] unitList)}\]
Initialization

```plaintext
Student top;

top = unitList[0];

int max;
max = a[0];

Declare a variable to hold the “best so far” and initialize it to the first element in the array
```
Initialization

for (int ai : a) {
    if (a[i] > max) {
        max = a[i];
    }
}

Check each element in turn, compare it with the best so far, and update the best so far if necessary

for (Student si : unitList) {
    if (si.getMark() > top.getMark()) {
        top = si;
    }
}

Return

```c
return max;
```

Finally *return* the extreme element – the highest `int` or the `Student` with the best mark

```c
return top;
```
Highest mark

• What if the user wishes to find the highest mark?

• Two options
  • Return the Student object, and let the user call getMark to find it out
  • Provide another simple method that uses topStudent as a helper

```java
public int highestMark(Student[] unitList) {
    return this.topStudent(unitList).getMark();
}
```
Common array issues

- The commonest problem with an array is using an array index that is out of bounds, either less than 0 or larger than the largest index.
- In particular, if we have an array:
  ```java
  int[] a = new int[6];
  ```
  then any attempt to access `a[6]` will generate an `ArrayIndexOutOfBoundsException` and execution will be terminated.
- Another common problem is *declaring* and *creating* an array of objects, but forgetting to populate it.
- Any attempt at making a method call such as `unitList[i].getMark()` will then result in a `NullPointerException`. 
Array literals

- The size is inferred from the data.

```java
private int[] numbers = { 3, 15, 4, 5 };
```

- Array literals in this form **can only be used in declarations**

- Related uses require `new`:

```java
numbers = new int[] {
    3, 15, 4, 5
};
```
Array length

private int[] numbers = { 3, 15, 4, 5 };

int n = numbers.length;

• **NB:** length is a field rather than a method!
• It cannot be changed – ‘fixed size’.
MORE THAN ONE DIMENSION
2-dimensional arrays

- We can declare and use arrays of more than one dimension!

```java
int[][] a = new int[4][3];
```

This creates an array with four “rows” and three “columns”

The “row” index ranges from 0-3 and the “column” index from 0-2
A 2-d array is really an “array of arrays” - an array of length 4, where each element is an array of length 3
2-dimensional arrays

Of course the computer does not actually store the entries in “rows” and so we should just view this 2d array as being 12 variables arranged in four groups of three.