Introduction to Programming

11. Arrays – part 2

Arrays in Java - revision

- We saw in part 1 of this lecture that
  - In Java, array types are *classes* (reference types), and each array is an *object*
  - An array object is an *indexed list* of elements of the same type (a primitive type or a reference type)
  - The name of the array type is written by stating the element type followed by one (or more) pair(s) of square brackets such as:

```
String[]  // type for arrays of Strings
```

  - An *array variable* can refer to an array object and is declared using the array type name:

```
String[] names;
```


Arrays – revision continued

- We also saw in part 1 of this lecture that
  - To create an array object, one can call the constructor for the array type and specify how many elements the array should contain

```
new String[10]
```

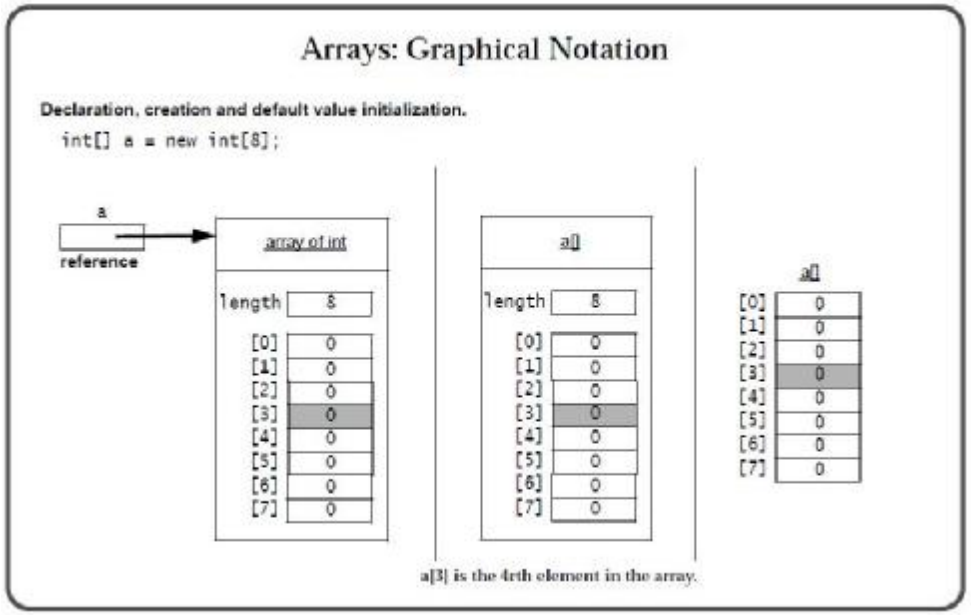
  - This creates an array object and returns a reference to it that can be assigned to an array variable:

```
String[] names = new String[10];
```

  - Each element of an array can be thought of as a variable of the element type (in this case, String) and each element is given a default value when the array is created (in this case, **`null`**)

Arrays – revision continued

- In Java, the index number of the first element in an array is always zero
- Arrays in Java have a field named `length` that is a constant and that records the number of elements the array has
- The index number of the last element in an array named a is at index number `a.length-1`
- The next slide, from a set that accompanies the book by Mughal, Hamre and Rasmussen, shows a variety of ways that one might graphically depict an array of int



Array revision continued

- Can initialise an array with a list of values when declaring the variable:

```
int[] primes = {2,3,5,7,11,13,17};
```

- Or later if including an explicit constructor call:

```
primes = new int[]{2,3,5,7,11,13,17};
```

  - Note: do not include the length of the array in the square brackets, this is calculated from the number of elements in the list

Array length

- The length of an array can be specified at runtime:

```
TextIO.put("Enter the array size: ");
int size = TextIO.getlnInt();
String[] names = new String[size];
```

- If the value of `size` is negative an exception is thrown, any other **`int`** value (including 0) is allowed

Array variable declarations

- Java allows you to put the square brackets after the variable name when declaring an array variable, instead of after the element type name

```
int primes[] = {2,3,5,7,11,13,17};
```

- Although legal, we will not declare array variables in this module by placing the brackets after the variable name but will always place them after the element type name - this is regarded as best practice in the Java community

Iterating over an array

- Can use a for loop to iterate over the array's elements

```
int[] primes = {2,3,5,7,11,13,17};
for (int i = 0; i < primes.length; i++) {
   TextIO.putln(primes[i] + " is prime");
}
```

  - Note: initialise the index to 0 and terminate the loop when the index is equal to the length of the array

The enhanced for loop

- If you do not need to update any of the array elements, Java provides an *enhanced for loop* (called a foreach loop in other languages) to iterate over the array's elements:

```
int[] primes = {2, 3, 5, 7, 11, 13, 17};
TextIO.put("The first seven primes are:");
for (int currentPrime : primes) {
   TextIO.put(" " + currentPrime);
}
```


The enhanced for loop cont'd

- Note the format of the loop:

Declare a variable of the element type

Array variable

```
for (int currentPrime : primes) {
   TextIO.put(" " + currentPrime);
}
```

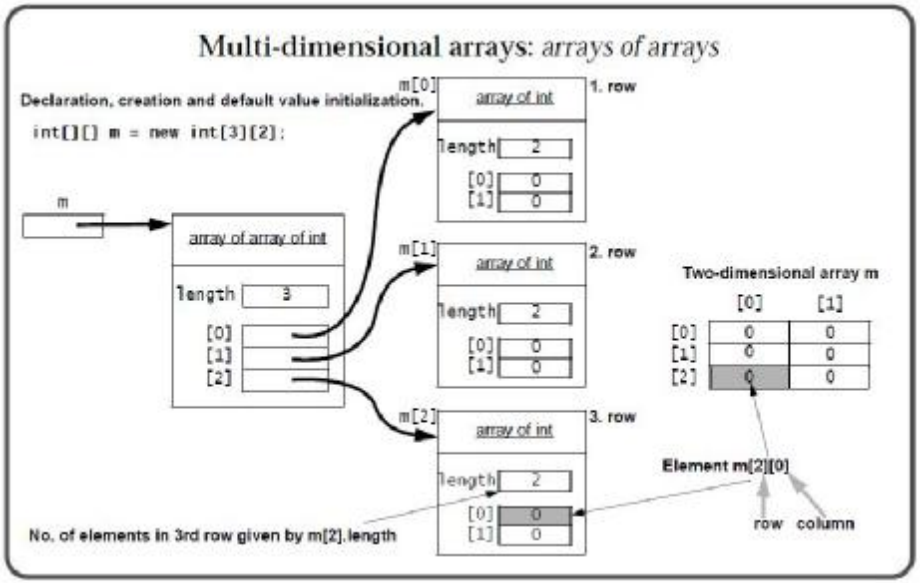
- The loop goes round as many times as there are elements in the array, once for each element.
- The variable (`currentPrime`) takes on the value of each element in turn.
- Note that you cannot use an enhanced for loop to change the value of an array element.

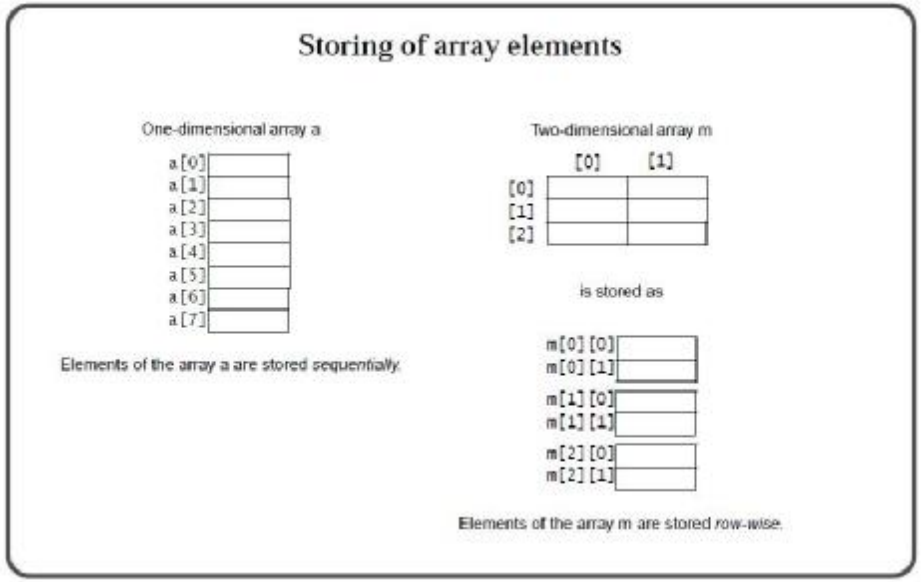
Two dimensional arrays

- The elements of an array can be of any type, including an array type
  - This allows for multidimensional arrays
- To declare a multidimensional array, include a pair of square brackets for each dimension

```
double[][] twoD = new double[5][5];
double[][][] threeD = new double[5][5][5];
```

- Next 2 slides by Mughal, Hamre and Rasmussen





Initialising a 2D array

- Can use initialization lists for two-dimensional arrays:

```
double[][] twoD = {   {1.0,0.0,0.0},
                      {0.0,1.0,0.0},
                      {0.0,0.0,1.0}
                  };  // a 3x3 array of double
```

- This is equivalent to:

```
double[][] twoD = new double[3][];
twoD[0] = new double[]{1.0,0.0,0.0};
twoD[1] = new double[]{0.0,1.0,0.0};
twoD[2] = new double[]{0.0,0.0,1.0};
```

Have to give a value for the number of rows...

...but not for the number of columns

Iterating over a 2D array

- To iterate over a two-dimensional array use a pair of nested for loops :

```
double[][] twoD = new double[2][3];
for (int i=0; i<twoD.length; i++) { // rows
    for (int j=0; j<twoD[i].length; j++) { // columns
        twoD[i][j] = TextIO.getDouble();
    }
}
```

twoD[i] is an array (whose length is 3)

- For a three-dimensional array you would use three nested for loops, and so on for higher dimensions

Arrays and Methods

- An array can be passed as a parameter to a method (e.g. the parameter to main())
- A method can update the elements of an array that it is passed as a parameter
  - We saw this in the lecture on developing an algorithm with the bubble sort example
- An array can also be the return-type for a method
- Some example methods follow on the next few slides

An example method

- Here is a method to find and return the smallest element in an array of **`double`**:

```
public static double findSmallest(double[] list) {
    double smallest = list[0];
    for (int i = 1; i < list.length; i++) {
          if (list[i] < smallest)
                smallest = list[i];
    }
    return smallest;
}
```


Comments on the method

- The variable `smallest` is initialised using the value of the first **`double`** in the array (`list[0]`)
  - this is better than initialising it to an arbitrary small value such as the value of the smallest possible **`double`**
- The for loop starts with index 1, as `list[0]` has already been looked at

Possible Exceptions

- The `findSmallest()` method will throw
  - a `NullPointerException` if the parameter, `list`, is **`null`**
  - an `ArrayIndexOutOfBoundsException` if the length of `list` is 0
- It would have been possible to include a check in the method for these scenarios and throw an `IllegalArgumentException` if either of them applies

Another example method

- This is the same logic but for an array of `String` (classes do not have a < operator):

```
public static String findSmallest(String[] list) {
    String smallest = list[0]; int i = 1;
    while (i < list.length && list[i] != null) {
        if (list[i].compareTo(smallest) < 0)
            smallest = list[i];
        i++;
    }
    return smallest;
}
```


Possible `null` elements

- As discussed last week, this method assumes that if there are any **`null`** values in the array they are at the end of the array – the while loop avoids a NullPointerException in the case there are such **`null`** values (if there is at least one String in the array).
- An alternative, which we used in the bubble sort example, is to also pass in an **`int`** variable that indicates how many elements are in use
  - This would work for arrays of primitive types too
- Next week we will meet the class, ArrayList, an alternative to arrays that avoids the issues arising from arrays having a fixed length

Two-dimensional arrays

- This method illustrates how to find the smallest value in a two-dimensional array:

```
public static int findSmallest(int[][] list) {
    int smallest = list[0][0];
    for (int i = 0; i < list.length; i++) {
        for (int j = 0; j < list[i].length; j++) {
            if (list[i][j] < smallest)
                smallest = list[i][j];
        }
    }
    return smallest;
}
```


A method that returns an array

- This method uses an array to return the smallest and the largest values in an array (smallest in [0], largest in [1]):

```
public static int[] findMinAndMax(int[] list) {
    int smallest = list[0]; int largest = list[0];
    for (int i = 1; i < list.length; i++) {
        if (list[i] < smallest)
            smallest = list[i];
        else if (list[i] > largest)
            largest = list[i];
    }
    return new int[]{smallest, largest};
}
```

Comments on findMinAndMax

- When a method has to return more than one value it is useful to bundle these into a data structure such as an array
- In this case, findMinAndMax() returns an array with two elements, the smallest and largest values in `list`
- The next slide gives a simple example application that calls the method

An example application

```
public static void main(String[] args) {
     int[] primesLessThan20 =
                        {11,7,3,17,5,13,2};
     int[] minAndMax =
          findMinAndMax(primesLessThan20);
     TextIO.putln("The smallest prime is " +
                         minAndMax[0]);
     TextIO.putln("The largest prime less " +
          " than 20 is " + minAndMax[1]);
}
```


Application output

- The application outputs:

```
The smallest prime is 2
The largest prime less than 20 is 17
```


This week and next week

- Have met multidimensional arrays
- Have started to discuss methods and arrays
- Part 3 of this lecture will be next week
- Continue to read and study chapter 7 of the book, on arrays