Introduction to Programming

10. Arrays – part 1

Arrays in Java

- Most programming languages, including Java, provide *array types*
- 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:

```
int[]          // arrays of ints
String[]       // arrays of Strings
```


Declaring an array variable

- To declare an array variable you use the array type name

```
int[] list;  /* can refer to arrays of
                int values /*

String[] names;  /* can refer to arrays
                    of String values */
```


Array objects

- When an array object is created, every element of the array is given a default value
  - 0 for primitive numeric types such as **int**, **char** and **double**
  - **false** for **boolean**
  - **null** for reference types such as String
- The simplest way to create an array object is to explicitly call the constructor for the array type
  - The syntax to call an array constructor is to use the word **new**, followed by the element-type of the array, and in the square brackets an int value that specifies how long the list of elements should be, e.g.:

```
new int[5]; // an array of 5 int elements
```

Creating an array object and initialising the array variable

- So, to create an array object, call the constructor for the array type and specify how many elements the array should contain

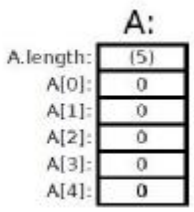
```
int[] list; // declare an array variable

list = new int[5]; /* create an array
object with five int elements and assign
its reference to the array variable */
```


More on array objects

- Once an array object is created, it cannot change in length
  - That is, it always contains the same number of elements
- An array object has a field called *length* which records how many elements the array object contains
  - This value reflects the length that was specified when the object was created

Array objects (Eck, page 115)

The statement A = new int[5]; creates an array that holds five elements of type int. A is a name for the whole array.



The array contains five elements, which are referred to as A[0], A[1], A[2], A[3], A[4]. Each element is a variable of type int. The array also contains A.length, whose value cannot be changed.

Summary on arrays so far

- The first array element has index 0
  - An array of 5 elements has elements [0] to [4]
- Each instance of an array type has a field called *length* = the number of elements
- Array instances (objects) are created using an array constructor, once created an array object cannot change in length
- An array variable is a reference variable that, once initialised, refers to an array object or has value **null**

Accessing an array element

- Can think of each array element as a *variable of the element type*
- Use the array variable name followed by the element's index number in square brackets to access the element

```
int[] list = new int[5];
list[0] = 1;  // regard list[0] as an int variable
```


Initialising an array object

- Can write a for loop that goes through each of the valid index numbers, from 0 to length-1, using the current index number to access and initialise each element in the array object in turn
- The for loop below sets the value of the five elements of `list` to 1, 2, 3, 4 and 5 respectively

```
int[] list = new int[5];
for (int i = 0; i < list.length; i++) {
    list[i] = i+1;  // use list[i] as a variable
}
```


Initialising an array object 2

- Here is an example of a for loop that accesses the array elements both to update them and to retrieve their values
  - This for loop sets the value of each element to twice that of the element before it, so starts at the second element (as there is no element before the first one)

```
int[] list = new int[5];
list[0] = 1;  // initialise the first array element
for (int i = 1; i < list.length; i++) {
    list[i] = list[i-1]*2;
}
```


Iterating over an array object

- The for loop below displays the value of each of the elements of `list` in turn

```
for (int i = 0; i < list.length; i++) {
    TextIO.putln(" list[" + i + "] = " + list[i]);
}
```

- For the list initialised on slide 11, this displays:

```
list[0] = 1
list[1] = 2
list[2] = 4
list[3] = 8
list[4] = 16
```

Notes on accessing elements

- So you can think of an array object as a collection of variables whose type is whatever the element type of the array is
- You access the element using its index
  - To update its value (as on slides 9, 10, 11)
  - To retrieve its value (as on slides 11, 12)

ArrayIndexOutOfBoundsException

- If the loop on slide 11 had started at 0 instead of 1, as below, it would have thrown an ArrayIndexOutOfBoundsException when trying to set list[0] to the value of list[0-1], as there is no element with index position, -1.

```
int[] list = new int[5];
list[0] = 1;  // initialise the first array element
for (int i = 0; i < list.length; i++) {
    list[i] = list[i-1]*2; // throws an exception!
}
```


Arrays of Reference Types

- When you create an array object, every element of the array is automatically initialised to its default value
  - This value is **null** if the element type is not a primitive type
- This gives rise to an added complication for arrays of reference types
  - An array of 5 int elements contains five ints, each of which is initialised automatically to zero
  - An array of 5 String elements does not contain any Strings when initialised to its default value, as **null** is not a String. Each String has to be created separately

An example

- An array of String

```
String[] pets;  /* can refer to arrays
                   of String values */
pets = new String[5]; /* creates an
      array of five String references,
      each of which is set to null */
```

  - At this point, we have not yet created any String objects

Example continued

- Can initialise each array element using its index value in a series of assignment statements

```
pets[0] = "Tabitha";
pets[1] = "Simon";
pets[2] = "Misty";
pets[3] = "Oscar";
pets[4] = "Max";
```


Example continued

- Or can use a loop to initialise the elements if the initial values are going to be read from the keyboard or a file:

```
for (int index=0; index < pets.length;
                                    index++) {
     TextIO.put("Enter pet's name: ");
     pets[index] = TextIO.getln();
}
```


Initialising an array

- So when creating an array there are usually two steps:
  - Create the array object and assign its reference to an array variable
  - Initialise each element of the array using its index to assign it a value, either in a series of assignment statements or in a loop

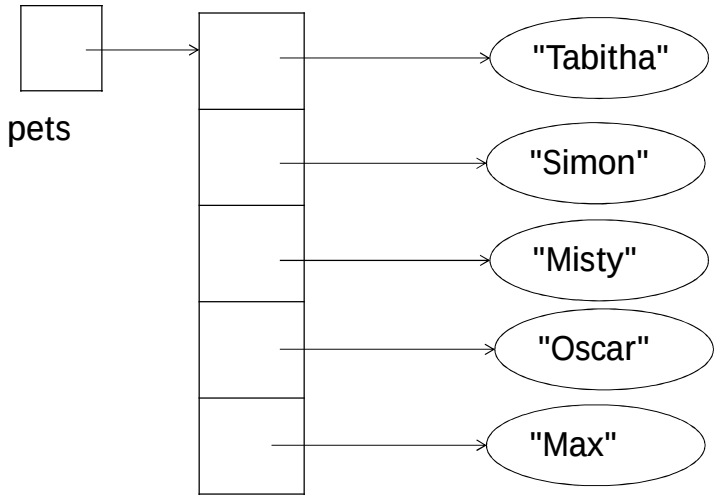
Combining array creation and initialisation

```
String[] pets = new String[]{"Tabitha",
       "Simon", "Misty", "Oscar", "Max"};
```

- Or just:

```
String[] pets = {"Tabitha", "Simon",
                 "Misty", "Oscar",
                 "Max"};
```

Result is (length field not shown)



Accessing the elements

- Once you have initialised the elements, you can access and manipulate them in the normal way

```
char initial = pets[0].charAt(0);
   // initial is assigned the value, 'T'

TextIO.put(pets[1].toUpperCase());
   // displays, SIMON
```

- These would throw `NullPointerException` if the elements had not been initialised

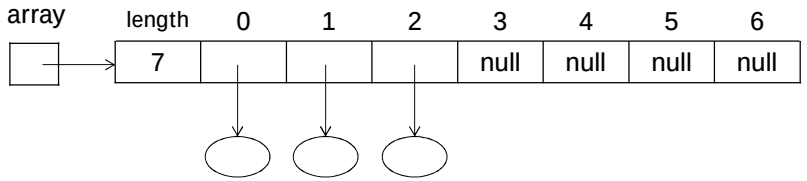
Arrays that are not “full”

- The size (length) of an array has to be stated when the array object is created
  - Array objects cannot change their length
- Often, the size given is an estimate of the maximum number of elements the collection will have and not all the array positions are necessarily going to be used all the time
  - That is, in an array of objects some elements may be `null`

Arrays that are not “full” continued

- In many applications, for an array that contains some `null` elements, it is convenient if all the `null` elements are located together at the end of the array
- If you maintain a variable, `count`, that counts how many positions in the array are in use, then
  - all the non-`null` elements are in consecutive locations from position 0 to position `count-1`
  - all the `null` elements are in consecutive locations from position `count` to position `length-1`.

An array with null elements



count

3

Elements are in locations 0 to `count`-1. The `null` values are in locations `count` to `array.length`-1.

An Example Application

- Consider a module that will take up to 20 students
- Can use an array to hold the class list
- Code on the next few slides...

The Student class

- Simplified Student record (see textbook page 192) – holds data about students on a course with three tests

```
public class Student {
    public String name; // Student's name.
    public double test1, test2, test3; /* Grades on
                                          three tests. */
   // Calculate the average test grade
    public double getAverage() {
        return (test1 + test2 + test3) / 3;
    }
} // end of class Student
```



```
public static void main(String[] args) {
   final int MAX_SIZE = 20;
   Student[] register = new Student[MAX_SIZE];
   int count = 0;
   TextIO.put("Enter student name " +
                  "or press <Enter> to quit: ");
   String name = TextIO.getln();
   while (name.length() > 0 && count < MAX_SIZE) {
      register[count] = new Student();
      register[count].name = name;
      count++;
      TextIO.put("Enter next student name " +
                  "or press <Enter> to quit: ");
      name = TextIO.getln();
   }
   ... // code here to use the array
}
```

Comments on the example

- The main() method on the previous slide creates an array that can store details of up to 20 students
- The user inputs student names at the keyboard, and terminates the list by pressing the return key when asked for the next student name
- The method declares a variable, count, which is used to count the students as their names are entered
- The loop that reads the names terminates when the String read has length zero (because the user has pressed the return key without entering a name) or when the module is full (count equals MAX_SIZE)

More Comments on example

- The loop that reads the names creates a `Student` object on each iteration and assigns the reference to it to the corresponding array element
- When all the student names have been read, the array contains references to `Student` objects in positions `0` to `count-1`, and the array elements from position count to `register.length-1` are all `null`

Iterating over arrays that are not full

- Maintaining a variable that counts the items in the array is useful in scenarios where you know that the array is not going to be full all the time
- The variable can be used in the condition in for loops, instead of `a.length`, that iterate over the array:

```
for (int i=0; i < count; i++) {
```

- If it is guaranteed that any `null` elements are at the end of the array, and you do not have a count of the elements that are not `null`, you can still avoid the `null` elements and iterate over the array using a while loop, where the condition is:

```
while (i < a.length && a[i] != null) {
```


Next Week

- Part 2 of this lecture will be next week
- Read chapter 7 of the book, on arrays, and study the examples