Arrays

Contents

Overview

Often we might want to store many values in a single variable.
For example, if we wanted to have a dataset for ages, we could make multiple variables, however, there is a much cleaner way of doing this; arrays.
Arrays are designed to hold data collections, they can be single or multi-dimensional, and work with indexes and elements.
Each value in an array is referred to as an element and that element is indexed.
The index is used as a point of reference for an element, the index also starts at 0.

Tutorial

Single Dimensional Array

There are three different ways to create a single dimensional array in Java.

```
public class Arrays {

    public int[] ageArray;
}
```

The above example shows the first method of creating a single-dimensional array, and it is creating an empty array that we can insert data into in the future.
The square brackets in the variable declaration are what tell Java that we want this variable to be an array.

```
public class Arrays {

    public int[] ageArray = {25, 43, 23, 46, 30};
}
```

The above example shows the second method of creating an array, which is creating the variable as an array and then initialising it with a dataset.

```
public class Arrays {

    public int[] ageArray = new int[5];
}
```

The above example shows the third method of creating an array, which is creating an empty array whilst specifying the length of the array.
Specifying the length of the array as 5 means that up to 5 values can be stored in that array but no more than 5.

Multi Dimensional Arrays

Multi dimensional arrays store data collections in a more sophisticated way than single dimensional arrays.
Each of the indexes within a multi dimensional array stores an array within them.
Much like single dimensional arrays, there are three different ways to create a multi dimensional array.

```
public class Arrays {

    public int[][] ageArray;
}
```

The above example shows the first method of creating a multi dimensional array and is creating an empty array for us that we can store data into in the future.
Java knows that we want this to be a multi dimensional array by the two sets of square brackets in the variable declaration.

```
public class Arrays {

    public int[][] ageArray = {{25, 30, 21},{56, 67, 39},{8}};
}
```

The above example shows the second method of creating a multi dimensional array, and is creating the array and initialising it with a dataset.
Each set of { } brackets is its own array that is to be stored within the multi dimensional array.
So 3 arrays are being stored in the array with the reference variable "ageArray".

```
public class Arrays {

    public int[][] ageArray = new int[3][2];
}
```

The above example shows the third method of creating a multi dimensional array, and is creating an empty array but specifying the length of the array.
The above definition will create an array that can store 3 arrays each with 2 values inside.

Assigning Array Values

We assign values in an array through the use of the index, or indices in the case of multi dimensional arrays.
Specifying the index assigns that value to the element, this is the same for both single and multi dimensional arrays.

```
public class Arrays {

    public static int[] ageArraySD = new int[5];

    public static void main(Sting[] args) {
        ageArraySD[2] = 33;
    }
}
```

The above example will input the value 33 into the array named ageArraySD at index 2, which will be the third value in the array.

```java
public class Arrays {

    public static int[][] ageArrayMD = new int[3][2];

    public static void main(String[] args) {
        ageArrayMD[2][1] = 59;
    }
}
```

The above example will input the value 59 into the array named ageArrayMD at index 2, which is an array so we give it a second index to tell Java which index within the inner array to store the value to.
So the value 59 will be stored in the array at index 2, which will be the third array in the multi dimensional array, and then index 1 in the inner array, which will be the second entry.

Looping Through an Array

For Each Loops

For each loops iterate through each element in an array or collection, this is a much tidier version of what regular for loops can achieve, and both have their benefits.

```java
public class Arrays {

    public static int num[] = {1,2,3,4,5,6,7,8,9,0};

    public static void main(String[] args) {
        for(int i : num) {
            System.out.println("Number: " + i);
        }
    }
}
```

The above example is declaring and initialising an array with the name num and assigning it a collection of values.
Then in the main method, we are looping through and printing out every value in the array.
The for loop is essentially saying "for every integer i in num do this", the do this is just whatever code is inside our for loops code block, so in this case, the print statement.

For Versus For Each

Both for loops and for each loops do the same thing, they iterate over a collection of data, however for loops have a count of which iteration we are currently running whereas for each loops do not have a count.
Generally, we would use a for loop when having the count of the current iteration is important to the code we want to run, and we would use a for each loop when we are doing the same thing to everything in the collection of data.

For Loop

```java
public class Arrays {

    public static int votes[] = {1,1,2,1,3}

    public static void process(int vote) {
        System.out.println(vote);
    }

    public static void main(String[] args) {
        for(int i = 0; i < votes.length; i++) {
            process(votes[i])
        }
    }
}
```

For Each Loop

```java
public class Arrays {

    public static int votes[] = {1, 1, 2, 1, 3};

    public static void process(int vote) {
        System.out.println(vote);
    }

    public static void main(String[] args) {
        for(int vote : votes) {
            process(vote);
        }
    }
}
```

Both of the above examples do the same thing, they both loop through the array and pass the value in the array at the current iteration to a method called process; which simply prints the vote for us.
Where they are different is that in the for loop, we need to give the array name and the index of the current iteration to the method call; whereas the for each loop simply passes the value at the current iteration, which is stored as a vote.

Looping Through a Multi Dimensional Array

We can also loop through a multi dimensional array using nested for loops, or nested for each loops.

For Loop

```java
public class Arrays {

    public static int twoDArray[][] = {{0, 1, 2}, {1, 2, 3}, {2, 3, 4}};

    public static void main(String[] args) {
        for(int i = 0; i < twoDArray.length; i++) {
            for(int j = 0; j < twoDArray[i].length; j++) {
                System.out.print(twoDArray[i][j]);
            }
            System.out.println();
        }
    }
}
```

In the above example, we are declaring and initialising a multi dimensional array, with 3 arrays inside, each with 3 values.
In the main method, we are then looping over the array of arrays with the for loop using i as its index.
During each iteration of the initial for loop, we loop over the array of values within the outer array and print each of the values to the console.
Once the three values inside the inner array have been printed, we print a new line so that the next array of values can be printed on its own line.

For Each Loop

```
public class Arrays {

    public static int twoDArray[][] = {{0, 1, 2}, {1, 2, 3}, {2, 3, 4}};

    public static void main(String[] args) {
        for(int[] a : twoDArray) {
            for(int b : a) {
                System.out.print(b);
            }
            System.out.println();
        }
    }
}
```

In the above example, we are declaring and initialising a multi dimensional array, with 3 arrays inside, each with 3 values.
In the main method, we are then looping over the array of arrays, saying for every array in twoDArray do the following.
In the nested for each loop, we are telling Java to do the following for every value in the array retrieved by the previous for each loop.
While looping through the nested for each loop we are simply printing the value to the console.
Once the nested for each loop has finished looping, we print a new line so that the next array of values within the array of arrays can have its values printed on their own line.

Using Both a For Loop and a For Each Loop

We can also use both a for loop and a for each loop when we are nesting loops.
So if we have a for each loop, we can still use a regular for loop as a nested loop.

```
public class Arrays {

    public static int nums[] = {1, 2, 3, 4, 5};

    public static void main(String[] args) {
        for(int i : nums) {
            for(int j = 0; j < nums.length; j++) {
                if(nums[j] == i) {
                    System.out.println("i:" + i + " j:" + nums[j] + " i and j 
match");
                }
            }
        }
    }

}
```

In the above example, we are declaring and initialising an array with values.
Then in the main method we are using a for each loop to loop over every item in the nums array.
Then we have a nested for loop which also loops over every item in the array.
If the value at index j is the same as the value in i, which is the value for the current iteration of the for each loop, then we print the values and print that they match.

Array List

In Java, we can import different utilities such as an ArrayList.
The ArrayList is a class that extends Collection and provides a lot of controls that regular arrays provide.
The ArrayList class serves the same purpose as an array but it is resizable, whereas the arrays we have been using are not; this can be very beneficial when we don't know how big we need our array to be as we can resize it after creation.
When using an ArrayList we can access elements stored within it by their integer index or by searching for the element in the ArrayList.
ArrayLists also come with a special iterator called ListIterator.

```java
import java.util.ArrayList;

public class Arrays {

    ArrayList<Object> objects = new ArrayList<>();
}
```

In the above example, we are importing the ArrayList class from java util; imports generally go above the class definition (at the top of the class).
The inside the class, we are declaring a variable with the type ArrayList holding the type Object and instantiating it as a new ArrayList, which essentially means we are creating a new Object.
Unlike the primitive arrays we were looking at earlier, ArrayLists can be used to store a collection of objects.

Exercises

1. Create an array that will hold 10 integer values, populate the array with values, then call and output each element.
2. Create a for loop that populates an integer array with values, outputting them at each iteration.
   Then create another loop that iterates through the array, changing the values at each point to equal itself times 10, outputting them at each iteration.

Numbers

1. Create a method that takes a number 10-99, and adds the two digits together for example 74 = 7 + 4 = 11.
2. Create a second method that when given the number 1-99 returns a `String` representation of this number, for example 1 = `one`, 11 = `eleven`, 21 = `twenty-one`.
3. Expand on the method you wrote in step 2 to allow the input 1-999.
4. Expand the method you wrote in step 2 to allow the input 1-9999.
5. Use a `for()`-loop to print the numbers 1-100 in words.
   For example; 1 = `one`, 100 = `one hundred`.