Hyperiondev

TASK

Java - Data Structures

Visit our website

Introduction

WELCOME TO THE JAVA - DATA STRUCTURES TASK!

This task introduces you to a fundamental data structure in Java: the array. An array is used to store a fixed-size collection of elements of the same type. This task focuses on both single-dimensional and multidimensional arrays and explains how arrays are declared, created, initialised and processed.

WHAT ARE ARRAYS?

The array is a data structure, provided by Java, which stores a **fixed number of elements of the same type**. An array is used to store a collection of data. However, it is more useful to think of an array as a collection of variables of the same type.

DECLARING ARRAYS

To use an array in a program, a variable to reference the array must be declared, and the array's element type must be specified. The array can have any element type. All elements that are in the array will have the same data type. To declare a variable, use the following syntax:

```
elementType[] arrayVariable;
```

The following example declares a variable `numArray` that references an array that stores elements of the type `double`.

```
double[] numArray;
```

CREATING ARRAYS

After declaring an array variable you can create an array using the new operator. The syntax is as follows:

```
elementType[] arrayVariable = new elementType[arraySize];
```

The following example declares an array variable called **numArray**, creates an array of 5 double elements and assigns the reference to the array to **numArray**.

```
double[] numArray = new double[5];
```

The array size must be given when space for an array is allocated. The size cannot be changed after an array is created. To find the size of an array, use **arrayVariable.length**.

Numeric data types are assigned the default value 0, **char** types are assigned the value **\u0000** and boolean types are assigned the value **false** when an array is created.

ARRAY INDICES

To access array elements you use an index. Array indices range from 0 to **arraryVariable.length - 1**. For example, the array created above (**numArray**), holds five elements with indices from 0 to 4. You can represent each element in the array using the following syntax:

```
arrayVariable[index];
```

For example, **numArray[1]** represents the second element in the array **numArray**.

INITIALISING AN ARRAY

To assign values to the elements we will use the syntax:

```
arrayVariable[index] = value;
```

The example below initialises the array referenced by the variable **numArray**:

```
numArray[0] = 23.6;
numArray[1] = 45.12;
numArray[2] = 8.4;
numArray[3] = 77.7;
numArray[4] = 1.34;
```

You can use shorthand notation in Java, known as the array initialiser, which combines the declaration, creation and initialiSation of an array in one statement.

The syntax for an array initialiser is as follows:

```
elementType[] arrayVariable = {value1, value2, ..., valueN};
```

For example, the following array initialiser statement is equivalent to the example above in which each element was assigned a value individually:

```
double[] numArray = {23.6, 45.12, 8.4, 7.77, 1.34};
```

PROCESSING ARRAYS

The easiest way to process the elements of an array is by using a *for loop*. A *for loop* is the natural choice when processing arrays because all of the elements in an array have the same data type and are processed in the same way. Since the size of the array is known, it makes sense to use a *for loop* rather than another repetition structure. Suppose you have created the following array:

```
double[] myArray = new double[10];
```

You can use a *for loop* to initialise the array with values entered by a user as follows:

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Please enter " + myArray.length + " values");
for (int i = 0; i < myArray.length; i++) {
    myArray[i] = input.nextDouble();
}
```

You can also display an array by printing each element in the array by using the following *for loop*:

```
for (int i = 0; i < myArray.length; i++) {
    System.out.print(myArray[i] + " ");
}
```

As you can see, it would be quite difficult to process each of the array's elements individually, especially if the array is large. *For loops* are a lifesaver when it comes to performing repetitive operations on elements in an array.

TWO-DIMENSIONAL ARRAYS

In the previous sections, we discussed simple single-dimensional arrays. However, arrays are not limited to one dimension. You can use a two-dimensional array to store data in a table or matrix.

DECLARING AND CREATING TWO-DIMENSIONAL ARRAYS

The syntax for creating a two-dimensional array is as follows:

```
elementType[][] arrayVariable;
```

The following example demonstrates how to declare and create a two-dimensional array of 5-by-5 int values.

```
int[][] table;
table = new int[5][5];
```

Each number between the square brackets represents the number of rows and columns in a table. The first index represents the rows and the second represents the columns. Each index begins with a 0, as with a one-dimensional array.

ASSIGNING VALUES TO ELEMENTS IN A TWO-DIMENSIONAL ARRAY

Like a single-dimensional array, use the array indices to access elements in a two-dimensional array. However, unlike a one-dimensional array, a two-dimensional array contains two sets of indices.

The example below shows how to assign the number 4 to the element in the second row and first column:

```
table[1][0] = 4;
```

The array initialiser can also be used to declare, create and initialise a two-dimensional array.

The following example creates an array and initialises it with values.

```
int[][] array = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12} };
```

This is equivalent to the following:

```
int[][] array = new int[4][3];
array[0][0] = 1;    array[0][1] = 2;    array[0][2] = 3;
array[1][0] = 4;    array[1][1] = 5;    array[1][2] = 6;
array[2][0] = 7;    array[2][1] = 8;    array[2][2] = 9;
array[3][0] = 10;   array[3][1] = 11;   array[3][2] = 12;
```

Take note:

Ragged Arrays

In a two-dimensional array, each row is itself an array. Therefore, the rows can have different lengths. Arrays with rows of varying lengths are known as ragged arrays.

Here is an example of a ragged array:

```
int[][] triArray = { { 1, 2, 3, 4},
                     {5, 6, 7},
                       {8, 9},
                       {10} };
```

If you know the sizes of a ragged array, but do not know the values of the elements you can still create one by using the following syntax:

```
int[][] triArray = new int[4][];
triArray[0] = new int[4];
triArray[1] = new int[3];
triArray[2] = new int[2];
triArray[3] = new int[1];
```

Note that when using new int [4] [ ] to create a new array, the first index (number of rows) must always be specified

ARRAYLISTS

You will be learning about ArrayLists in more detail in a later task, but for now, ArrayLists are arrays in which its length is malleable — you can insert and remove elements as you like. You can create an initialise an ArrayList as follows:

```
ArrayList<String> drivers = new ArrayList<>();
drivers.add("Larry");
drivers.add(0, "Jill");       // adds Jill at index 0
drivers.remove(1)       // removes item at index 1
```

Common ArrayList methods include:

- *add(index, element)* - adds the given element at the given index
- *contains(element* - returns a boolean to show if the given element is in the ArrayList
- *get(index)* - returns the element at the given index
- *indexOf(element)* - returns the index of the given element
- *isEmpty()* - returns a boolean to show if the ArrayList has no elements
- *remove(index)* - removes the element at the given index
- *size()* - returns the number of elements in the ArrayList

Compulsory Task 1

Follow these steps:

- Create a new file called **IQ.java**
- Create an array called '*testResults*' that stores a list of results that 5 people have scored on an IQ test.
- Write the code that will calculate the average IQ of the five people based on the results in my '*testResults*'.
- Based on the average IQ, decide which description should be assigned. Use the table below:

IQ score	Description
130+	Very superior
120-129	Superior
110-119	High average
90-109	Average
80-89	Low average
70-79	Borderline
69 and below	Extremely low

Compulsory Task 2

Follow these steps:

- Create a new file called **multipleElements.java**
- Write a program that creates a multi-dimensional array with the first 2 multiples of the numbers in the first row.
- Firstly, ask the user to enter four numbers. These will inhabit the first row of your array.
- The next two rows will be the first two multiples of the numbers above them. For example, if the user inputs 1, 2, 3 and 4, the 3-by-4 array will look like this:

  1 2 3 4

  2 4 6 8

  3 6 9 12
- Print out the array.

Compulsory Task 3

Follow these steps:

- Create a new file called **longWords.java**
- Write a program that determines how many words greater than 4 letters long have been entered and calculate the average length of all the words entered.
- Firstly, ask the user to enter any number of words and store them in an array. The user should enter 0 to indicate the end of their input.

- The program should then determine the number of words longer than 4 letters entered by the user, and print out the result.
- The average length of the words given should then be calculated and displayed.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.