Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as `number0`, `number1`, ..., and `number99`, you declare one array variable such as `numbers[0]`, `numbers[1]`, and ..., `numbers[99]` to represent individual variables.

### Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```java
dataType[] arrayRefVar;  // preferred way
```

or

```java
dataType arrayRefVar[];  // works, but not preferred way
```

**Note**

The style `dataType[] arrayRefVar` is preferred. The style `dataType arrayRefVar[]` comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

### Example

```java
double[] myList;  // preferred way
```

or

```java
double myList[];  // works but not preferred way
```

### Creating Arrays

You can create an array by using the `new` operator with the following syntax:

```java
arrayRefVar = new dataType[arraySize];
```
The above statement does two things:

- It creates an array using `new dataType[arraySize];`
- It assigns the reference of the newly created array to the variable `arrayRefVar`.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```java
dataType[] arrayRefVar = new dataType[arraySize];
```

Alternatively you can create arrays as follows:

```java
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

## ACCESSING ARRAY ELEMENTS

The array elements are accessed through the index.

Array indices are 0-based; that is, they start from 0 to `arrayRefVar.length-1`.

## EXAMPLE

Following statement declares an array variable, `myList`, creates an array of 10 elements of `double` type and assigns its reference to `myList`:

```java
double[] myList = new double[10];
```

Following picture represents array `myList`. Here, `myList` holds ten double values and the indices are from 0 to 9.
When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

**EXAMPLE**

Here is a complete example of showing how to create, initialize and process arrays:

```java
public class TestArray {
    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }

        // Summing all elements
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
            total += myList[i];
        }
        System.out.println("Total is " + total);

        // Finding the largest element
        double max = myList[0];
        for (int i = 1; i < myList.length; i++) {
            if (myList[i] > max) max = myList[i];
        }
        System.out.println("Max is " + max);
    }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

**THE FOREACH LOOPS**

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

**EXAMPLE**

The following code displays all the elements in the array myList:
public class TestArray {

    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (double element: myList) {
            System.out.println(element);
        }
    }
}

This would produce the following result:

    1.9
    2.9
    3.4
    3.5

### PASSING ARRAYS TO METHODS

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array:

```java
public static void printArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2:

```java
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

### RETURNING AN ARRAY FROM A METHOD

A method may also return an array. For example, the method shown below returns an array that is the reversal of another array:

```java
public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
    }
    return result;
}
```
THE ARRAYS CLASS

The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements. These methods are overloaded for all primitive types.

<table>
<thead>
<tr>
<th>Methods with Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  public static int binarySearch(Object[] a, Object key)</td>
</tr>
<tr>
<td>Searches the specified array of Object (Byte, Int, double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, -(insertion point + 1).</td>
</tr>
<tr>
<td>2  public static boolean equals(long[] a, long[] a2)</td>
</tr>
<tr>
<td>Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other primitive data types (Byte, short, Int, etc.)</td>
</tr>
<tr>
<td>3  public static void fill(int[] a, int val)</td>
</tr>
<tr>
<td>Assigns the specified int value to each element of the specified array of ints. Same method could be used by all other primitive data types (Byte, short, Int etc.)</td>
</tr>
<tr>
<td>4  public static void sort(Object[] a)</td>
</tr>
<tr>
<td>Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. Same method could be used by all other primitive data types (Byte, short, Int etc.)</td>
</tr>
</tbody>
</table>

EXAMPLES

As arrays are extremely useful for JAVA programming, let us study several simple examples.

EVEN ODD NUMBER EXAMPLE

/* This Java Even Odd Number Example shows how to check if the given number is even or odd. */

class FindEvenOrOddNumber {
  public static void main(String[] args) {
    //create an array of 10 numbers
    int[] numbers = new int[]{1,2,3,4,5,6,7,8,9,10};

    for(int i=0; i < numbers.length; i++){
      /* Use modulus operator to check if the number is even or odd: If we divide any number by 2 and reminder is 0 then the number is even, otherwise it is odd. */
      if(numbers[i]%2 == 0)
        System.out.println(numbers[i] + " is even number.");
      else
System.out.println(numbers[i] + " is odd number.");
}
}

Output of the program would be
1 is odd number.
2 is even number.
3 is odd number.
4 is even number.
5 is odd number.
6 is even number.
7 is odd number.
8 is even number.
9 is odd number.
10 is even number.

FIND LARGEST AND SMALLEST NUMBER IN AN ARRAY EXAMPLE
/* This Java Example shows how to find largest and smallest number in an array. */
public class FindLargestSmallestNumber {
    public static void main(String[] args) {
        //array of 10 numbers
        int numbers[] = new int[]{32,43,53,54,32,65,63,98,43,23};

        //assign first element of an array to largest and smallest
        int smallest = numbers[0];
        int largest = numbers[0];

        for(int i=1; i< numbers.length; i++) {
            if(numbers[i] > largest)
                largest = numbers[i];
            else if (numbers[i] < smallest)
                smallest = numbers[i];
        }
        System.out.println("Largest Number is : " + largest);
        System.out.println("Smallest Number is : " + smallest);
    }
}

Output of this program would be
Largest Number is : 98
Smallest Number is : 23

I/O

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class JavaFactorialUsingRecursion {

    public static void main(String args[]) throws NumberFormatException, IOException{
        System.out.println("Enter the number: ");

        //get input from the user
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        int a = Integer.parseInt(br.readLine());

        //call the recursive function to generate factorial
        int result= fact(a);

        System.out.println("Factorial of the number is: " + result);
    }

    static int fact(int b)
    {
        if(b <= 1) //if the number is 1 then return 1
            return 1;
        else //else call same function with the value-1
            return b * fact(b-1);
    }
}

Output of this Java example would be

Enter the number:
5
Factorial of the number is: 120