Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java: Array - Introduction/5aa9f73a746fb501b1efcaba

TUTORIAL

Java: Array - Introduction

Topics

Array is one of the commonly used data structures. Array is a data structure consists of related data items of the same type. Arrays are “static” entities, in that they remain the same size once they are created, although an array reference may be reassigned to a new array of a different size.

Arrays

An array is a group of contiguous memory locations that all have the same name and the same type. To refer to a particular location or element in the array, we specify the name of the array and the position number (or index or subscript) of the particular element in the array.

Figure below shows an integer array called *arr*. This array contains 8 elements. A program refers to any one of these elements by giving the name of the array followed by the position number of the particular element in square brackets ([]). The first element in every array has position number zero (sometimes called the zeroth element). Thus, the first element of array *arr* is *arr*[0], the second element of array *arr* is *arr*[1], the seventh element of array *arr* is *arr*[6], and, in general, the ith element of array *arr* is *arr*[i - 1]. Array names follow the same conventions as other variable names.

Formally, the position number in square brackets is called a subscript (or an index). A subscript must be an integer or an integer expression. If a program uses an expression as a subscript, the program evaluates the expression to determine the subscript. For example, if we assume that variable a is 2 and that variable b is 3, then the statement

```
arr[ a + b ] += 2;
```

adds 2 to array element *arr*[5]. Note that a subscripted array name is an lvalue — it can be used on the left side of an assignment to place a new value into an array element.

```
-57      36     3       7       2      -67       5      3
arr[0] arr[1] arr[2]  arr[3]  arr[4]  arr[5]   arr[6]  arr[7]
```

Figure: Example Array arr

Let us examine array *arr* in Figure above more closely. The name of the array is *arr*. Every array in Java knows its own length and maintains this information in a variable called length. The expression *arr*.length accesses array *arr*'s length variable to determine the length of the array. The array's 8 elements are referred to as *arr*[0], *arr*[1], *arr*[2], ..., *arr*[7]. The value of *arr*[0] is -57, the value of *arr*[1] is 36, the value of *arr*[2] is 3, the value of *arr*[6] is 5 and the value of *arr*[7] is 3. To calculate the sum of the values contained in the first three elements of array c and store the result in variable sum, we would write

```
sum = arr[ 0 ] + arr[ 1 ] + arr[ 2 ];
```

To divide the value of the seventh element of array *arr* by 2 and assign the result to the variable x, we would write

```
x = arr[ 6 ] / 2;
```

The brackets used to enclose the subscript of an array are one of Java's many operators. Brackets are in the highest level of

precedence in Java.

Declaring and Allocating Arrays

Arrays are objects that occupy space in memory. All objects in Java (including arrays) must be allocated dynamically with operator new. For an array, the programmer specifies the type of the array elements and the number of elements as part of the new operation. To allocate 12 elements for integer array c, use the declaration

```
int c[] = new int[ 12 ];
```

The preceding declaration also can be performed in two steps as follows:

```
int c[]; // declares the array
c = new int[ 12 ]; // allocates the array
```

When allocating an array, each element of the array receives a default value–zero for the numeric primitive-data-type elements, false for boolean elements or null for references (any nonprimitive type). A program can allocate memory for several arrays with a single declaration. The following declaration reserves 100 elements for String array b and 27 elements for String array x:

```
String b[] = new String[ 100 ], x[] = new String[ 27 ];
```

When declaring an array, the type of the array and the square brackets can be combined at the beginning of the declaration to indicate that all identifiers in the declaration represent arrays, as in

```
double[] array1, array2;
```

which declares both array1 and array2 as arrays of double values. As shown previously, the declaration and initialization of the array can

be combined in the declaration. The following declaration reserves 10 elements for array1 and 20 elements for array2:

```
double[] array1 = new double[ 10 ], array2 = new double[ 20 ];
```

A program can declare arrays of any data type. It is important to remember that every element of a primitive data type array contains one value of the declared data type. For example, every element of an int array contains an int value. However, in an array of a non-primitive type, every element of the array is a reference to an object of the array's declared data type. For example, every element of a String array is a reference to a String. In arrays that store references, the references have the value null by default.

The program below uses operator new to allocate dynamically an array of 10 int elements, which are initially zero (the default value in an array of type int). Line 4 declares array—a reference capable of referring to an array of int elements. Line 5 allocates the 10 elements of the array with new and initializes the reference. The program builds its output in the String called output that will be displayed on screen. Line 6 appends to output the headings for the columns displayed by the program. The columns represent the subscript for each array element and the value of each array element, respectively. Lines 9–10 use a for structure to append the subscript number (represented by counter) and value of each array element to output. Note the use of zero-based counting (remember, subscripts start at 0), so that the loop can access every array element. Also, note the expression array.length in the for structure condition to determine the length of the array. In this example, the length of the array is 10, so the loop continues executing as long as the value of control variable counter is less than 10. For a 10-element array, the subscript values are 0 through 9, so using the less than operator < guarantees that the loop does not attempt to access an element beyond the end of the array.

```
1 public class Main                                    Java
2 {
3   // main method begins execution of Java application
```

```
  public static void main( String args[])
  {
    int array[];
  // declare reference to an array
    array = new int[ 10 ];
       // dynamically allocate array
    String output = "Subscript \t Value\n";
    System.out.println("Array length is : " +
array.length);
    // append each array element's value to String
output
    for(int counter = 0; counter < array.length;
counter++)
    {
      output += counter + "\t-\t\t" + array[counter] +
"\n";
    }
    System.out.println(output);
  }
}
```

A program can allocate and initialize the elements of an array in the array declaration by following the declaration with an equal sign and a comma-separated initializer list enclosed in braces ({ and }). In this case, the array size is determined by the number of elements in the initializer list. For example, the declaration

```
int n[] = { 10, 20, 30, 40, 50 };
```

creates a five-element array with subscripts of 0, 1, 2, 3 and 4. Note that the preceding declaration does not require the new operator to create the array object. When the compiler encounters an array declaration that includes an initializer list, the compiler counts the number of initializers in the list and sets up a new operation to allocate the appropriate number of array elements. The above program can be written as below:

Java

```
public class Main
{
  // main method begins execution of Java application
  public static void main( String args[])
  {
```

```
    int array[] = {11,22,33,44,55,66,77,88,99,111};
           // declare reference to an array
    String output = "Subscript \t Value\n";
    System.out.println("Array length is : " +
array.length);
    // append each array element's value to String
output
    for(int counter = 0; counter < array.length;
counter++)
    {
      output += counter + "\t-\t\t" + array[ counter ]
+ "\n";
    }
    System.out.println(output);
  }
}
```

Sum of Elements in Array

Often, the elements of an array represent a series of values to use in a calculation. For example, if the elements of an array represent the grades for an exam in a class, the professor may wish to total the elements of an array, then calculate the class average for the exam. The code below sums the values contained in the 10-element integer array. The program declares, allocates and initializes the array and use for loop structure to perform the calculations.

Java

```
public class Main
{
  // main method begins execution of Java application
  public static void main( String args[])
  {
    int array[] = {11,22,33,44,55,66,77,88,99,111};
   // declare reference to an array
    int total = 0;
    for(int counter = 0; counter < array.length;
counter++)
      total += array[ counter ];
    System.out.println("Sum of array elements is : " +
total);
  }
}
```