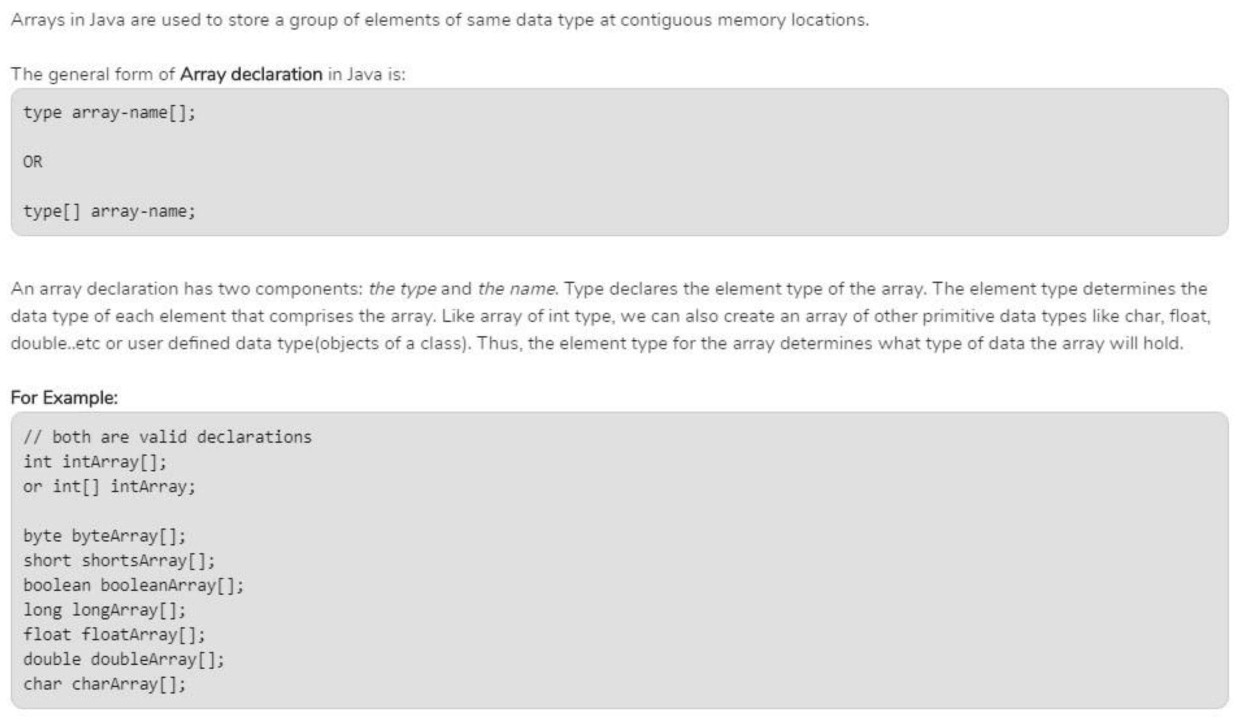
Implementing Arrays in Java



Arrays in Java are used to store a group of elements of same data type at contiguous memory locations.

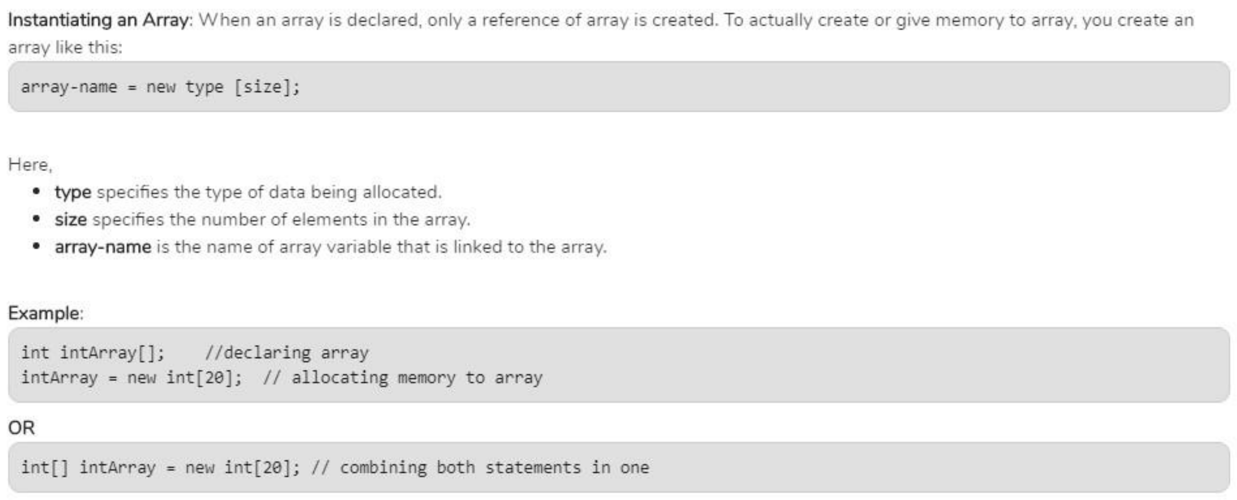
The general form of **Array declaration** in Java is:

```
type array-name[];

OR

type[] array-name;
```

An array declaration has two components: *the type* and *the name*. Type declares the element type of the array. The element type determines the data type of each element that comprises the array. Like array of int type, we can also create an array of other primitive data types like char, float, double..etc or user defined data type(objects of a class). Thus, the element type for the array determines what type of data the array will hold.

For Example:

```
// both are valid declarations
int intArray[];
or int[] intArray;

byte byteArray[];
short shortsArray[];
boolean booleanArray[];
long longArray[];
float floatArray[];
double doubleArray[];
char charArray[];
```



Instantiating an Array: When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:

```
array-name = new type [size];
```

Here,

- **type** specifies the type of data being allocated.
- **size** specifies the number of elements in the array.
- **array-name** is the name of array variable that is linked to the array.

Example:

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

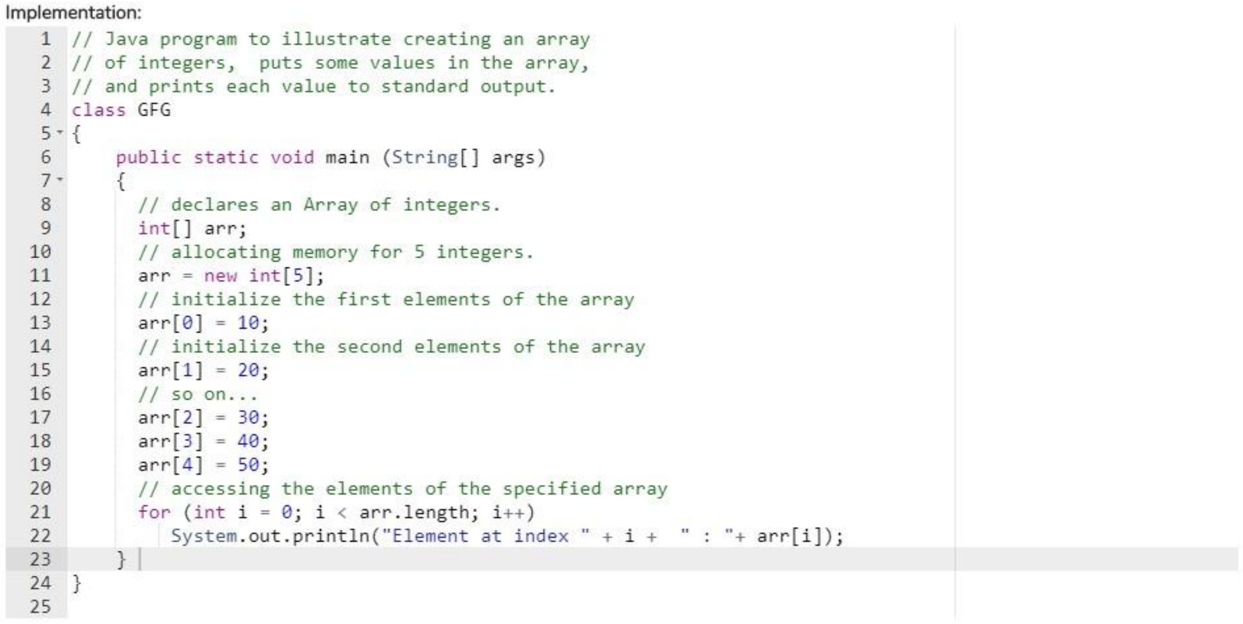
OR

```
int[] intArray = new int[20]; // combining both statements in one
```

Accessing Java Array Elements using for Loop: Each element in the array is accessed by its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop as shown below.

```
// Accessing the elements of the specified array

for (int i = 0; i < arr.length; i++)

  System.out.println("Element at index " + i +

                              " : "+ arr[i]);



Here, arr.length gives the number of elements

present in the array named arr.
```

Implementation:



```java
// Java program to illustrate creating an array
// of integers,  puts some values in the array,
// and prints each value to standard output.
class GFG
{
    public static void main (String[] args)
    {
      // declares an Array of integers.
      int[] arr;
      // allocating memory for 5 integers.
      arr = new int[5];
      // initialize the first elements of the array
      arr[0] = 10;
      // initialize the second elements of the array
      arr[1] = 20;
      // so on...
      arr[2] = 30;
      arr[3] = 40;
      arr[4] = 50;
      // accessing the elements of the specified array
      for (int i = 0; i < arr.length; i++)
         System.out.println("Element at index " + i +  " : "+ arr[i]);
    }
}

```

Output:

```
Element at index 0 : 10
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50
```

Java also provides some inbuilt classes which can be used for implementing arrays or sequential lists. Let's look at some of these in details.

ArrayList in Java

ArrayList is a part of collection framework and is present in java.util package. It provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of array manipulation is needed.

Constructors in Java ArrayList:

- **ArrayList()**: This constructor is used to build an empty array list.
- **ArrayList(Collection c)**: This constructor is used to build an array list initialized with the elements from collection c.
- **ArrayList(int capacity)**: This constructor is used to build an array list with initial capacity being specified.

Creating a generic integer ArrayList:

```
ArrayList arrli = new ArrayList();
```

Implementation:

```java
// Java program to demonstrate working of
// ArrayList in Java
import java.io.*;
import java.util.*;
class arrayli
{
    public static void main(String[] args)
                        throws IOException
    {
        // size of ArrayList
        int n = 5;
        // declaring ArrayList with initial size n
        ArrayList<Integer> arrli = new ArrayList<Integer>(n);
        // Appending the new element at the end of the list
        for (int i=1; i<=n; i++)
            arrli.add(i);
        // Printing elements
        System.out.println(arrli);
        // Remove element at index 3
        arrli.remove(3);
        // Displaying ArrayList after deletion
        System.out.println(arrli);
        // Printing elements one by one
        for (int i=0; i<arrli.size(); i++)
            System.out.print(arrli.get(i)+" ");
    }
}

```

Output:

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Click Here to learn ArrayList in Java in details!

Vector Class in Java

The Vector class implements a growable array of objects. Vectors basically falls in legacy classes but now it is fully compatible with collections.

- Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.
- They are very similar to ArrayList but Vector is synchronised and have some legacy method which collection framework does not contain.
- It extends AbstractList and implements List interfaces.

Constructor:

- **Vector()**: Creates a default vector of initial capacity is 10.
- **Vector(int size)**: Creates a vector whose initial capacity is specified by size.
- **Vector(int size, int incr)**: Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time that a vector is resized upward.
- **Vector(Collection c)**: Creates a vector that contains the elements of collection c.

Implementation:

```java

// Java code to illustrate Vector

import java.util.*;
class Vector_demo {
    public static void main(String[] arg)
    {
        // Create a vector
        Vector<Integer> v = new Vector<Integer>();

        // Insert elements in the Vector
        v.add(1);
        v.add(2);
        v.add(3);
        v.add(4);
        v.add(3);

        // Print the Vector
        System.out.println("Vector is " + v);
    }
}

```

Output:

```
Vector is [1, 2, 3, 4, 3]
```