Java Arrays

An array is a collection of similar types of data.

For example, if we want to store the names of 100 people then we can create an array of the string type that can store 100 names.

```
String[] array = new String[100];
```

Here, the above array cannot store more than 100 names. The number of values in a Java array is always fixed.

How to declare an array in Java?

In Java, here is how we can declare an array.

```
dataType[] arrayName;
```

dataType - it can be primitive data types like int, char, double, byte, etc. or Java objects
arrayName - it is an identifier
For example,

```
double[] data;
```

Here, data is an array that can hold values of type double.

To define the number of elements that an array can hold, we have to allocate memory for the array in Java. For example,

```
// declare an array
double[] data;

// allocate memory
data = new Double[10];
```

Here, the array can store 10 elements. We can also say that the size or length of the array is 10.

In Java, we can declare and allocate memory of an array in one single statement. For example,

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

How to Initialize Arrays in Java?

In Java, we can initialize arrays during declaration. For example,

```
//declare and initialize and array
int[] age = {12, 4, 5, 2, 5};
```

Here, we have created an array named age and initialized it with the values inside the curly brackets.

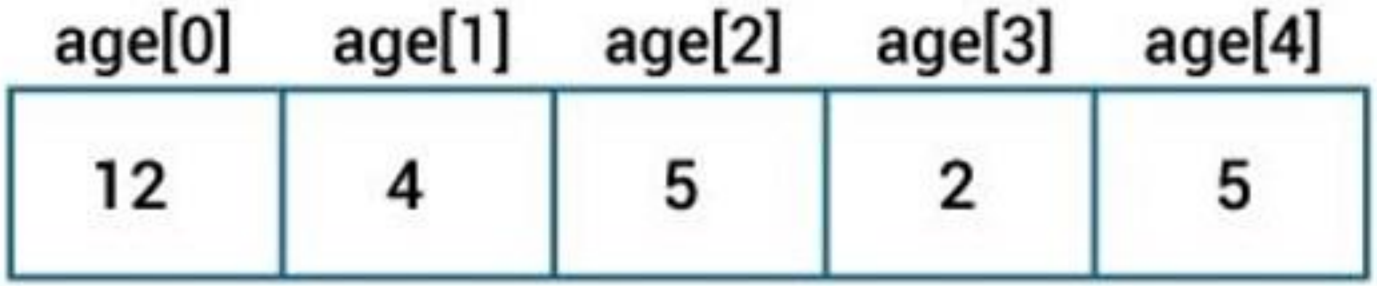
Note that we have not provided the size of the array.

In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).

In the Java array, each memory location is associated with a number. The number is known as an array index. We can also initialize arrays in Java, using the index number. For example,

```
// declare an array
int[] age = new int[5];

// initialize array
age[0] = 12;
age[1] = 4;
age[2] = 5;
..
```



How to Access Elements of an Array in Java?

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

```
// access array elements
array[index]
```

Example: Access Array Elements

```
class Main {
 public static void main(String[] args) {
  // create an array
  int[] age = {12, 4, 5, 2, 5};
  // access each array elements
  System.out.println("Accessing Elements of Array:");
  System.out.println("First Element: " + age[0]);
  System.out.println("Second Element: " + age[1]);
  System.out.println("Third Element: " + age[2]);
  System.out.println("Fourth Element: " + age[3]);
  System.out.println("Fifth Element: " + age[4]);
 }
}
```

Looping Through Array Elements

```
class Main {
 public static void main(String[] args) {

  // create an array
  int[] age = {12, 4, 5};

  // loop through the array
  // using for loop
  System.out.println("Using for Loop:");
  for(int i = 0; i < age.length; i++) {
   System.out.println(age[i]);
  }
 }
}
```

Example: Using the for-each Loop

```
class Main {
 public static void main(String[] args) {

  // create an array
  int[] age = {12, 4, 5};

  // loop through the array
  // using for loop
  System.out.println("Using for-each Loop:");
  for(int a : age) {
   System.out.println(a);
  }
 }
}
```

Example: Compute Sum and Average of Array Elements

```
class Main {
 public static void main(String[] args) {
  int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
  int sum = 0;
  Double average;
  // access all elements using for each loop
  // add each element in sum
  for (int number: numbers) {
   sum += number;
  }
  // get the total number of elements
  int arrayLength = numbers.length;
  // calculate the average
  // convert the average from int to double
  average =  ((double)sum / (double)arrayLength);
  System.out.println("Sum = " + sum);
  System.out.println("Average = " + average);
 }
}
```

Multidimensional Arrays

Arrays we have mentioned till now are called one-dimensional arrays. However, we can declare multidimensional arrays in Java.

A multidimensional array is an array of arrays. That is, each element of a multidimensional array is an array itself. For example,

```
double[][] matrix = {{1.2, 4.3, 4.0},
    {4.1, -1.1}};
```

Here, we have created a multidimensional array named matrix. It is a 2-dimensional array.

```
int[][] a = new int[3][4];
```

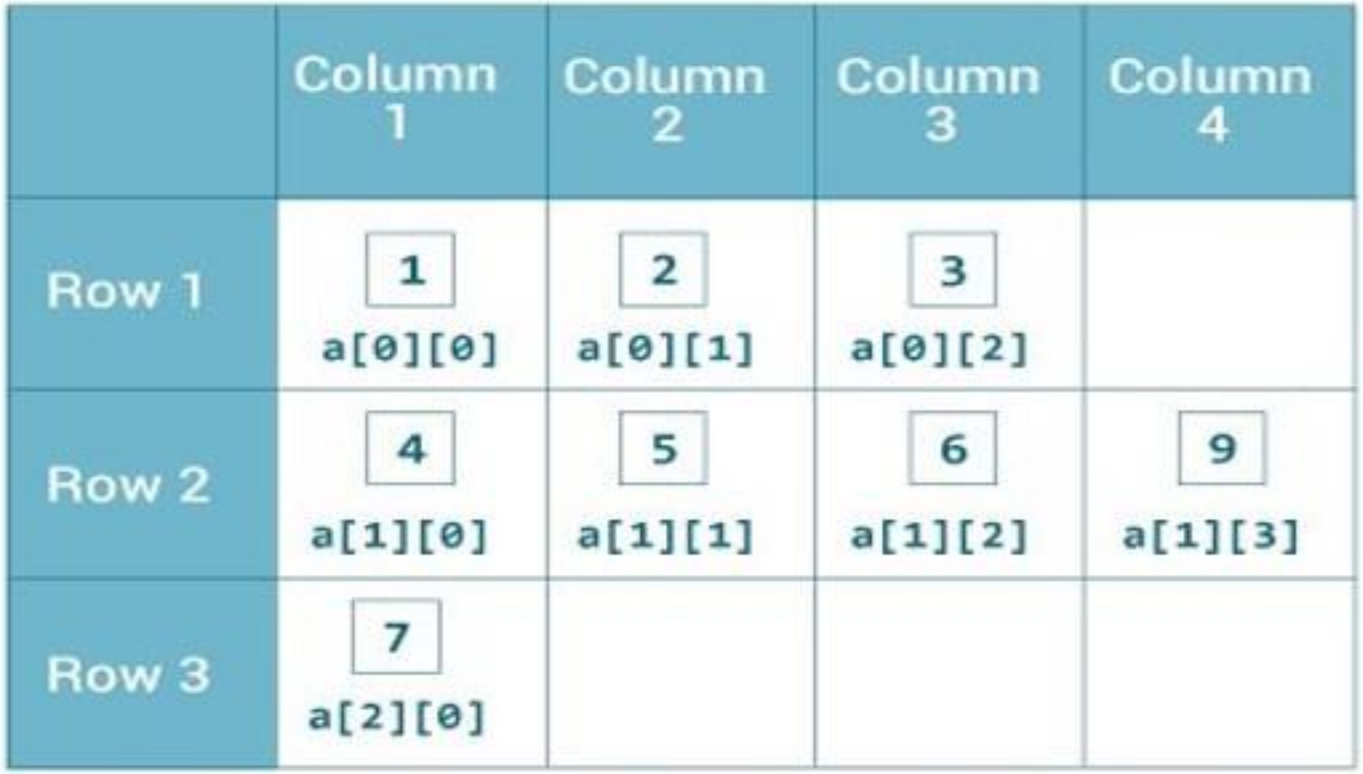
	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

How to initialize a 2d array in Java?

Here is how we can initialize a 2-dimensional array in Java.

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

And also, unlike C/C++, each row of the multidimensional array in Java can be of different lengths.



	Column 1	Column 2	Column 3	Column 4
Row 1	1 a[0][0]	2 a[0][1]	3 a[0][2]	
Row 2	4 a[1][0]	5 a[1][1]	6 a[1][2]	9 a[1][3]
Row 3	7 a[2][0]			

Example: Print all elements of 2d array Using Loop

```
class MultidimensionalArray {
   public static void main(String[] args) {

      int[][] a = {
         {1, -2, 3},
         {-4, -5, 6, 9},
         {7},
      };

      for (int i = 0; i < a.length; ++i) {
         for(int j = 0; j < a[i].length; ++j) {
            System.out.println(a[i][j]);
         }
      }
   }
}
```

Java Copy Arrays

In Java, we can copy one array into another. There are several techniques you can use to copy arrays in Java.

1. Copying Arrays Using Assignment Operator

Let's take an example,

```
class Main {
    public static void main(String[] args) {
        int [] numbers = {1, 2, 3, 4, 5, 6};
        int [] positiveNumbers = numbers;    // copying arrays
        for (int number: positiveNumbers) {
            System.out.print(number + ", ");
        }
    }
}
```

2. Using Looping Construct to Copy Arrays

```
import java.util.Arrays;
class Main {
   public static void main(String[] args) {
      int [] source = {1, 2, 3, 4, 5, 6};
      int [] destination = new int[6];
      // iterate and copy elements from source to destination
      for (int i = 0; i < source.length; ++i) {
         destination[i] = source[i];
      }

       // converting array to string
      System.out.println(Arrays.toString(destination));
   }
}
```

3. Copying Arrays Using arraycopy() method

In Java, the System class contains a method named arraycopy() to copy arrays. This method is a better approach to copy arrays than the above two.

The arraycopy() method allows you to copy a specified portion of the source array to the destination array. For example,

```
arraycopy(Object src, int srcPos,Object dest, int destPos, int length)
```

Here,
- src - source array you want to copy
- srcPos - starting position (index) in the source array
- dest - destination array where elements will be copied from the source
- destPos - starting position (index) in the destination array
- length - number of elements to copy

```
// To use Arrays.toString() method
import java.util.Arrays;

class Main {
  public static void main(String[] args) {
    int[] n1 = {2, 3, 12, 4, 12, -2};

    int[] n3 = new int[5];

    // Creating n2 array of having length of n1 array
    int[] n2 = new int[n1.length];

    // copying entire n1 array to n2
    System.arraycopy(n1, 0, n2, 0, n1.length);
    System.out.println("n2 = " + Arrays.toString(n2));

    // copying elements from index 2 on n1 array
    // copying element to index 1 of n3 array
    // 2 elements will be copied
    System.arraycopy(n1, 2, n3, 1, 2);
    System.out.println("n3 = " + Arrays.toString(n3));
  }
}
```

4. Copying Arrays Using copyOfRange() method

We can also use the copyOfRange() method defined in Java Arrays class to copy arrays. For example,

```
// To use toString() and copyOfRange() method
import java.util.Arrays;
class ArraysCopy {
   public static void main(String[] args) {
      int[] source = {2, 3, 12, 4, 12, -2};
      // copying entire source array to destination
      int[] destination1 = Arrays.copyOfRange(source, 0, source.length);
      System.out.println("destination1 = " + Arrays.toString(destination1));

      // copying from index 2 to 5 (5 is not included)
      int[] destination2 = Arrays.copyOfRange(source, 2, 5);
      System.out.println("destination2 = " + Arrays.toString(destination2));
   }
}
```

5. Copying 2d Arrays Using Loop

```
import java.util.Arrays;

class Main {
  public static void main(String[] args) {
         int[][] source = {
         {1, 2, 3, 4},
         {5, 6},
         {0, 2, 42, -4, 5}
         };
     int[][] destination = new int[source.length][];
     for (int i = 0; i < destination.length; ++i) {
       // allocating space for each row of destination array
       destination[i] = new int[source[i].length];
       for (int j = 0; j < destination[i].length; ++j) {
         destination[i][j] = source[i][j];
       }
     }
     // displaying destination array
     System.out.println(Arrays.deepToString(destination));

  }
}
```

Copying 2d Arrays using arraycopy()

```
import java.util.Arrays;
class Main {
   public static void main(String[] args) {
      int[][] source = {
           {1, 2, 3, 4},
           {5, 6},
           {0, 2, 42, -4, 5}
           };
      int[][] destination = new int[source.length][];
     for (int i = 0; i < source.length; ++i) {
          // allocating space for each row of destination array
          destination[i] = new int[source[i].length];
          System.arraycopy(source[i], 0, destination[i], 0, destination[i].length);
      }
      // displaying destination array
      System.out.println(Arrays.deepToString(destination));
   }
}
```

Java OOP

Java Class and Objects

Java is an object-oriented programming language. The core concept of the object-oriented approach is to break complex problems into smaller objects.

An object is any entity that has a **state** and **behavior**.

For example, a bicycle is an object. It has
•**States**: idle, first gear, etc
•**Behaviors**: braking, accelerating, etc.

Java Class

A class is a blueprint for the object. Before we create an object, we first need to define the class.

We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Since many houses can be made from the same description, we can create many objects from a class.

Create a class in Java

We can create a class in Java using the class keyword. For example,

```
class ClassName {
 // fields
 // methods
}
```

Here, fields (variables) and methods represent the state and behavior of the object respectively.

fields are used to store data
methods are used to perform some operations

For our bicycle object, we can create the class as

```
class Bicycle {

 // state or field
 private int gear = 5;

 // behavior or method
 public void braking() {
  System.out.println("Working of Braking");
 }
}
```

In the above example, we have created a class named Bicycle. It contains a field named gear and a method named braking().

Here, Bicycle is a prototype. Now, we can create any number of bicycles using the prototype. And, all the bicycles will share the fields and methods of the prototype.

Java Objects

An object is called an instance of a class. For example, suppose Bicycle is a class then MountainBicycle, SportsBicycle, TouringBicycle, etc can be considered as objects of the class.

Creating an Object in Java

Here is how we can create an object of a class.

```
className object = new className();
```

```
// for Bicycle class
Bicycle sportsBicycle = new Bicycle();

Bicycle touringBicycle = new Bicycle();
```

We have used the new keyword along with the constructor of the class to create an object. Constructors are similar to methods and have the same name as the class. For example, Bicycle() is the constructor of the Bicycle class.

Access Members of a Class

We can use the name of objects along with the . operator to access members of a class. For example,

```
class Bicycle {
 // field of class
 int gear = 5;
 // method of class
 void braking() {
   ...
 }
}
// create object
Bicycle sportsBicycle = new Bicycle();
// access field and method
sportsBicycle.gear;
sportsBicycle.braking();
```

```
class Lamp {
  // stores the value for light
 // true if light is on
 // false if light is off
 boolean isOn;
 // method to turn on the light
 void turnOn() {
  isOn = true;
  System.out.println("Light on? " + isOn);
 }
 // method to turnoff the light
 void turnOff() {
  isOn = false;
  System.out.println("Light on? " + isOn);
 }
}
class Main {
 public static void main(String[] args) {
  // create objects led and halogen
  Lamp led = new Lamp();
  Lamp halogen = new Lamp();
  // turn on the light by
  // calling method turnOn()
  led.turnOn();

  // turn off the light by
  // calling method turnOff()
  halogen.turnOff();
 }
}
```