# ***Arrays***

An array is an indexed collection of fixed number of homogenous data elements.

The main advantage of arrays is we can represent huge number of values using single variable so that readability of the code is improved.

But the main disadvantage of arrays is fixed in size i.e once we create an array there is no chance of increasing or decreasing the size based on our requirement. Hence to use array concept, compulsorily we should know the size in advance which in not always possible.

***One dimensional array declaration:***

* int[] x; ..recommended
* int []x;
* int x[];

The first one is recommended because name is clearly separated from type.

***At the time of declaration we can’t specify the size otherwise we will get compile time error.***

* int[6] x; ..invalid
* int[] x; ..valid

***Two dimensional array declaration:***

* int[][] x;
* int [][]x;
* int x[][];
* int[] []x;
* int[] x[];
* int []x[];
* which of the following are valid?
* int[] a, b;

a’s dimension = 1, b’s dimension = 1

* int[] a[], b;

a’s dimension = 2, b’s dimension = 1

* int[] a[], b[];

a’s dimension = 2, b’s dimension = 2

* int[] []a, b;

a’s dimension = 2, b’s dimension = 2

* int[] []a, b[];

a’s dimension = 2, b’s dimension = 3

* int[] []a, []b;

**Compile time error**

If we want to specify dimension before the variable , that facility is available only for first variable in a declaration. If we are trying to apply it for remaining variables we will get compile time error.

E.g : int[] []a ,[]b ,[]c;

Valid invalid invalid

***Three dimensional array declaration:***

* int[][][] a;
* int [][][]a;
* int a[][][];
* int[] [][]a;
* int[] []a[];
* int[] a[][];
* int[][] a[];
* int[][] []a;
* int [][]a[];
* int []a[][];

***One dimensional Array creation***

Every array in java is an object. Hence we can create arrays using **new** operator.

E.g : int[] a = new int[3];

For every array type corresponding classes are available and these classes are part of java language and not available at the programmer level.

* **int**[] x = **new** **int**[3];

System.***out***.print(x.getClass().getName());

**Output: [I**

|  |  |
| --- | --- |
| Array Type | Corresponding class name |
| int[] | **[I** |
| int[][] | **[[I** |
| double[] | [D |
| short[] | [S |
| byte[] | [B |
| boolean[] | [Z |

During array creation the size of the array should be specified compulsorily otherwise compile time error is encountered.

* int[] x = new int[3]; ..valid
* int[] x = new int[] ..invalid
* int[] x = new int[0] ..valid

It is legal to have an array with size 0 in java

* int[] x = new int[-3]

If we are trying to specify array size with some negative integer value then we will get **runtime exception** as : java.lang.NegativeArraySizeException: -3

To specify array size the allowed data types are byte, short, char, int. If we are trying to specify any other type then we will get compile time error.

* int[] x = new int[10]; ..valid
* int[] x = new int[‘a’]; ..valid
* byte b = 20;

int[] x = new int[b]; ..valid

* short s = 30;

int[] x = new int[s]; ..valid

* int[] x = new int [10L]; ..invalid

The maximum allowed array size in java is 2147483647 which is the maximum value of int data type.

1. int[] x = new int[2147483647]; ..valid
2. int[] x = new int[2147483648]; ..invalid

Even in the first example we might get runtime exception if sufficient heap memory not available.

***Two dimensional Array creation***

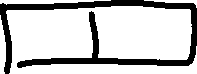
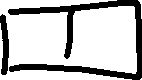
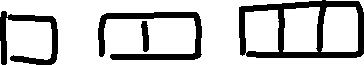
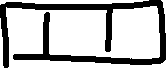
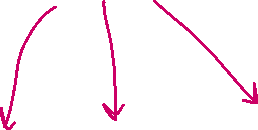
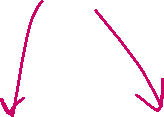
In java two dimensional array is not implemented using matrix style. Array of arrays approach is followed for multidimensional creation. The main advantage of this approach is memory utilization will be improved.

x

int[][] x = new int[2][];

x[0] = new int[2];

x[1] = new int[3];



int[][][] x = new int[2][][];

x[0] = new int[3][];

x[0][0] = new int[1];

x[0][1] = new int[2];

x[0][2] = new int[3];

x[1] = new int[2][2];

**Which of the following are valid?**

1. int[] a = new int[] ..invalid
2. int[] a = new int[3] ..valid
3. int[][] a = new int[][] ..invalid
4. int[][] a = new int[3][] ..valid
5. int[][] a = new int[][4] ..invalid
6. int[][] a = new int[3][4] ..valid
7. int[][][] a = new int[3][4][5] ..valid
8. int[][][] a = new int[3][4][] ..valid
9. int[][][]a = new int[3][][5] ..invalid
10. int[][][]a = new int[][4][5] ..invalid

***Array Initialisation:***

Once we create an array every element is by default initialized with default value.

Int[] x = new int[3]; X

EG: 1

SOP(x); 🡪 [I@5e265ba4

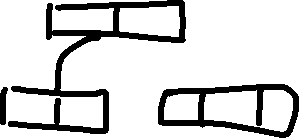
SOP(x[0]); 🡪 0

Whenever we are trying to print any reference variable internally tostring() method is called which is implemented by default to return the string in the following form : classname@hashcode\_in\_hexadecimal\_form.



EG : 2

**int**[][] x = **new** **int**[2][3];



System.***out***.println(x); 🡪[[I@5e265ba4

System.***out***.println(x[0]); 🡪[I@36aa7bc2

System.***out***.print(x[0][0]); 🡪0

Null null



EG: 3

**int**[][] x = **new** **int**[2][];



System.***out***.println(x); 🡪[[I@5e265ba4

System.***out***.println(x[0]); 🡪null



System.***out***.print(x[0][0]); 🡪 java.lang.NullPointerException: Cannot load from int array because "x[0]" is null

**NOTE:**

If we are trying to perform any operation on null then we will get runtime exception saying null pointer exception. Null is default type for objects

Once we create an array element by default is initialized with default values. If we are not satisfied with the default values then we can override these values with our customized values

EG : int[] a = new int[2];

Int[0] = 20;

Int[1] = 30;

Int[2]= 40; ..Runtime : ArrayIndexOutOfBoundException

Int[-6] = 30; .. Runtime : ArrayIndexOutOfBoundException

Int[2.4] = 90; .. Type mismatch: cannot convert from double to int

***Array declaration creation and initialisation in a single line***

***Shortcut representation:***

int[] x ;

x = new int[3];

x[0] = 10;

x[1] = 20; int[] x = {10, 20, 30};

x[2] = 30;

**Similarly,**

Char[] ch = {‘a’, ‘e’, ‘i’ ,’o’, ‘u’}

String[] = {“A”, “AA”, “AAA”}

We can use this shortcut for multidimensional arrays also.

EG 1.: int[][] x = {{10, 20}, {30, 40, 50}};

EG 2.: int[][][] x = {***{*** {10,20, 30}, {40, 50, 60}***}***, ***{*** {70, 80}, {90, 100, 110}***}***}

SOP(X[0][1][2]) 🡪60

SOP(X[1][0][1]) 🡪80

SOP(X[2][0][0]) 🡪invalid

SOP(X[1][2][0]) 🡪invalid

SOP(X[1][1][1]) 🡪100

SOP(X[2][1][0]) 🡪invalid

valid

* int x = 10; int x;

x = 10;

invalid

int[] x = {10, 20} int[] x;

x = {10,20}

If we want to use this shortcut compulsory we should perform all activities in a single line. If we try to divide into multiple lines then we will get compile time error as Array constants can only be used in initializers

* Length vs length()

***Length variable:***

Length is a final variable applicable for arrays. Length variable represents the size of the array

EG : int[] x = new int[6];

SOP(x.length()) 🡪 Cannot invoke length() on the array type int[]

SOP(x.length) 🡪o/p = 6

***Length Method:***

Length method is final method applicable for string objects. Length method returns number of characters present in a string.

EG : String s = “Darshana”;

SOP(s.length) 🡪 length cannot be resolved or is not a field

SOP(s.length()) 🡪 o/p = 8

**Note:**

Length variable is applicable for arrays but not for string objects whereas length method is applicable for string object but not for arrays.

EG :

String s = {“A”, “AA”, “AAA”};

* SOP(s.length); 🡪o/p = 3
* SOP(s.length()); 🡪invalid
* SOP(s[0].length); 🡪invalid
* SOP(s[0].length()); 🡪o/p = 1

In multidimensional arrays length variable only represents the base size but not total size.

EG : int[][] x = new int[6][3];

SOP(x.length) 🡪6

SOP(x[0].length) 🡪3

There is no direct way to find total length of multidimensional array but indirectly we can find as follows,

X[0].length + x[1].length + x[2].length + .. .

***Anonymous Arrays***

Sometimes we can declare an array without name. Such type of nameless arrays are called anonymous arrays.

The main purpose of anonymous arrays is just for instant use (one time usage)

We can create anonymous arrays as follows: new int[] {10, 20, 30, 40}

While creating anonymous arrays we cannot specify the size otherwise we will get compile time error.

new int[] {10, 20} ..valid

new int[3] {20,30} ..invalid

We can also create multidimensional anonymous arrays.

EG : new int[][]{{10, 20}, {30, 40}}

Based on our requirement we can give the name for anonymous arrays then it is no longer anonymous

EG : int[] x = new int[]{10, 20, 30};

***Array Element Assignment***

* ***Case 1:***

In case of primitive type arrays as array elements we can provide any type which can be implicitly promoted to declared type.

EG 1 : int[] x = new int[5];

x[0] = 10;

x[1] = ‘a’;

byte b = 20;

x[2] = b;

short s = 30;

x[3] = s;

x[4] = 10L ..invalid

EG 2 :

In case of float type arrays the allowed data types are byte, short, char, int, long, float

* ***Case 2:***

In case of object type arrays as array elements we can provide either declared type objects or its child class objects.

EG 1:

Object[] a = new Object[10];

a[0] = new object(); ..valid

a[1] = new String(“Darsh”); ..valid

a[2] = new Integer(10); ..valid

EG 2:Number class includes byte, short, int, long, float, double

Number[] n = new Number[10];

n[0] = new Integer(10); ..valid

n[1] = new Double(10.5); ..valid

n[2] = new String(“Durga”); ..invalid

* ***Case 3:***

For interface type arrays as array elements its implementation class objects are allowed.

EG 1:

Runnable[] r = new Runnable[10];

r[0] = new Thread(); ..valid

r[1] = new String(“Darsh”); ..invalid

|  |  |
| --- | --- |
| Array Type | Allowed element type |
| Primitive arrays | Any type which can be implicitly promoted to declared type |
| Object type arrays | Either declared type or its child class objects |
| Abstract class type arrays | Its child class objects |
| Interface type arrays | Its implementation class objects |

***Array Variable Assignment***

***Case 1:***

Element level promotions are not applicable at array level.

Char element can be promoted to int type whereas char array cannot be promoted to int array.

EG: int[] x = {10,34,45};

Char[] ch = {‘a’, ‘d’};

int[] b = x; ..valid

int[] c = ch; ..invalid

Which of the following promotions will be performed automatically?

|  |  |
| --- | --- |
| Char 🡪 int | Valid |
| Char[] 🡪 int[] | Invalid |
| Int 🡪 double | Valid |
| Int[] 🡪 double[] | Invalid |
| Float 🡪 int | Invalid |
| Float[] 🡪 int[] | Invalid |
| String 🡪 object | Valid |
| String[] 🡪 object[] | Valid |

But in the case of object type arrays child class type array can be promoted to parent class type array.

EG:

String[] s = {“aa”, “dcf”};

Object[] a = s;

***Case 2:***

Whenever we are assigning one array to another array internal elements will not be copied just reference variables will be reassigned.

EG:



Int[] a = {12,5,76,5,67,53};



Int[] b = {42,66};



1. a = b; ..valid
2. b = a; ..valid



***Case 3:***

Whenever we are assigning one array to another array the dimensions must be matched.

In the place of one dimensional int array we should provide one dimensional array only. If we are trying to provide any other dimension then we will get compile time error.

EG:

Int[][] a = new int[3][];

a[0] = new int[4][3]; ..invalid

a[0] = 10; ..invalid

a[0] = new int[2]; ..valid

**NOTE:**

Whenever we are assigning one array to another array both dimensions and type must be matched but sizes are not required to match.