



# Arrays in java


In this session we will learn:

- 1) Array Introduction
- 2) Array declaration
- 3) Array construction
- 4) Array initialization
- 5) Array declaration, construction, initialization in a single line.
- 6) length Vs length() method
- 7) Anonymous arrays
- 8) Array element assignments
- 9) Array variable assignments.



## Array Introduction

An array in java is a collection of values of similar data type, stored in the contiguous memory location, sharing a common name, and distinguished by an element's index.



## Features of Arrays in Java


1. It is generally used to store a group of elements (or values) or a collection of related data items that share a common name.
2. The elements of an array can be either primitive data types or reference types (including itself array type).
3. Elements in Java array are always numbered from 0 to  $(n - 1)$  where  $n$  is the individual elements in the array.
4. The elements in the array are ordered or arranged in contiguous memory locations.
5. Arrays are objects in Java. Therefore, they can hold reference variables of other objects.
6. Arrays are created dynamically during runtime in Java.

## Features of Arrays in Java

7. They are dynamic, created on the heap memory.
8. The length of an array is the number of elements it contains.
9. Arrays may be assigned to variables of the Object type.
10. Any method of Object class can be called on the array.
11. Array objects implement **Cloneable** and **Serializable** interfaces..
12. Arrays in Java can be duplicated with **Object.clone()** method.
13. They can be verified for equality with **Arrays.equals()** method.



## Types of Arrays in Java

1. Single dimensional arrays (or 1D array called one-dimensional array)
  2. Multidimensional arrays (or 2D, 3D arrays)
- 



## One dimensional Array

1. A one-dimensional array contains a list of variables of the same type and is accessed through a common name.
2. Each variable in the array is called an array element.
3. Accessing of arrays is checked at runtime, an attempt to use an index that is less than zero or greater than or equal to the length of array generates an exception named **IndexOutOfBoundsException**.

# Array declarations:

## Single dimensional array declaration:

### Example:

`int[] a; //recommended to use because name is clearly separated from the type`

`int []a;`

`int a[];`

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

### Example:

`int[] a;//valid`

`int[5] a;//invalid`

## Example:

Specifies an array of  
variable of type int

We are creating a  
new array object

**int[ ] num = new int[6];** // An array of 6 integers

The name of array

Array object is of type int  
and has six elements



Fig: Creating an array object in Java



## Which of the following declarations are valid?

- 1) `int[] a1,b1;`
- 2) `int[] a2[],b2;`
- 3) `int[] []a3,b3;`
- 4) `int[] a,[]b; // invalid`

## Points to remember:

If we want to specify the dimension before the variable that rule is applicable only for the 1st variable.

Second variable onwards we can't apply in the same declaration.

Example:

```
int[] []a,[]b;
```

**invalid**

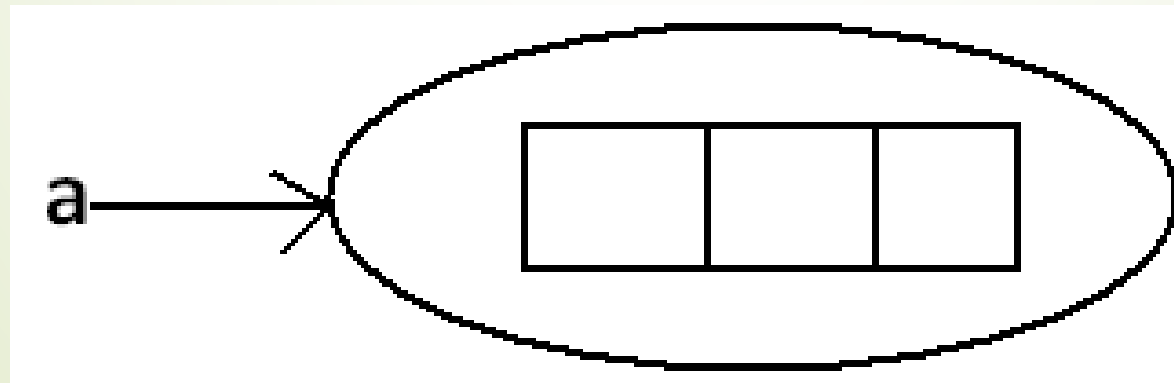
**valid**

## Array construction:

Every array in java is an object hence we can create by using new operator.

**Example:**

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



## Every array type corresponding classes are available

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

Array Type	corresponding class name
int[ ]	[I
int[ ][ ]	[[I
double[ ]	[D

## Rules to remember:

### Rule 1:

At the time of array creation compulsory we should specify the size otherwise we will get compile time error.

### Example:

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

```
int[ ] a=new int[ ];      //C.E:array dimension missing
```

## Rules to remember:

### Rule 2:

It is legal to have an array with size zero in java.

Example:

```
int [ ] a=new int [ 0 ];  
System.out.println(a.length); // 0
```

### Rule 3:

If we are taking array size with -ve int value then we will get runtime exception saying NegativeArraySizeException.

Example:

```
int[ ] a=new int[-3]; //R.E:NegativeArraySizeException
```

## Rules to remember:

### Rule 4:

The allowed data types to specify array size are byte, short, char, int.

By mistake if we are using any other type we will get compile time error.

### Example:

```
int[ ] a=new int['a'];           //(valid)
```

```
byte b=10;
```

```
int[ ] a=new int[ b ];           //(valid)
```

```
short s=20;
```

```
int[ ] a=new int[ s ];           //(valid)
```

```
int[ ] a=new int[ 10l ];         //C.E:possible loss of precision//(invalid)
```

```
int[ ] a=new int[ 10.5 ];        //C.E:possible loss of precision//(invalid)
```

## Rules to remember:

Rule 5:

The maximum allowed array size in java is maximum value of int size **[2147483647]**.

Example:

```
int[ ] a1=new int[ 2147483647 ]; // (valid)
```

```
int[ ] a2=new int[ 2147483648 ];
```

```
//C.E:integer number too large: 2147483648(invalid)
```

In the first case we may get RE : OutOfMemoryError.



## Array Initialization:

Whenever we are creating an array every element is initialized with default value automatically.

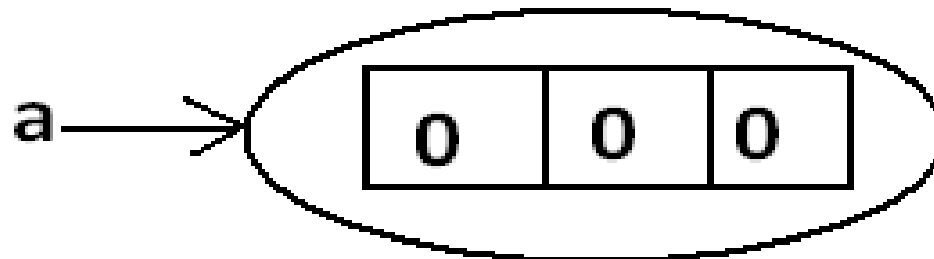
### Example 1:

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

```
System.out.println[a];    //[I@3e25a5
```

```
System.out.println(a[0]); //0
```

**Note:** Whenever we are trying to print any object reference internally **toString()** method will be executed which is implemented by default to return the following. **classname@hexadecimalstringrepresentationofhashcode**.



## Array Initialization:

Once we created an array all its elements by default initialized with default values.

If we are not satisfied with those default values then we can replays with our customized values.

### Example:

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

```
a[0]=10;
```

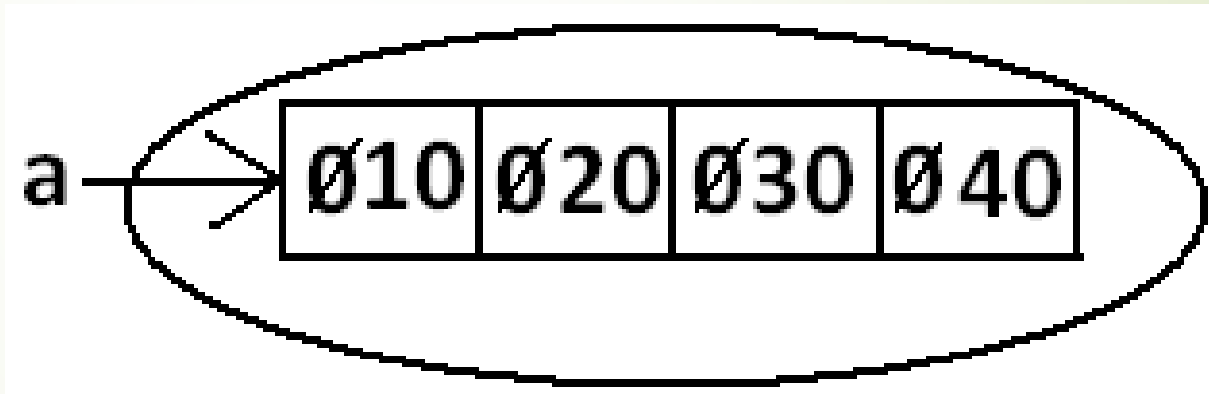
```
a[1]=20;
```

```
a[2]=30;
```

```
a[3]=40;
```

```
a[4]=50;    //R.E: ArrayIndexOutOfBoundsException: 4
```

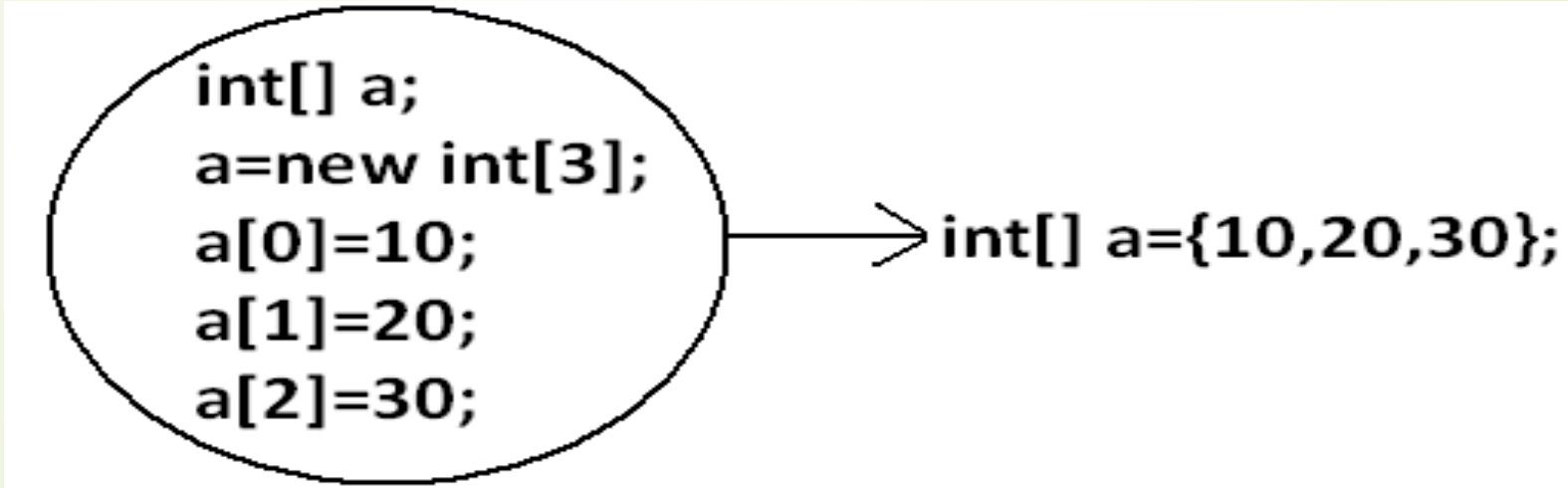
```
a[-4]=60;   //R.E: ArrayIndexOutOfBoundsException: -4
```



## Declaration, construction and initialization of an array in a single line:

We can perform declaration, construction and initialization of an array in a single line.

Example:



```
int[] a;  
a=new int[3];  
a[0]=10;  
a[1]=20;  
a[2]=30;
```

The diagram illustrates the transformation of a multi-line array declaration and initialization into a single line. On the left, a circle contains the following code: `int[] a;`, `a=new int[3];`, `a[0]=10;`, `a[1]=20;`, and `a[2]=30;`. An arrow points from this circle to the right, where the single-line equivalent is shown: `int[] a={10,20,30};`.

```
int[] a={10,20,30};
```

```
char[ ] ch={'a','e','i','o','u'}; // (valid)
```

```
String[ ] s={"John","kate","Andy","Meson"}; // (valid)
```

## Declaration, construction and initialization of an array in a single line:

If we are trying to divide into multiple lines then we will get compile time error.

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

```
int[] x;  
x=new int[3];  
x={10,20,30};
```

**C.E:illegal start of expression**

## length Vs length():

### length:

1. It is the final variable applicable only for arrays.
2. It represents the size of the array.

### Example:

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

```
System.out.println(x.length()); //C.E: cannot find symbol
```

```
System.out.println(x.length); //3
```

## length Vs length():

### length() method:

1. It is a final method applicable for String objects.
2. It returns the no of characters present in the String.

### Example:

```
String str="India";
```

```
System.out.println(str.length);    //C.E:cannot find symbol
```

```
System.out.println(str.length());  //5
```

## Anonymous Arrays:

Sometimes we can create an array without name such type of nameless arrays are called anonymous arrays.

The main objective of anonymous arrays is **"just for instant use"**.

We can create anonymous array as follows:

```
new int[ ] {10,20,30,40}; //(valid)
```

```
new int[ ][ ] {{10,20},{30,40}}; // (valid)
```



## Anonymous Arrays:

At the time of anonymous array creation we can't specify the size otherwise we will get compile time error.

### Example:

```
new int[3]{10,20,30,40}; //C.E: ';' expected(invalid)
```

```
new int[ ] {10,20,30,40}; //(valid)
```

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

### Example:

```
int[ ] arr = new int[ ]{10,20,30,40}; // (valid)
```



# Array element assignments:

## Case 1:

In the case of primitive array as array element any type is allowed which can be promoted to declared type.

## Example 1:

For the int type arrays the allowed array element types are byte, short, char, int.

```
int[ ] a=new int[10];
```

```
a[0]=97;//(valid)
```

```
a[1]='a';//(valid)
```

```
byte b=10;
```

```
a[2]=b;//(valid)
```

```
short s=20;
```

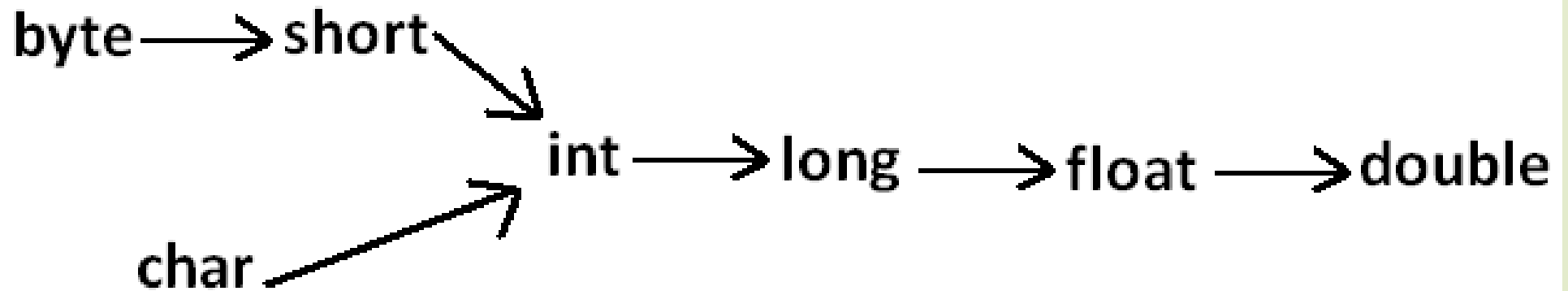
```
a[3]=s;//(valid)
```

```
a[4]=10l;//C.E:possible loss of precision
```

## Array element assignments:

### Example 2:

For float type arrays the allowed element types are byte, short, char, int, long, float.



## Array element assignments:

### Case 2:

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

### Example 1:

```
Object[] a=new Object[10];  
a[0]=new Integer(10);      //(valid)  
a[1]=new Object();          //(valid)  
a[2]=new String("John");    //(valid)
```

## Array element assignments:

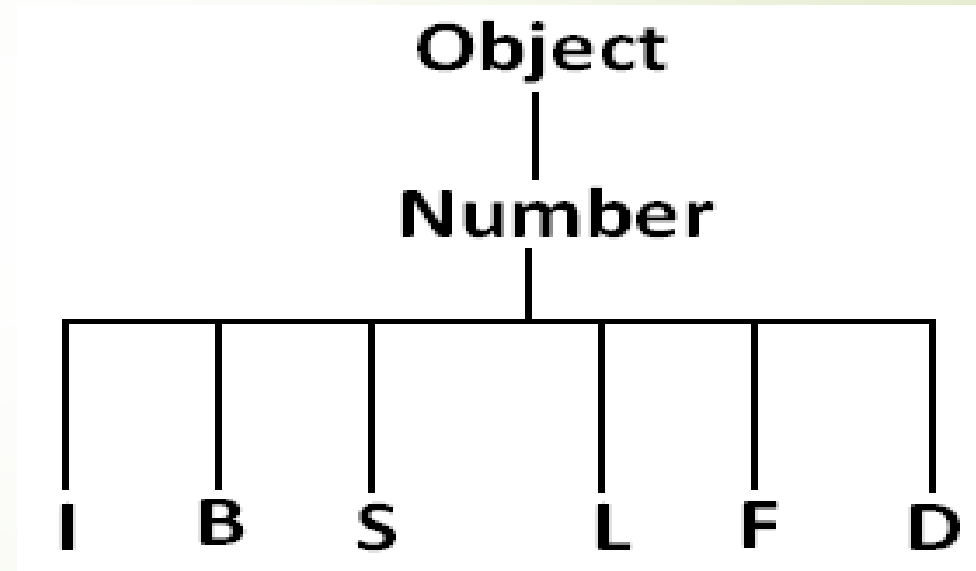
### Case 2:

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

### Example 2:

```
Number[ ] n=new Number[10];  
n[0]=new Integer[10];    //(valid)  
n[1]=new Double(10.5);  //(valid)
```

```
n[2]=new String("John");    //C.E:incompatible types //(invalid)
```



## Array element assignments:

### Case 3:

In the case of interface type arrays as array elements we can provide its implemented class objects.

### Example:

```
Runnable[ ] r=new Runnable[10 ]; // runnable is an interface  
r[0]=new Thread();  
r[1]=new String("Mike");           //C.E: incompatible types
```

## Array element assignments:

### Summary:

Array Type	Allowed Element Type
1) Primitive arrays.	1) Any type which can be promoted to declared type.
2) Object type arrays.	2) Either declared type or its child class objects allowed.
3) Interface type arrays.	3) Its implemented class objects allowed.
4) Abstract class type arrays.	4) Its child class objects are allowed.

## Array variable assignments:

### Case 1:

Element level promotions are not applicable at array object level.

**Ex :** A char value can be promoted to **int** type but char array cannot be promoted to **int** array.

### Example:

```
int[ ] a = {10,20,30};
```

```
char[ ] ch = {'a','b','c'};
```

```
int[ ] b = a;      //(valid)
```

```
int[ ] c = ch;    //C.E:incompatible types (invalid)
```



## Array variable assignments:

Which of the following promotions are valid?

- 1)char \_\_\_\_\_int (valid)
- 2)char[]\_\_\_\_\_int[] (invalid)
- 3)int \_\_\_\_\_long (valid)
- 4)int[] \_\_\_\_\_long[] (invalid)
- 5)double \_\_\_\_\_float (invalid)
- 6)double[]\_\_\_\_\_float[] (invalid)
- 7)String\_\_\_\_\_Object (valid)
- 8)String[]\_\_\_\_\_Object[] (valid)

**Note:** *In the case of object type arrays child type array can be assign to parent type array variable.*

**Example:**

```
String[ ] str ={"A","B"};
```

```
Object[ ] obj = str;
```



## Array variable assignments:

### Case 2:

Whenever we are assigning one array to another array internal elements won't be copy just reference variables will be reassigned hence sizes are not important but types must be matched.

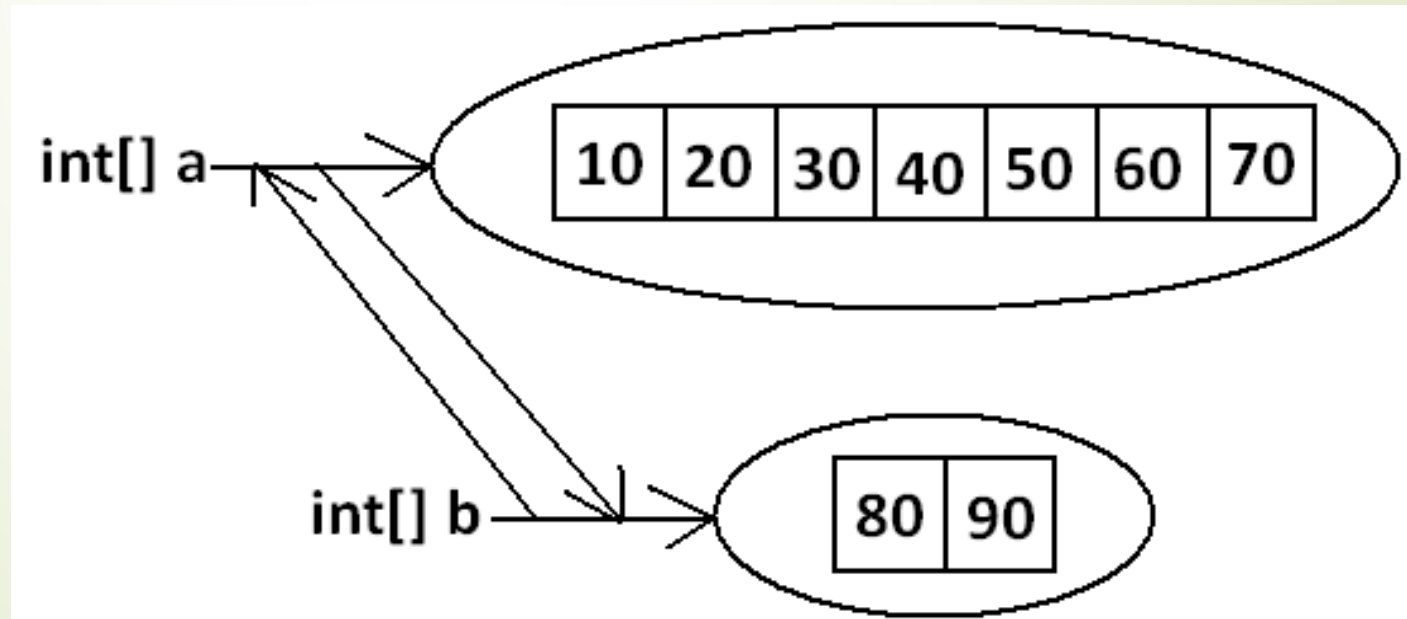
### Example:

```
int[ ] a = {10,20,30,40,50,60,70};
```

```
int[ ] b = {80,90};
```

```
a = b; //(valid)
```

```
b=a; //(valid)
```



## Two dimensional array declaration:

### Example:

```
int[ ][ ] a;
```

```
int [ ][ ]a;
```

```
int a[ ][ ];
```

```
int[ ] [ ]a;
```

```
int[ ] a[ ];
```

```
int [ ]a[ ];
```

**All are valid.(6 ways)**

## Which of the following declarations are valid?

- 1) `int[] a=new int[]`      `//C.E: array dimension missing(invalid)`
- 2) `int[][] a=new int[3][4];`      `//(valid)`
- 3) `int[][] a=new int[3][];`      `//(valid)`
- 4) `int[][] a=new int[][4];`      `//C.E:']' expected(invalid)`

## Two dimensional array declaration:

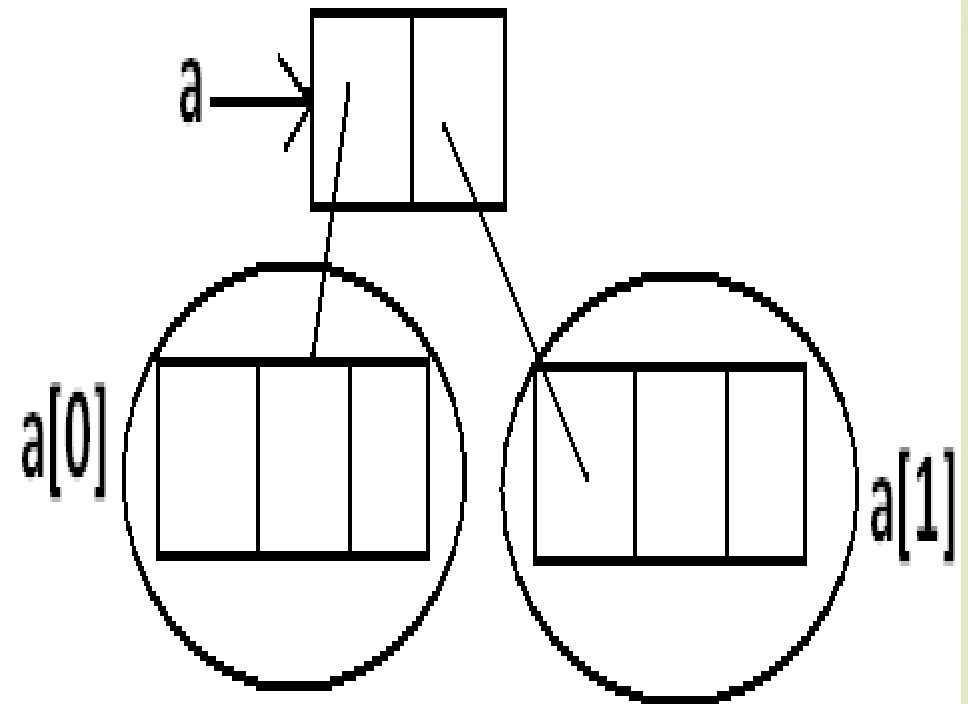
`int[][] a=new int[2][3];` base size

`System.out.println(a);`      `//[[I@3e25a5`

`System.out.println(a[0]);`      `//[I@19821f`

`System.out.println(a[0][0]);`      `//0`

memory representation



## Two dimensional array declaration:

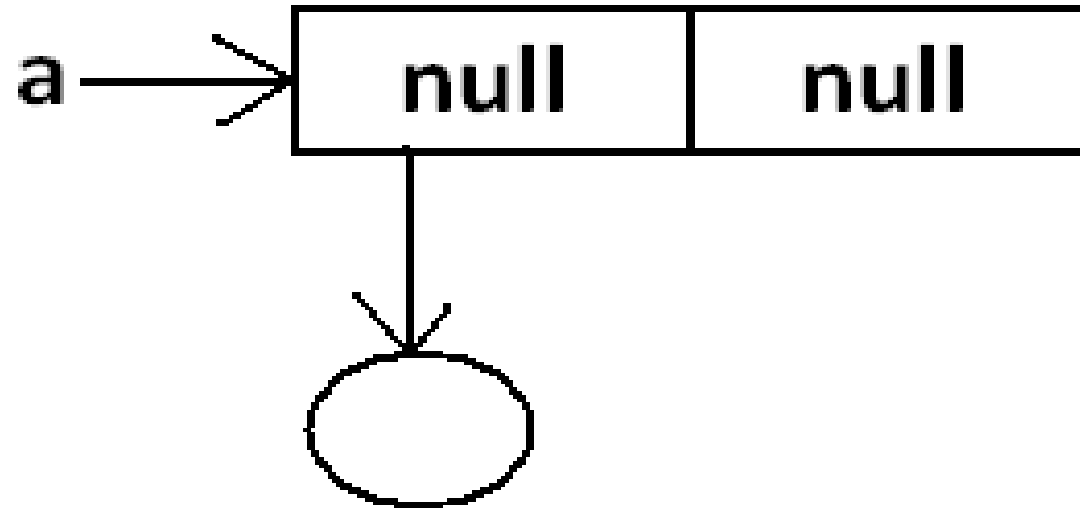
### Example 3:

```
int[][] a=new int[2][ ];
```

```
System.out.println(a);      //[I@3e25a5
```

```
System.out.println(a[0]);    //null
```

```
System.out.println(a[0][0]); //R.E: NullPointerException
```

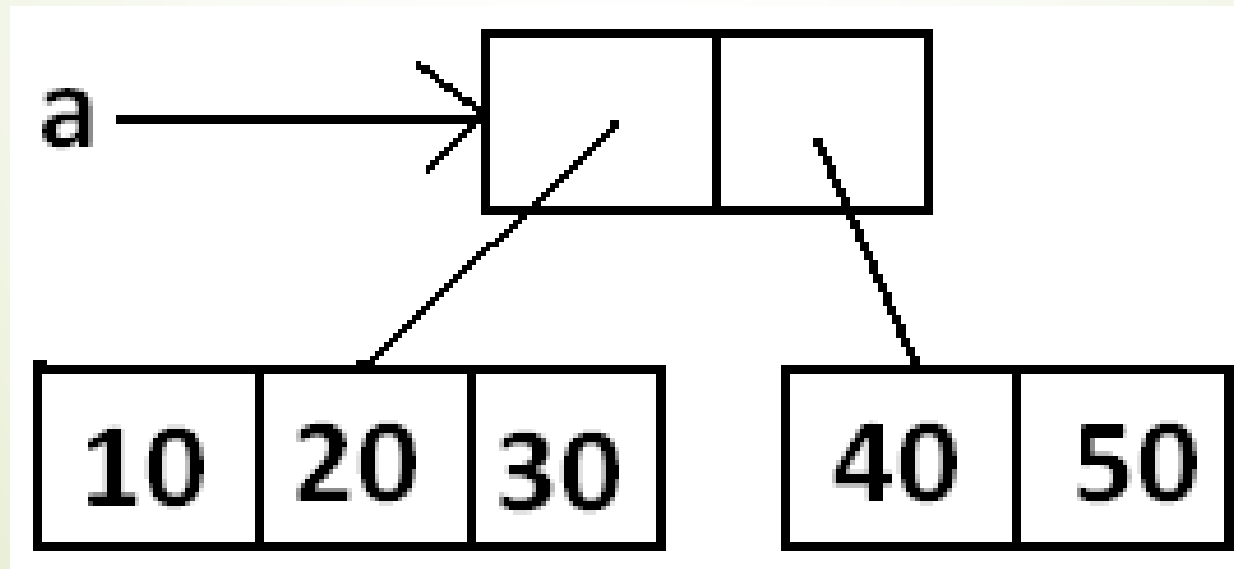


## Two dimensional array declaration:

We can extend this short cut even for multi dimensional arrays also.

**Example:**

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



## Two dimensional array:

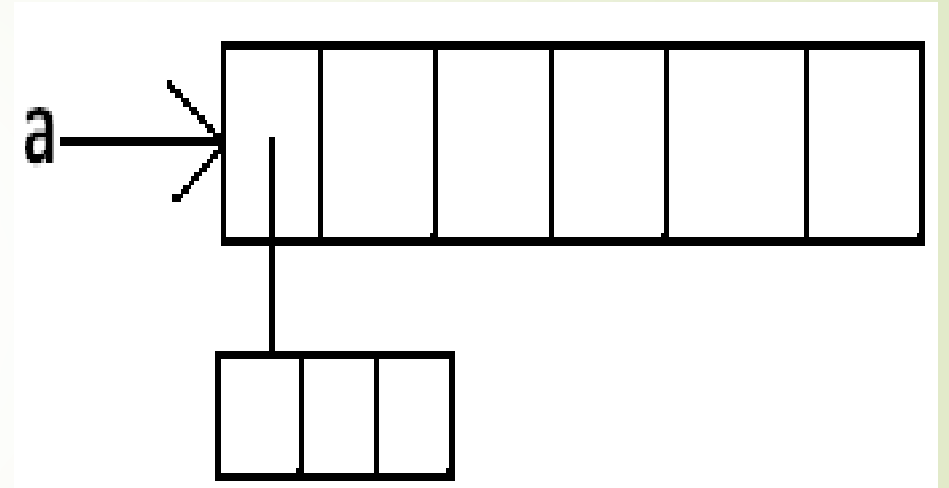
In multidimensional arrays length variable represents only base size but not total size.

Example:

```
int[ ][ ] a=new int[6][3];
```

```
System.out.println(a.length);    //6
```

```
System.out.println(a[0].length); //3
```



There is no direct way to find total size of multi dimensional array but indirectly we can find as follows:

**$x[0].length + x[1].length + x[2].length + \dots$**

## Jagged Array

Jagged arrays are also known as ragged arrays.

They are the arrays containing arrays of different length.

Jagged array may consist of various rows with different columns in each row.

### **Example:**

```
int[ ][ ] jaggedArray = new int[3][ ];  
jaggedArray[0] = new int[ ]{0,1,2,3};  
jaggedArray[1] = new int[ ]{4,5};  
jaggedArray[2] = new int[ ]{6,7,8};
```



# Assignment

*		1	2	3	4	5
-----						
1		1	2	3	4	5
2		2	4	6	8	10
3		3	6	9	12	15
4		4	8	12	16	20
5		5	10	15	20	25