

JAVA ARRAY

ARRAY

- Array is a **linear** or **sequential data structure** to store the similar type of data/elements.
- It arranges the element in **contiguous memory locations** using indices like 0, 1, 2, ... n .
- An array is a **group of elements of the same type**. These elements are grouped together under a common name (homogenous data structure)
- In java, arrays are **reference type (object type)**
- An array variable does not actually store the array - it is a reference variable that points to an array object.

POINTS ABOUT ARRAY

- In java, it is possible to create array of objects.
- In c/c++ an array is **value type**. But in c#/java, an array is **reference type**. So it must be created with the help of new modifier.
- Array is group of elements having a same name. (homogenous elements)
- Java supports both **single dimensional & multidimensional arrays**.
- A specific element in an array is accessed by using its **index position**
- The **size of the array** must be mentioned at the time of creation.

Range

Range of array : 0 to n-1

TYPES OF ARRAYS

- Java supports two different types of arrays. They are

1. Regular (Rectangular Array)

- One Dimensional Array
 - Array with **only one** subscript
- Multi Dimensional array
 - Array with **more than one** subscript

2. Jagged Array (Ragged Arrays)

- Array of arrays

Note

- The subscript of an array always **begins with 0**
- The subscript is **an integer**
- The subscript **cannot be a negative number**
- The subscript should be enclosed with **[]**

ONE DIMENSIONAL ARRAY (1D ARRAY)

- It involves two step process

Method 1

- Seperate declaration and definition into two line statements

OR

- Combine both declaration and definition into one line statement

Declaration

- Just declare the array with base type
- Here the array actually points to null.

Syntax

```
type arrayname[];
```

OR

```
type []arrayname;
```

Example

```
int arr[];
```

OR

```
int[] arr;
```

Definition (Creation)

- Allocating memory spaces for the declared array is called as array creation
- This is done by using **new** modifier.

Syntax

```
arrayname=new type[size];
```

Example

```
arr=new int[5];
```

Combining Declaration & Definition

- We can combine both declaration and definition of array into one line statement

Syntax

```
type arrayname[]=new type[size];
```

OR

```
type []arrayname=new type[size]
```

Example

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

OR

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

Method 2 (Direct Array Initializaton)

- Combine declaration, definition and initialization (without using new modifier) into one line statement.
- Stroing values in the array element is called as array initialization

Syntax

```
type arrayname[]={list of values};
```

OR

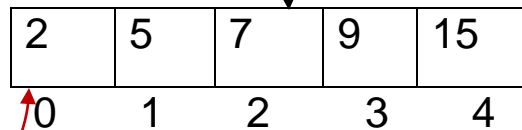
```
type[] arrayname={list of values}
```

Example

```
int[] num={2,5,7,9,15};
```

OR

```
int num[]={2,5,7,9,15};
```



2	5	7	9	15
0	1	2	3	4

← index locations

- `int num[]={2,5,7,9,15};` // declaration, creation and initialization is done at the same line.
- Here, the size of the array will be automatically measured according to number of elements in array.
- The num points to the **base address of the array**
- The first location of the array is called as base address

Index based Storage

- Like c/c++, we can store elements to array at any location **using array index**.

Syntax

```
arrayname[index value]=constant / value
```

Example

```
arr[0]=25;           // store value 25 at index position 0  
arr[3]=91;           // store value 91 at index position 3
```

Array length

- Size of an array can be obtained by accessing the builtin property length.
- The length property provides the [total number of elements in the array](#).

Syntax

```
<array-name>.length;
```

Example

```
{  
int[] marks={35,92,73};  
int size=marks.length;  
System.out.println("size : "+size);  
}
```

Output

```
size=3;
```

I. 1D ARRAY : STORAGE & RETRIEVAL

(SD.java)

1. SOURCE CODE

```
import java.io.*;

public class SD
{
    int[] num=new int[100];
    int n;
    // data input stream is used to get the dynamic keyboard input
    DataInputStream ds=new DataInputStream(System.in);
    // store the list of dynamic elements one by one into an array
    public void store()throws Exception
    {
        System.out.print("Enter the no of elements to store : ");
        // convert String number to int number using parsing method
        n=Integer.parseInt(ds.readLine());
        for(int i=0;i<n;i++)
        {
            System.out.print("Element #"+(i+1)+" : ");
            int ele=Integer.parseInt(ds.readLine());
            // store the elements into array using index
            num[i]=ele;
        }
    }
    // display the contents of an array
    public void disp()
```

```
{  
    System.out.println("Elements in the array");  
    for(int i=0;i<n;i++)  
    {  
        System.out.print(num[i]+" ");  
    }  
    System.out.println();  
}
```

// main method : execution entry point

```
public static void main(String[] args)throws Exception  
{  
    System.out.println("=====");  
    System.out.println("\t1D Array Storage & Retrieval");  
    System.out.println("=====");
```

// object creation

```
    SD obj=new SD();
```

// call the instance methods

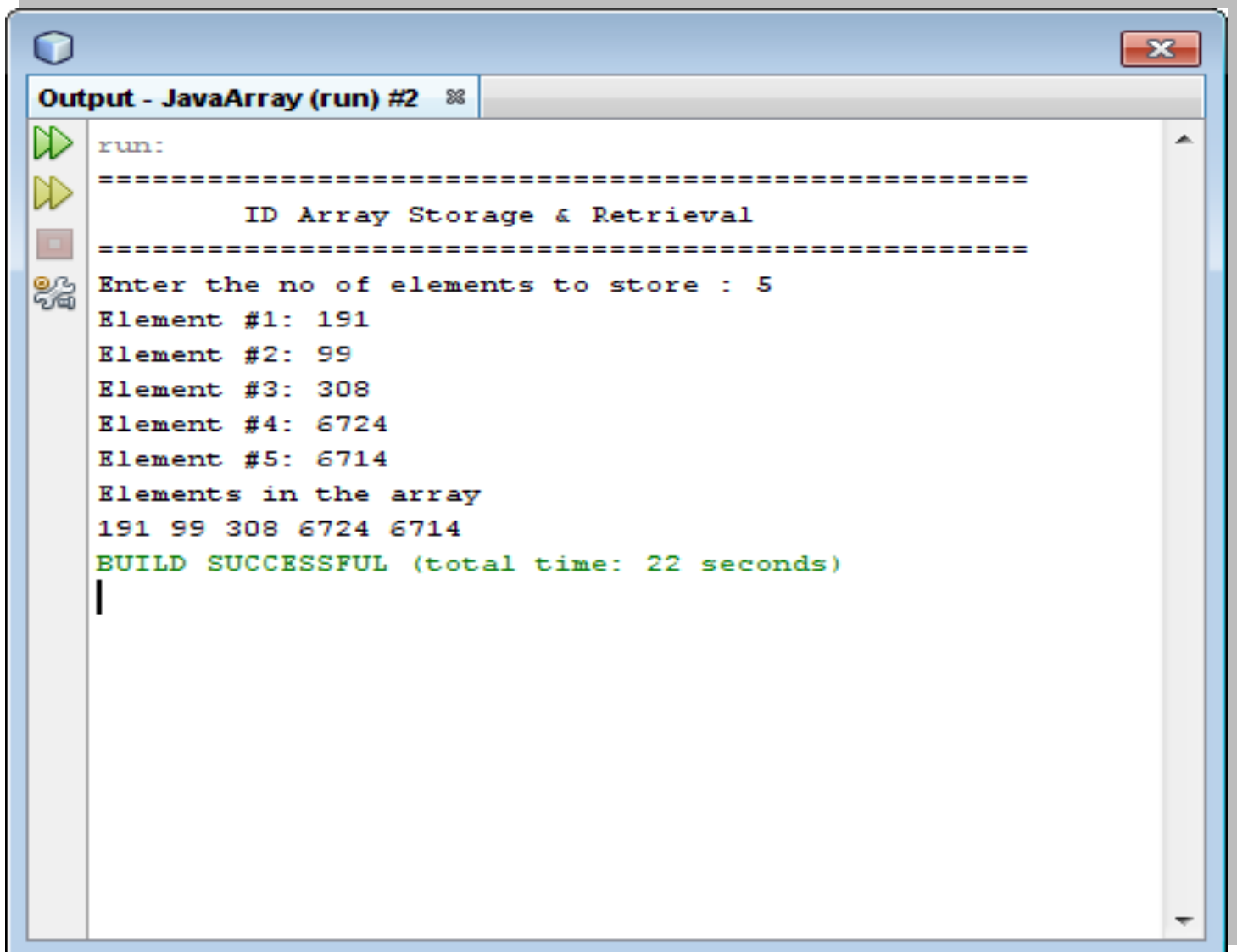
```
    obj.store();
```

```
    obj.disp();
```

```
}
```

```
}
```


2. OUTPUT



```
run:
=====
      ID Array Storage & Retrieval
=====
Enter the no of elements to store : 5
Element #1: 191
Element #2: 99
Element #3: 308
Element #4: 6724
Element #5: 6714
Elements in the array
191 99 308 6724 6714
BUILD SUCCESSFUL (total time: 22 seconds)
|
```

II. 1D ARRAY : STORAGE & RETRIEVAL

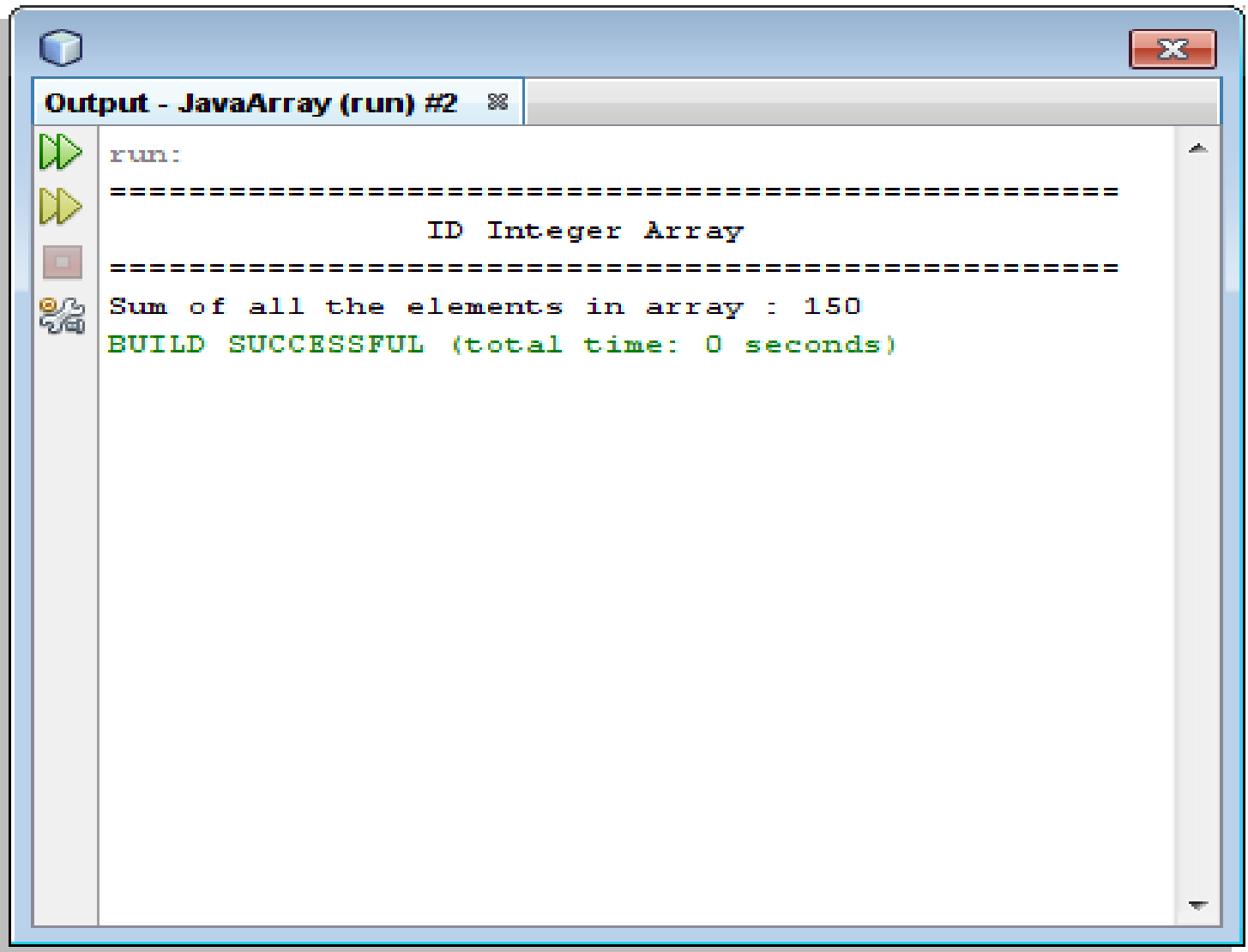
(Sum.java)

1. SOURCE CODE

```
public class Sum
{
// 1D array creation
    int ele[]={34,56,10,20,30};
    int s=0;
// instance method
    public void findsum()
    {
        for(int i=0;i<ele.length;i++)
        {
            s+=ele[i];
        }
        System.out.println("Sum of all the elements in array : "+s);
    }
// main method
    public static void main(String[] args)throws Exception
    {
        System.out.println("=====");
        System.out.println("\t\t1D Integer Array ");
        System.out.println("=====");
// object creation
        Sum obj=new Sum();
// calling instance method using an object
```

```
obj.findsum();  
}  
}
```

2. OUTPUT



Enhanced for loop (foreach loop)

- The **enhanced for loop** is a specialized for loop that simplifies looping through an **array** or a **collection**.
- It gives more performance than simple for loop
- It is a new feature which was introduced in **java 5.0 / J2SE 5.0**.
- Here **we don't need to mention the condition, increment / decrement values**.
- Instead of having three components, the enhanced for loop has two components.

Syntax

```
for(basetype var: exp / collections)
{
    // code
}
```

Where,

- Base type can be value type (e.g: int, float) or reference type (e.g: Object, String)
- var → loop variable
- exp can be → array element
- collections can be → ArrayList, Vector, Stack, LinkedList, etc,...

usage

- This faster loop is mostly used in large document retrieval

Disadvantages

- It is used to retrieve elements only in forward direction (first to last order). It **does not support reverse direction**.
- It **does not support array indexing**.

III. 1D ARRAY RETRIEVAL USING e-FOR LOOP

(Adisp.java)

1. SOURCE CODE

```
public class Adisp
{
    // 1D array creation
    int num[]={2,3,5,7,8,9,15};
    // instance method
    public void calc()
    {
        System.out.println("Retrieval using simple for loop");
        // simple for loop
        for(int i=0;i<num.length;i++)
        {
            System.out.print(num[i]+" ");
        }
        System.out.println();
        System.out.println("Retrieval using faster for loop");
        // faster for loop
        for(int i:num)
        {
            System.out.print(i+" ");
        }
        System.out.println();
    }
}
```

// main method

```
public static void main(String[] args)throws Exception
{
    System.out.println("=====");
    System.out.println("\t\tID Array Retrieval ");
    System.out.println("=====");
```

// object creation

```
    Adisp obj=new Adisp();
```

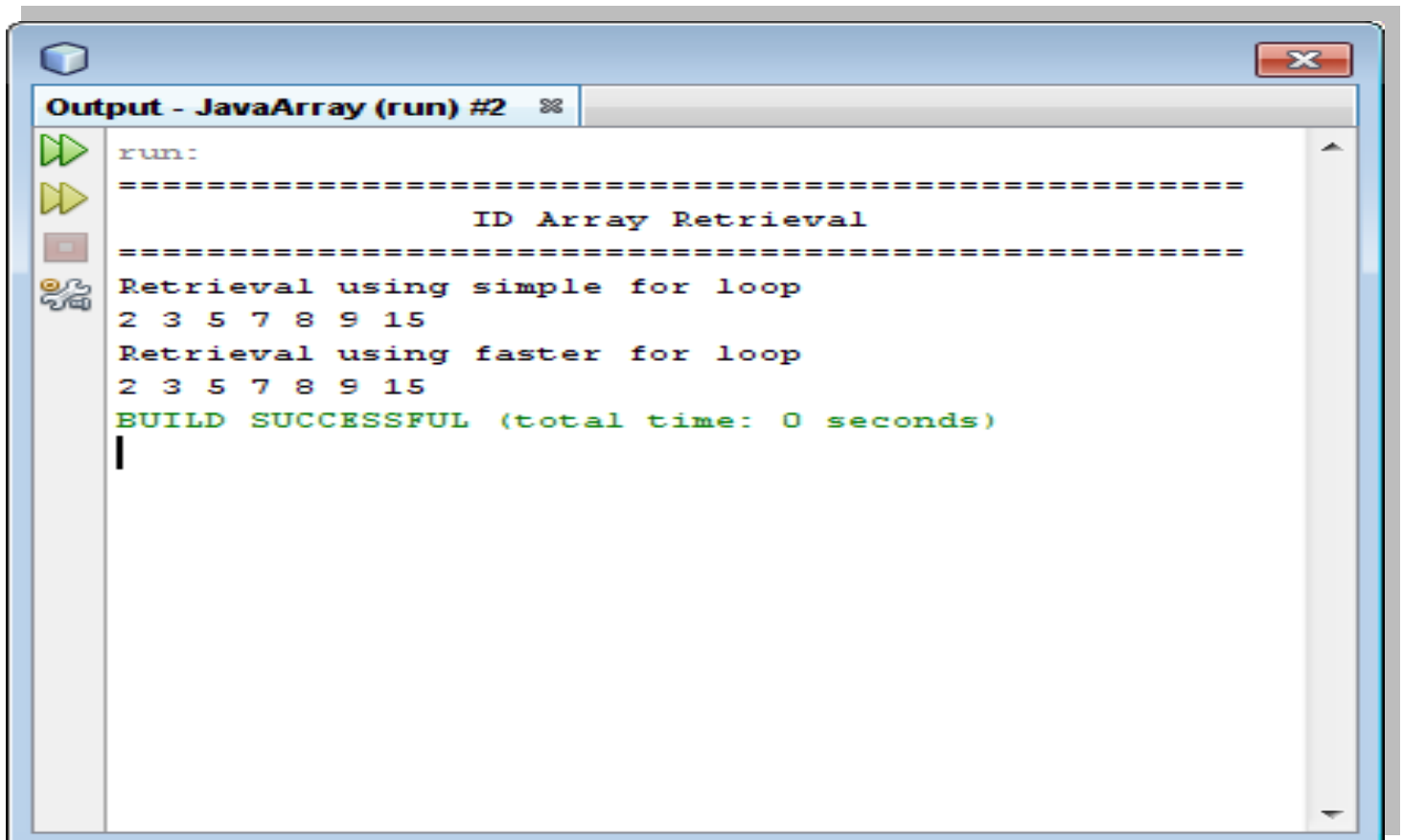
// calling instance method

```
    obj.calc();
```

```
}
```

```
}
```

2. OUTPUT



```
Output - JavaArray (run) #2
run:
=====
                ID Array Retrieval
=====
Retrieval using simple for loop
2 3 5 7 8 9 15
Retrieval using faster for loop
2 3 5 7 8 9 15
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. MULTIDIMENSIONAL ARRAY

- Array with **more than one subscript**
- 2-D array is **an array of 1-d arrays**.
- We store multiple one-d arrays in another one-d array to form 2-d array.
- It is generally used to **store row-column data structure**.

Table / Matrix

- Collection of rows and columns.

Let as take 4 x 3 Matrix

A[4][3]

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]
[3][0]	[3][1]	[3][2]

Syntax

Method 1

type array-name[][]; **// declaration**

array-name=new type[row][column]; **// definition**

- The **new** keyword must be used to **allocate memory** for an array.

OR

Method 2

type array-name={list of values}; **// declaration and definition**

Example

Splitting Declaration and Definition

```
int stud[][];           // declaration  
stud=new int[100][100]; // definition
```

OR

Combining Declaration and Definition

```
int stud[][]=new int[100][100]; // declaration + definition
```


IV. 2D ARRAY : MATRIX ADDITION

(TwoD_Matrix.java)

1. SOURCE CODE

```
public class TwoD_Matrix
{
    // 2D array creation
    int a[][]={{1,1},{1,1}};
    int b[][]={{2,2},{2,2}};
    int c[][]=new int[2][2];
    // print 1st matrix values
    void matrix_a()
    {
        System.out.print("Matrix A : ");
        for(int i=0;i<2;i++)
        {
            for(int j=0;j<2;j++)
            {
                System.out.print(a[i][j]+" ");
            }
        }
        System.out.println();
    }
    // print 2nd matrix values
    void matrix_b()
    {
```

```
System.out.print("Matrix B : ");
for(int i=0;i<2;i++)
{
    for(int j=0;j<2;j++)
    {
        System.out.print(b[i][j]+" ");
    }
}
System.out.println();
}
```

// add the two matrix values

```
public void add_matrix()
{
    for(int i=0;i<a.length;i++)
    {
        for(int j=0;j<b.length;j++)
        {
            c[i][j]=a[i][j]+b[i][j];
        }
    }
}
```

// print the resultant matrix

```
void matrix_sum()
{
    System.out.print("Matrix C : ");
```

```
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            System.out.print(c[i][j]+" ");
        }
    }
    System.out.println();
}
```

// main method

```
public static void main(String[] args)throws Exception
{
    System.out.println("=====");
    System.out.println("\t\t2D Array  Matrix Addition ");
    System.out.println("=====");
}
```

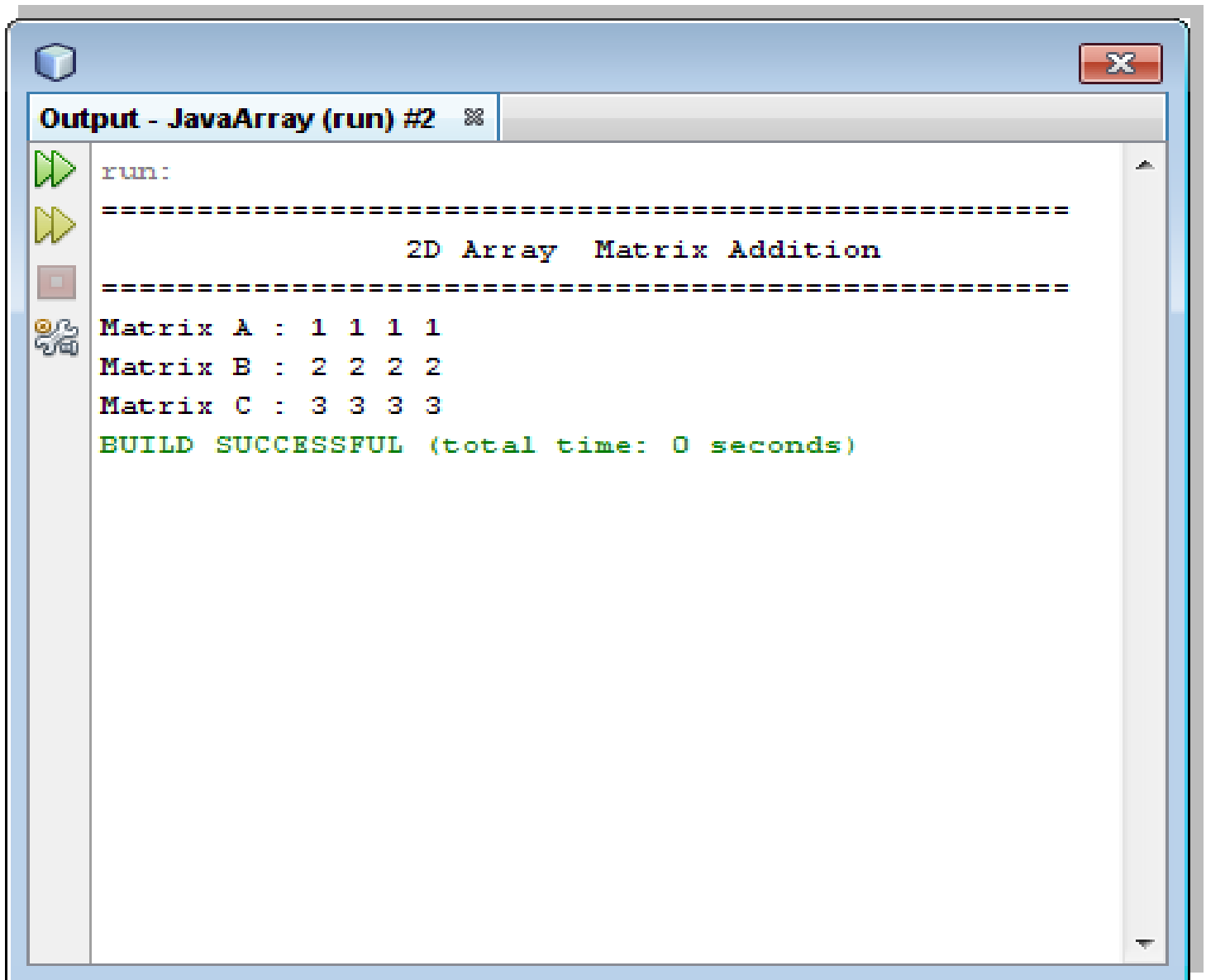
// object cration

```
TwoD_Matrix obj=new TwoD_Matrix();
```

// calling all instance methods

```
    obj.matrix_a();
    obj.matrix_b();
    obj.add_matrix();
    obj.matrix_sum();
}
}
```

2. OUTPUT



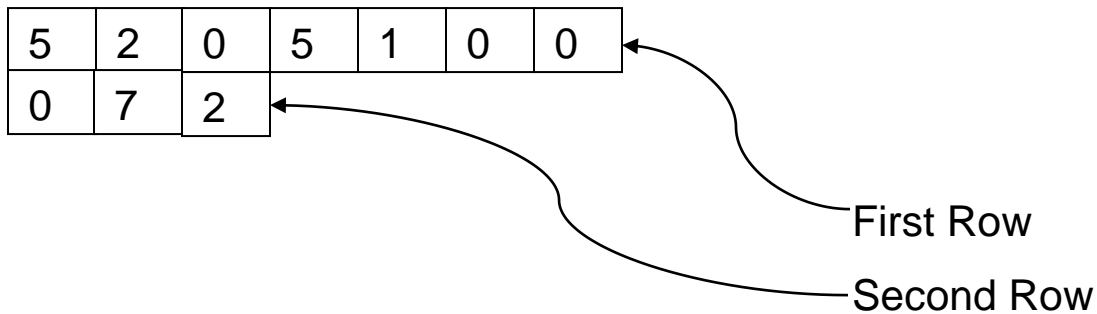
```
run:
=====
                2D Array  Matrix Addition
=====
Matrix A : 1 1 1 1
Matrix B : 2 2 2 2
Matrix C : 3 3 3 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. JAGGED ARRAY (RAGGED ARRAY) / ARRAY OF ARRAYS

- Array of arrays (jagged arrays) is faster than multi-dimensional arrays and can be used more effectively.
- A jagged array is a type of array **whose elements are arrays**. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an "array of arrays."
- It is a new feature supported by Java. In 2D **jagged arrays**, each row, may contain different lengths.

Existing Problem

- Let us take a 2D integer array of size 2 x 7 with following elements



- Here first row **stores 7 elements** and second row **stores only three elements** instead of 7 elements. So that, memory space will be unused (Wastage of storage space)
- In order to **reduce the unwanted memory space** in 2D array, then we can use jagged array.

Example

- Let us design a two-dimensional array with 4 rows where the first row contains 4 elements, the second row with 1 element, the third row with 2 elements and the fourth row with 3 elements.

Following is the schematic representation of the array.

Columns					
0					
1					
2					
3					
s t u d e n t	0	44	55	66	77
	1	36			
	2	87	97		
	3	68	78	88	

Figure : Varying columns 2D array - matrix form

SYNTAX

```
type arrayname[][]=new type[row][];

arrayname[row-0]=new type[col-value-1];
arrayname[row-1]=new type[col-value-2];
arrayname[row-2]=new type[col-value-3];
```

OR

```
type arrayname= {
    {v1,v2, ... vn0}
    {v1,v2, ... vn1}
    {v1,v2, ... vn2}
};
```

List of 1D arrays

Example

```
int users[][]=new int[3][]; // storing three 1D arrays
users[0]=new int[10];      // 1st 1D array with 10 elements
users[1]=new int[5];       // 2nd 1D array with 5 elements
users[2]=new int[3];       // 3rd 1D array with 3 elements
```

23	34	44	55	66	78	89	23	34	99
82	45	12	37	33					
15	26	77							

NOTE

1. To check the total size of array

Syntax

```
arrayname.length;
```

2. To check the size of individual row of array

Syntax

```
arrayname[row_indexposition].length;
```

DIFFERENCE BETWEEN ARRAY & JAGGED ARRAY

S.N	ARRAY	JAGGED ARRAY
1.	Here rows and columns are same size	Here rows and columns are different size
2.	It stores the value types	It stores the 1D array of objects (reference types)
3.	Rectangler array	Special type of array (Jagged Array)

VI. JAGGED ARRAY (RAGGED ARRAY)

(JaggedArray.java)

1. SOURCE CODE

```
public class JaggedArray
{
// jagged array declaration
    int stud[][]=new int[4][];
    public void credisp()
    {
// jagged array definition
        stud[0]=new int[3];
        stud[1]=new int[2];
        stud[2]=new int[1];
        stud[3]=new int[5];
// assign the elements to jagged array
//1st 1D array - initialization
        stud[0][0]=11;
        stud[0][1]=39;
        stud[0][2]=99;
// 2nd 1D array - initialization
        stud[1][0]=54;
        stud[1][1]=86;
// 3rd 1D array - initialization
        stud[2][0]=15;
// 4th 1D array - initialization
        stud[3][0]=10;
        stud[3][1]=20;
        stud[3][2]=30;
        stud[3][3]=45;
```

```
stud[3][4]=75;
System.out.println("Elements in the Jagged Array");
for(int i=0;i<stud.length;i++)
{
    System.out.println("----- Contents of 1D Array " +(i+1)+ " -----");
    for(int j=0;j<stud[i].length;j++)
    {
        System.out.print(stud[i][j]+" ");
    }
    System.out.println();
}
}
```

// main method

```
public static void main(String[] args)throws Exception
{
    System.out.println("=====");
    System.out.println("\t\t2D Jagged Array ");
    System.out.println("=====");
```

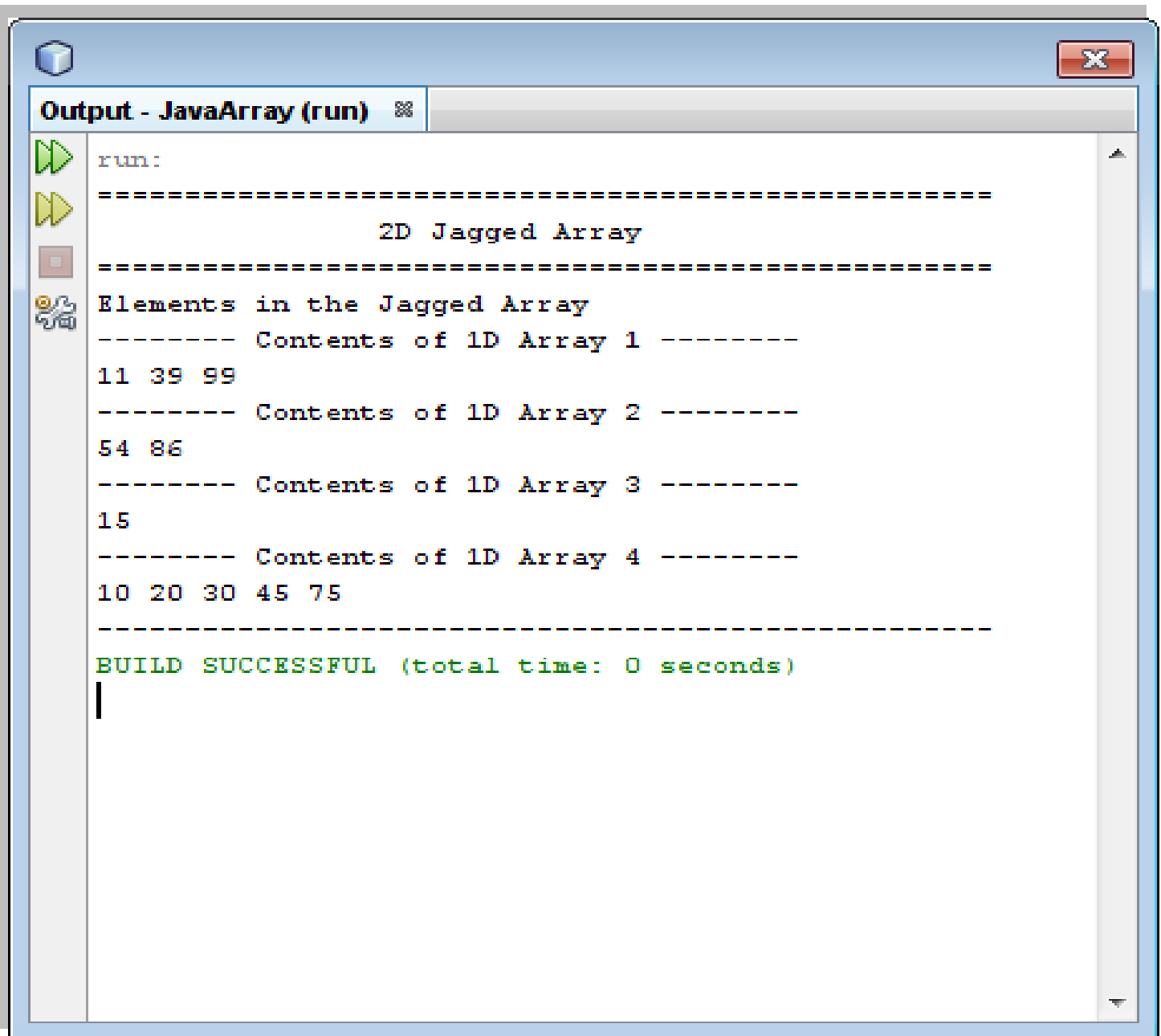
// object creation

```
JaggedArray obj=new JaggedArray();
```

// calling instance method using object

```
obj.credisp();
System.out.println("-----");
}
}
```

2. OUTPUT



```
run:
=====
                2D Jagged Array
=====
Elements in the Jagged Array
----- Contents of 1D Array 1 -----
11 39 99
----- Contents of 1D Array 2 -----
54 86
----- Contents of 1D Array 3 -----
15
----- Contents of 1D Array 4 -----
10 20 30 45 75
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```

V. 2D JAGGED CHAR ARRAY : RETRIEVAL

(TwoD_names.java)

1. SOURCE CODE

```
public class TwoD_names
{
// 2D char array creation
    char names[][]={{'r','a','m'},{'l','a','x','m','a','n'},{'s','h','i','v','a'}};
    void disp()
    {
        System.out.println("----- for loop -----");
        System.out.println("$$$-Names in the 2D char array-$$$");
        System.out.println("Character based Retrieval");
// for-loop: retrieval names using index position (character based)
        for(int i=0;i<names.length;i++)
        {
            for(int j=0;j<names[i].length;j++)
            {
                System.out.print(names[i][j]);
            }
            System.out.println();
        }
        System.out.println("$$$-Names in the 2D char array-$$$");
        System.out.println("String based Retrieval");
// for-loop: retrieval names using String Array index position
        for(int i=0;i<names.length;i++)
        {
            System.out.println(names[i]);
        }
        System.out.println("----- enhanced for loop -----");
```

```
System.out.println("$$$-Names in the 2D char array-$$$");
```

```
System.out.println("String based Retrieval");
```

```
// faster for-loop
```

```
for(char[] s:names)
```

```
{
```

```
    System.out.println(s);
```

```
}
```

```
}
```

```
// main method
```

```
public static void main(String[] args)throws Exception
```

```
{
```

```
    System.out.println("=====");
```

```
    System.out.println("\t\t2D Character Array ");
```

```
    System.out.println("=====");
```

```
// object creation
```

```
    TwoD_names obj=new TwoD_names();
```

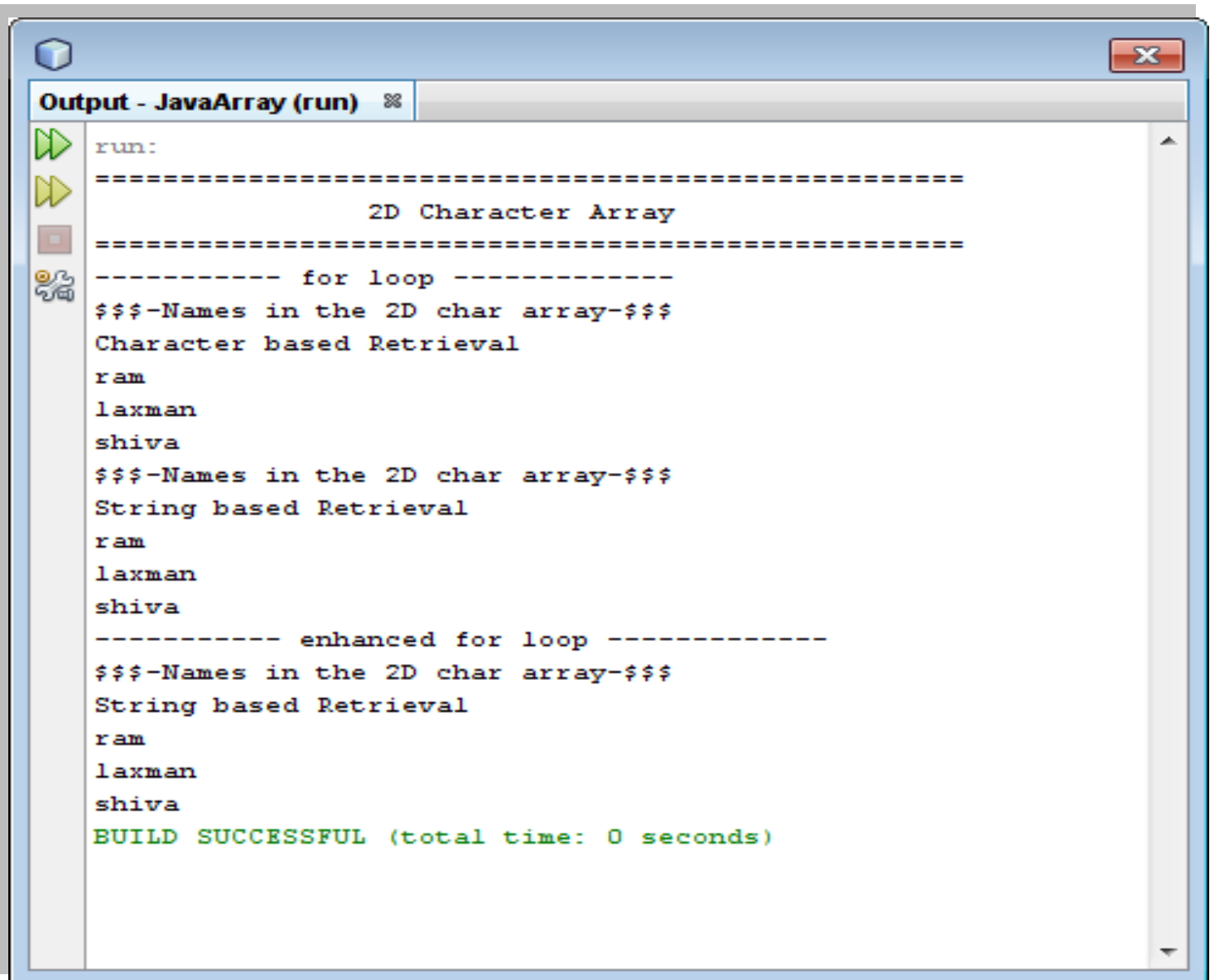
```
// calling instance method using object
```

```
    obj.disp();
```

```
}
```

```
}
```

2. OUTPUT



```
run:
=====
                2D Character Array
=====
----- for loop -----
$$$-Names in the 2D char array-$$$
Character based Retrieval
ram
laxman
shiva
$$$-Names in the 2D char array-$$$
String based Retrieval
ram
laxman
shiva
----- enhanced for loop -----
$$$-Names in the 2D char array-$$$
String based Retrieval
ram
laxman
shiva
BUILD SUCCESSFUL (total time: 0 seconds)
```

USAGE OF 2D INTEGER JAGGED ARRAY

```
public class Test
{
    public static void main(String[] args)
    {
        // jagged array creation
        int ele[][]={{ 5, 2, 0, 5, 1, 0, 0 }, { 0, 7, 2 }};
        System.out.println("Elements in the Jagged Array");
        for(int i=0;i<ele.length;i++)
        {
            for(int j=0;j<ele[i].length;j++)
            {
                System.out.print(ele[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

OUTPUT

Elements in the Jagged Array

5 2 0 5 1 0 0 ← contents of 1st 1D array

0 7 2 ← contents of 2nd 1D array

Advantages

- The main advantage of jagged array is to save the computer memory (reduce the unused array location)
- Because this jagged array usually used for optimisation slots of array.