

# Arrays and Lists

---

Yao, Zhiyang

## Background

---

- Programmer often need the ability to represent a group of values as a list
  - List may be one-dimensional or multidimensional
- Java provides arrays and the collection classes
- Consider arrays first

$$\begin{bmatrix} 1 & 1 & 3 & -2 & 2 \\ 0 & 1 & 4 & 2 & 1 \\ 2 & 3 & 1 & 1 & -3 \\ 3 & 1 & 1 & -1 & -1 \\ 2 & 2 & 1 & 6 & -2 \end{bmatrix}^{-1},$$

## Basic terminology

---

- List is composed of *elements*
- Elements in a list have a *common name*
- The list as a whole is referenced through the common name
- List elements are of the same type — the base type
- Elements of a list are referenced by *subscripting* (indexing) the common name

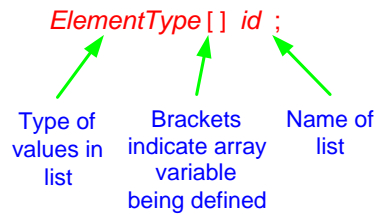
## Java array features

---

- Subscripts are denoted as expressions within brackets: [ ]
- Base (element) type can be any type
- Size of array can be specified at run time
- Index type is integer and the index range must be 0 ... n-1
  - Where n is the number of elements
- Automatic bounds checking
  - Ensures any reference to an array element is valid
- Data field length specifies the number of elements in the list
- Array is an object
  - Has features common to all other objects

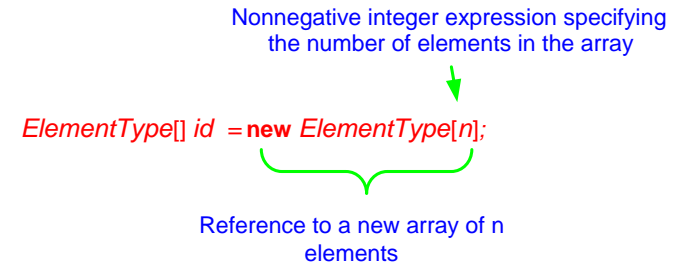
## Array variable definition styles

### Without initialization



## Array variable definition styles

### With initialization



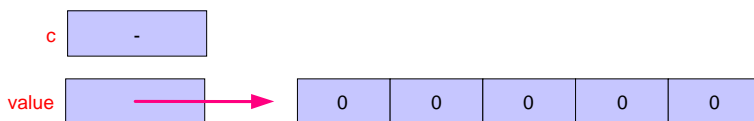
## Example

### Definitions

```
char[] c;  
int[] value = new int[10];
```

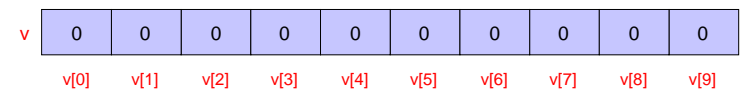
### Causes

- Array object variable `c` is un-initialized
- Array object variable `v` references a new ten element list of integers
  - Each of the integers is default initialized to 0



## Consider

```
int[] v = new int[10];  
int i = 7;  
int j = 2;  
int k = 4;  
v[0] = 1;  
v[i] = 5;  
v[j] = v[i] + 3;  
v[j+1] = v[i] + v[0];  
v[v[j]] = 12;  
System.out.println(v[2]);  
v[k] = stdin.nextInt();
```



## Consider

---

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdIn.nextInt();
```

v	1	0	0	0	0	0	0	0	0	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

---

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdIn.nextInt();
```

v	1	0	0	0	0	0	0	5	0	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

---

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdIn.nextInt();
```

v	1	0	8	0	0	0	0	5	0	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

---

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdIn.nextInt();
```

v	1	0	8	6	0	0	0	5	0	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdIn.nextInt();
```

v	1	0	8	6	0	0	0	5	12	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);      8 is displayed
v[k] = stdIn.nextInt();
```

v	1	0	8	6	0	0	0	5	12	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

```
int[] v = new int[10];
int i = 7;
int j = 2;
int k = 4;
v[0] = 1;
v[i] = 5;
v[j] = v[i] + 3;
v[j+1] = v[i] + v[0];
v[v[j]] = 12;
System.out.println(v[2]);
v[k] = stdIn.nextInt();      Suppose 3 is extracted
```

v	1	0	8	6	3	0	0	5	12	0
	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

## Consider

### □ Segment

```
int[] b = new int[100];
b[-1] = 0;
b[100] = 0;
```

### □ Causes

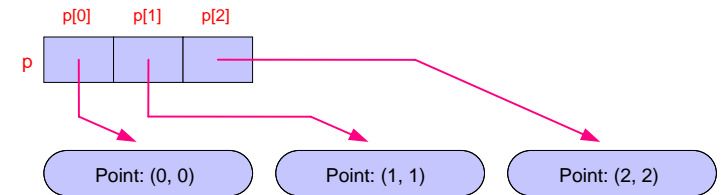
- Array variable to reference a new list of 100 integers
  - Each element is initialized to 0
- Two exceptions to be thrown
  - -1 is not a valid index – too small
  - 100 is not a valid index – too large
    - IndexOutOfBoundsException

## Consider

```
Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
```

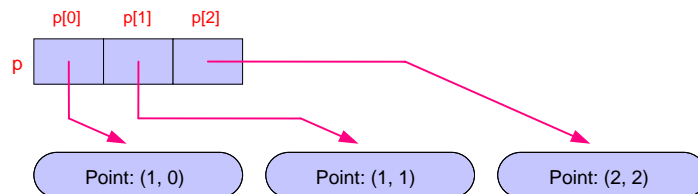
## Consider

```
Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
```



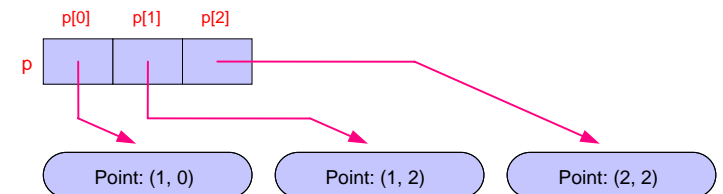
## Consider

```
Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
```



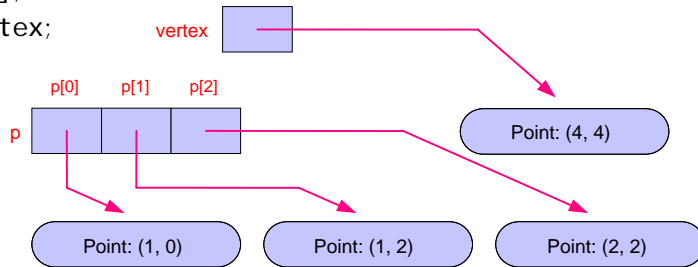
## Consider

```
Point[] p = new Point[3];
p[0] = new Point(0, 0);
p[1] = new Point(1, 1);
p[2] = new Point(2, 2);
p[0].setX(1);
p[1].setY(p[2].getY());
Point vertex = new Point(4, 4);
p[1] = p[0];
p[2] = vertex;
```



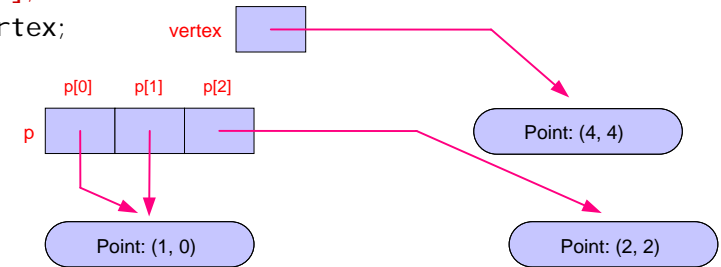
## Consider

```
Point[] p = new Point[3];  
p[0] = new Point(0, 0);  
p[1] = new Point(1, 1);  
p[2] = new Point(2, 2);  
p[0].setX(1);  
p[1].setY(p[2].getY());  
Point vertex = new Point(4, 4);  
p[1] = p[0];  
p[2] = vertex;
```



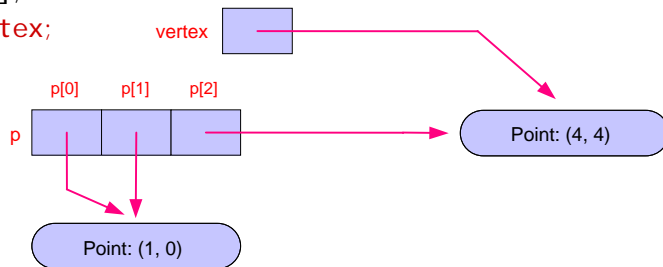
## Consider

```
Point[] p = new Point[3];  
p[0] = new Point(0, 0);  
p[1] = new Point(1, 1);  
p[2] = new Point(2, 2);  
p[0].setX(1);  
p[1].setY(p[2].getY());  
Point vertex = new Point(4, 4);  
p[1] = p[0];  
p[2] = vertex;
```



## Consider

```
Point[] p = new Point[3];  
p[0] = new Point(0, 0);  
p[1] = new Point(1, 1);  
p[2] = new Point(2, 2);  
p[0].setX(1);  
p[1].setY(p[2].getY());  
Point vertex = new Point(4, 4);  
p[1] = p[0];  
p[2] = vertex;
```



## Explicit initialization

### □ Syntax

`id` references an array of  $n$  elements. `id[0]` has value  $exp_0$ , `id[1]` has value  $exp_1$ , and so on.

$ElementType[] id = \{ exp_0, exp_1, \dots, exp_{n-1} \};$

Each  $exp_i$  is an expression that evaluates to type `ElementType`

## Explicit initialization

---

### □ Example

```
String[] puppy = { "nilla", "darby",  
    "galen", "panther" };  
int[] unit = { 1 };
```

### □ Equivalent to

```
String[] puppy = new String[4];  
puppy[0] = "nilla";    puppy[1] = "darby";  
puppy[2] = "galen";    puppy[3] =  
    "panther";  
int[] unit = new int[1];  
unit[0] = 1;
```

## Array members

---

### □ Member length

#### ■ Size of the array

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

## Array members

---

### □ Member clone()

#### ■ Produces a shallow copy

```
Point[] u = { new Point(0, 0), new  
    Point(1, 1)};  
Point[] v = u.clone();
```

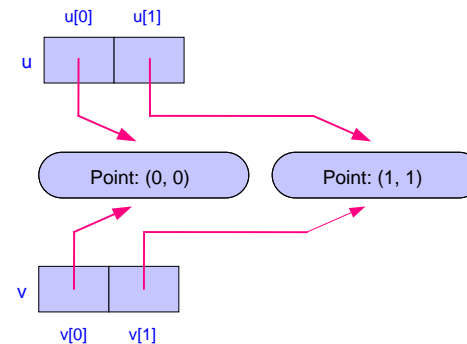
## Array members

---

### □ Member clone()

#### ■ Produces a shallow copy

```
Point[] u = { new Point(0, 0), new  
    Point(1, 1)};  
Point[] v = u.clone();
```



## Array members

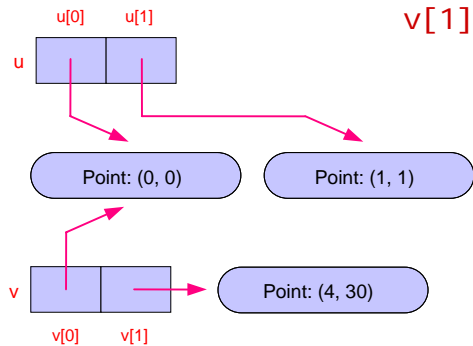
### Member clone()

- Produces a shallow copy

```
Point[] u = { new Point(0, 0), new  
Point(1, 1)};
```

```
Point[] v = u.clone();
```

```
v[1] = new Point(4, 30);
```



## Array members

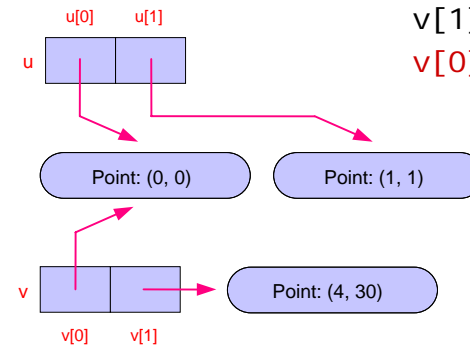
### Member clone()

- Produces a shallow copy

```
Point[] u = { new Point(0, 0), new  
Point(1, 1)};
```

```
Point[] v = u.clone();
```

```
v[1] = new Point(4, 30);  
v[0].setX(3); //what will  
// happen then?
```

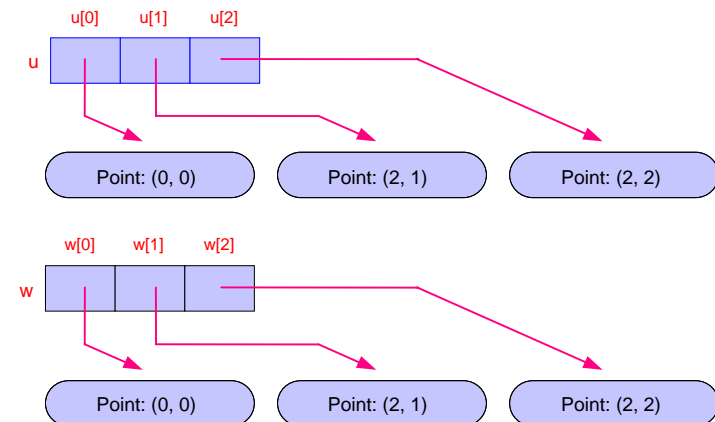


## Making a deep copy

### Example

```
Point[] w = new Point[u.length];  
for (int i = 0; i < u.length; ++i) {  
    w[i] = u[i].clone();  
}
```

## Making a deep copy





## ArrayTools.java method binarySearch()

```
public static int binarySearch(char[] data, char key) {
    int left = 0;
    int right = data.length - 1;
    while (left <= right) {
        int mid = (left + right)/2;
        if (data[mid] == key) {
            return mid;
        }
        else if (data[mid] < key) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}
```

Algorithm Tips:  
Sequential:  $O(N)$   
Binary:  $O(\log N)$

## Sorting

### □ Problem

- Arranging elements so that they are ordered according to some desired scheme
  - Standard is non-decreasing order
    - Why don't we say increasing order?

### □ Major tasks

- Comparisons of elements
- Updates or element movement

## Selection sorting

### □ Algorithm basis

- On iteration  $i$ , a selection sorting method
  - Finds the element containing the  $i$ th smallest value of its list  $v$  and exchanges that element with  $v[i]$

### □ Example – iteration 0

- Swaps smallest element with  $v[0]$
- This results in smallest element being in the correct place for a sorted result

	0	1	2	3	4	5	6	7	8	9
v	'E'	'W'	'Q'	'R'	'T'	'Y'	'U'	'I'	'O'	'P'

## Selection sorting

### □ Algorithm basis

- On iteration  $i$ , a selection sorting method
  - Finds the element containing the  $i$ th smallest value of its list  $v$  and exchanges that element with  $v[i]$

### □ Example – iteration 0

- Swaps smallest element with  $v[0]$
- This results in smallest element being in the correct place for a sorted result

	0	1	2	3	4	5	6	7	8	9
v	'E'	'W'	'Q'	'R'	'T'	'Y'	'U'	'I'	'O'	'P'

A pink double-headed arrow points from index 0 to index 2, indicating the swap of the smallest element 'E' with the element 'Q' at index 2.

## Selection sorting

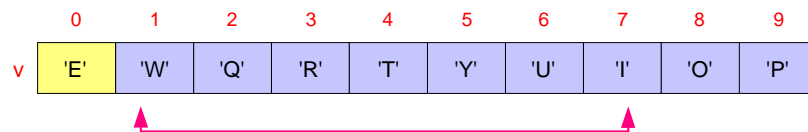
---

### □ Algorithm basis

- On iteration  $i$ , a selection sorting method
  - Finds the element containing the  $i$ th smallest value of its list  $v$  and exchanges that element with  $v[i]$

### □ Example – iteration 1

- Swaps second smallest element with  $v[1]$
- This results in second smallest element being in the correct place for a sorted result



## Selection sorting

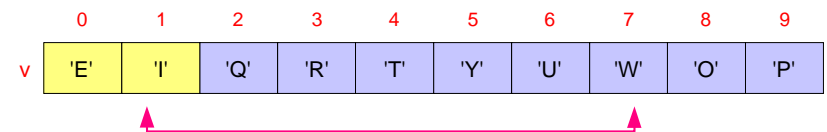
---

### □ Algorithm basis

- On iteration  $i$ , a selection sorting method
  - Finds the element containing the  $i$ th smallest value of its list  $v$  and exchanges that element with  $v[i]$

### □ Example – iteration 1

- Swaps second smallest element with  $v[1]$
- This results in second smallest element being in the correct place for a sorted result



## ArrayTools.java selection sorting

---

```
public static void selectionSort(char[] v) {
    for (int i = 0; i < v.length-1; ++i) {
        // guess the location of the i th smallest element
        int guess = i;
        for (int j = i+1; j < v.length; ++j) {
            if (v[j] < v[guess]) { // is guess ok?
                // update guess to index of smaller element
                guess = j;
            }
        }

        // guess is now correct, so swap elements
        char rmbv = v[i];
        v[i] = v[guess];
        v[guess] = rmbv;
    }
}
```

## Iteration i

---

```
// guess the location of the i th smallest element
int guess = i;
for (int j = i+1; j < v.length; ++j) {
    if (v[j] < v[guess]) // is guess ok?
        // update guess with index of smaller element
        guess = j;
}

// guess is now correct, swap elements v[guess] and
v[0]
```

# Multidimensional arrays

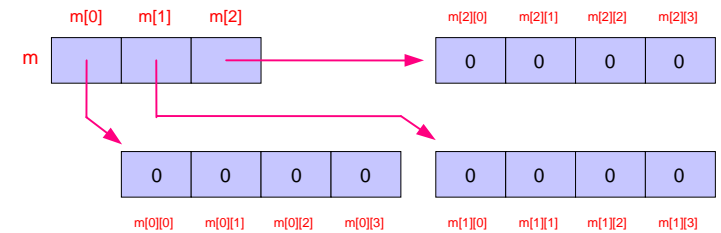
- Many problems require information be organized as a two-dimensional or multidimensional list
- Examples
  - Matrices
  - Graphical animation
  - Economic forecast models
  - Map representation
  - Time studies of population change
  - Microprocessor design

# Example

- Segment

```
int[][] m = new int[3][];  
m[0] = new int[4];  
m[1] = new int[4];  
m[2] = new int[4];
```

- Produces

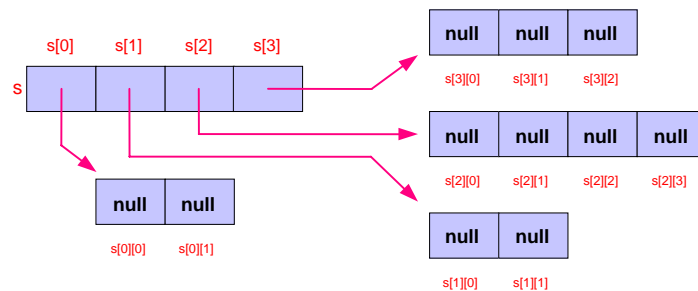


# Example

- Segment

```
String[][] s = new String[4][];  
s[0] = new String[2];  
s[1] = new String[2];  
s[2] = new String[4];  
s[3] = new String[3];
```

- Produces

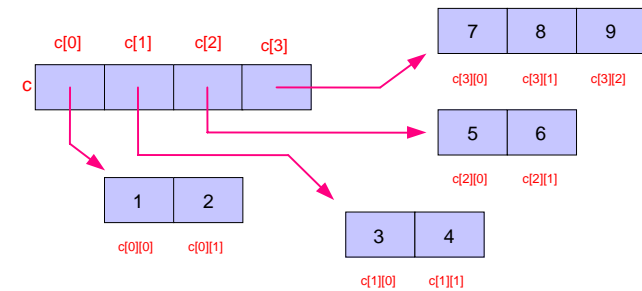


# Example

- Segment

```
int c[][] = {{1, 2}, {3, 4}, {5, 6}, {7, 8, 9}};
```

- Produces



## Matrices

---

- A two-dimensional array is sometimes known as a matrix because it resembles that mathematical concept
- A matrix  $a$  with  $m$  rows and  $n$  columns is represented mathematically in the following manner

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & & & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

## Matrix addition

---

- Definition  $C = A + B$ 
  - $c_{ij} = a_{1i}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$
  - $c_{ij}$  is sum of terms produced by multiplying the elements of  $a$ 's row  $i$  with  $b$ 's column  $c$

## Matrix addition

---

```
public static double[][] add(double[][] a, double[][]
    b) {
    // determine number of rows in solution
    int m = a.length;

    // determine number of columns in solution
    int n = a[0].length;

    // create the array to hold the sum
    double[][] c = new double[m][n];

    // compute the matrix sum row by row
    for (int i = 0; i < m; ++i) {
        // produce the current row
        for (int j = 0; j < n; ++j) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }

    return c;
}
```

## ArrayList

---

- Although arrays in Java are more robust, but still they cannot be resized, that is there are no Java array operations that support the insertion of a new elements or the deletion of existing elements.
- ArrayList
  - **java.util.ArrayList** allows for expandable arrays, and is the Collections replacement for the older Vector class. An ArrayList has the following advantages over an array:
  - An ArrayList automatically expands as data is added.
  - Access to any element of an ArrayList is  $O(1)$ . Insertions and deletions are  $O(N)$ .
  - An ArrayList has methods for inserting, deleting, and searching.
  - An ArrayList can be traversed using either iterators or indexes.
  - **Automatic expansion.**
  - **Objects only.**