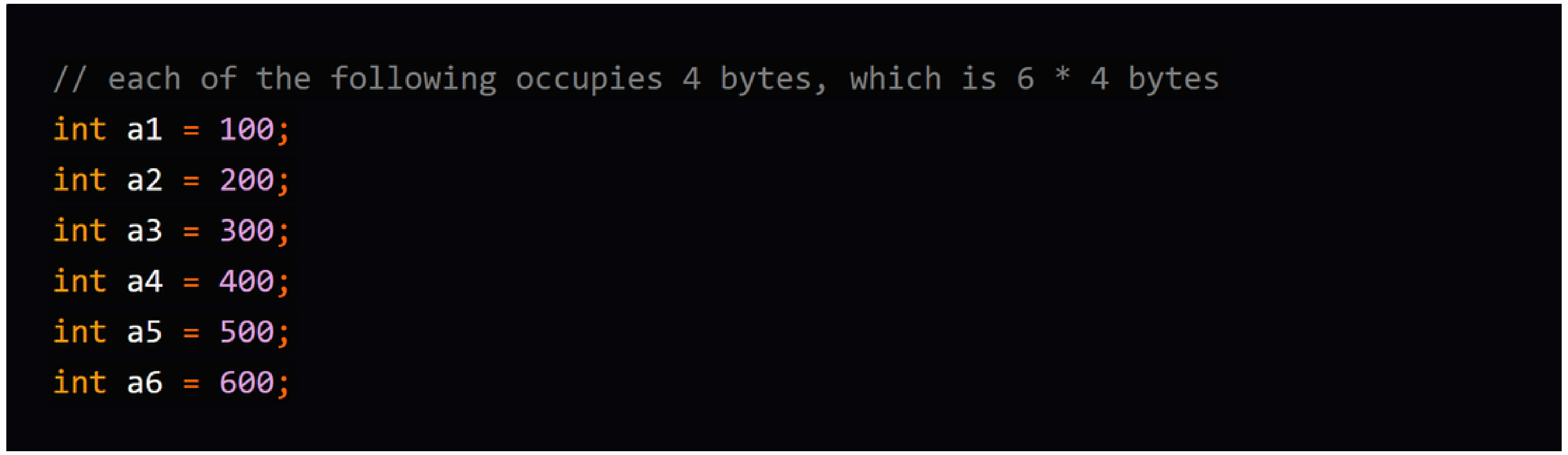
# Data Structures

# Sec 3 : Array & Linked List

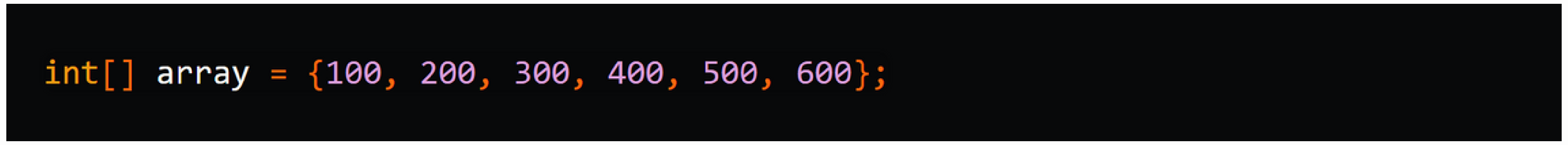
By: Eng.Rahma Osama Eng.Sandra Sameh

# Array As A Data Structure

- An Array is a data structure that stores a **fixed-size** sequence of elements of **the same type**.
- It allows you to store **multiple values in a single variable** instead of declaring separate variables for each value.
- Each element can be **accessed directly** by its index.
- **Indexing** in Java arrays **starts from 0**.



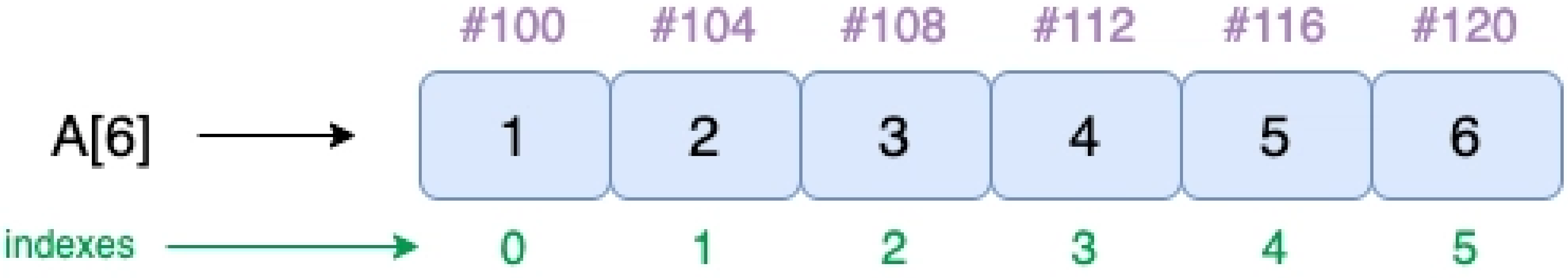
```
// each of the following occupies 4 bytes, which is 6 * 4 bytes
int a1 = 100;
int a2 = 200;
int a3 = 300;
int a4 = 400;
int a5 = 500;
int a6 = 600;
```



```
int[] array = {100, 200, 300, 400, 500, 600};
```

# Characteristics Of Array

- **Homogeneous Elements:** All elements in an array must be of the same data type (e.g., all int, all String, etc.).
- **Store Primitives and Objects:** Java arrays can hold both primitive types (like int, char, boolean) and object references (like String, Integer, etc.).
- **Contiguous Memory Allocation:** For primitive types, elements are stored in contiguous memory locations , For non-primitive types, references to objects are stored contiguously.
- **Fixed Size:** Once an array is created, its size cannot be changed “static data structure”.
- **Zero-Based Indexing:** The first element is at index 0, and the last element is at length - 1.
- **Fast Access (O(1)):** Each element can be directly accessed using its index in constant time.
- **Arrays are Objects:** In Java, arrays are objects, not primitive types, and are created dynamically using new. “Static Structure but Dynamic Memory Allocation”



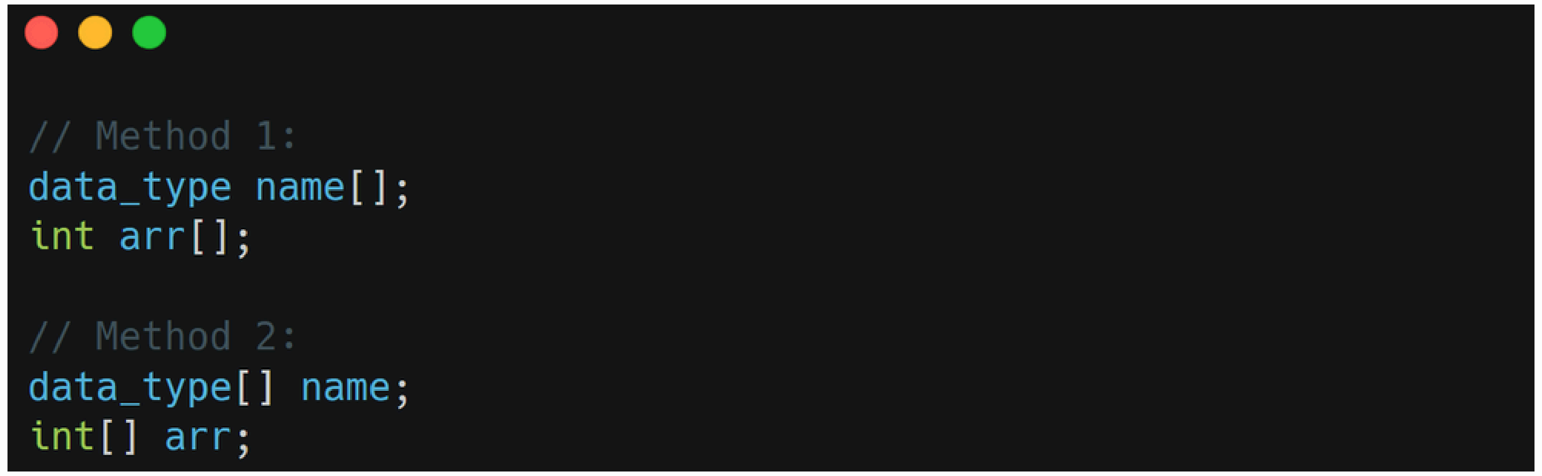
| | | | | | | |
|---|---|---|---|---|---|---|
| | #100 | #104 | #108 | #112 | #116 | #120 |
| A[6] → | 1 | 2 | 3 | 4 | 5 | 6 |
| indexes → | 0 | 1 | 2 | 3 | 4 | 5 |

Integer in Java takes 4 bytes

Array above occupies the contiguous memory from #100 to #120

# Declaration and initialization

### The general form of array declaration is :

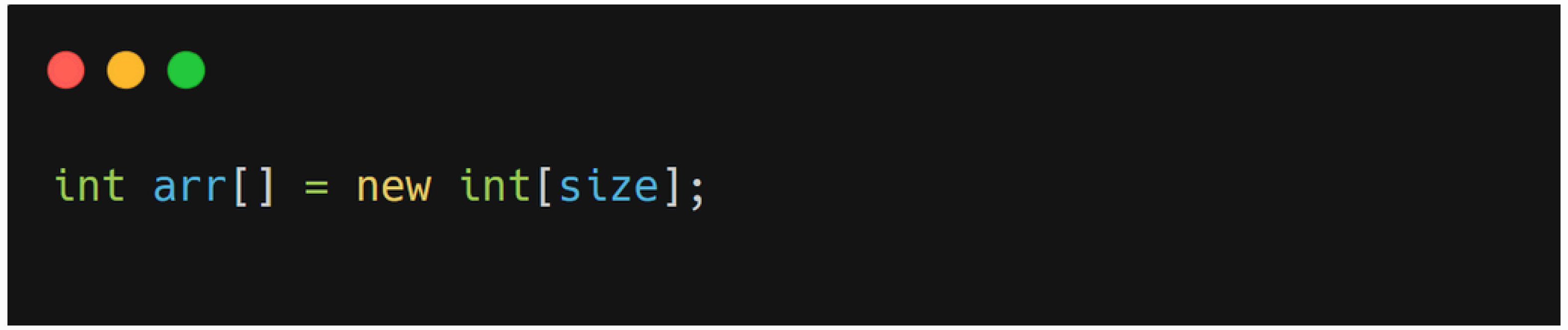


```
// Method 1:
data_type name[];
int arr[];

// Method 2:
data_type[] name;
int[] arr;
```

### Initialization an Array in Java:

When an array is declared, only a reference of an array is created.

We use new to allocate an array of given size.



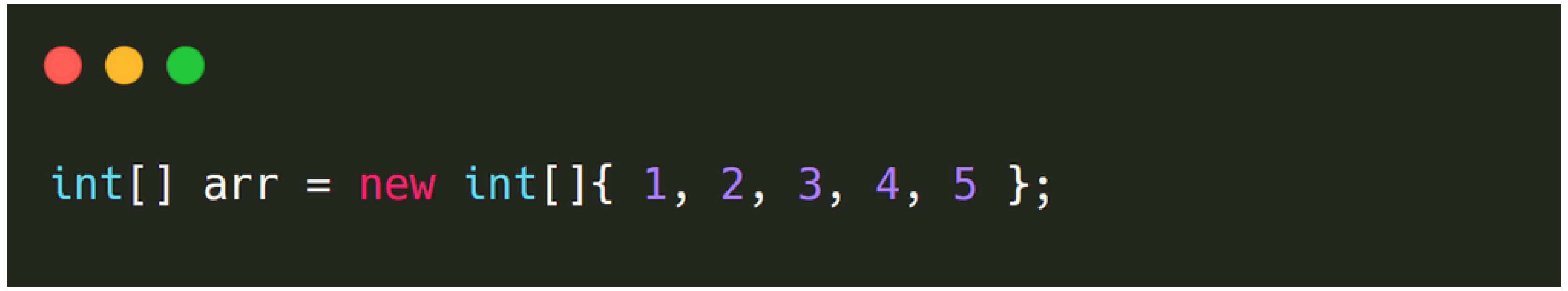
```
int arr[] = new int[size];
```

**Note:** It is just how we can create is an array variable, no actual array exists. It merely tells the compiler that this variable (int Array) will hold an array of the integer type

- The elements in the array allocated by new will **automatically be initialized** to zero (for numeric types), false (for boolean) or null (for reference types).

# Declaration and initialization

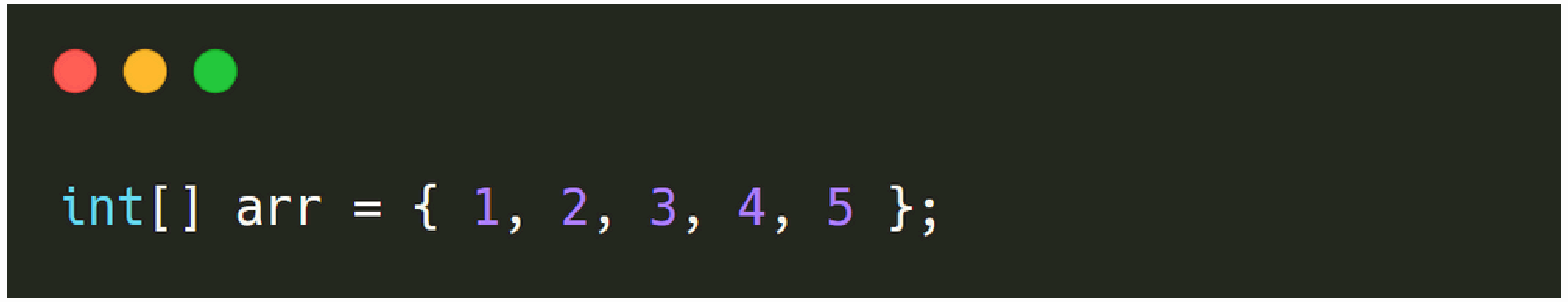
**Array Literal in Java:**



```java
int[] arr = new int[]{ 1, 2, 3, 4, 5 };
```

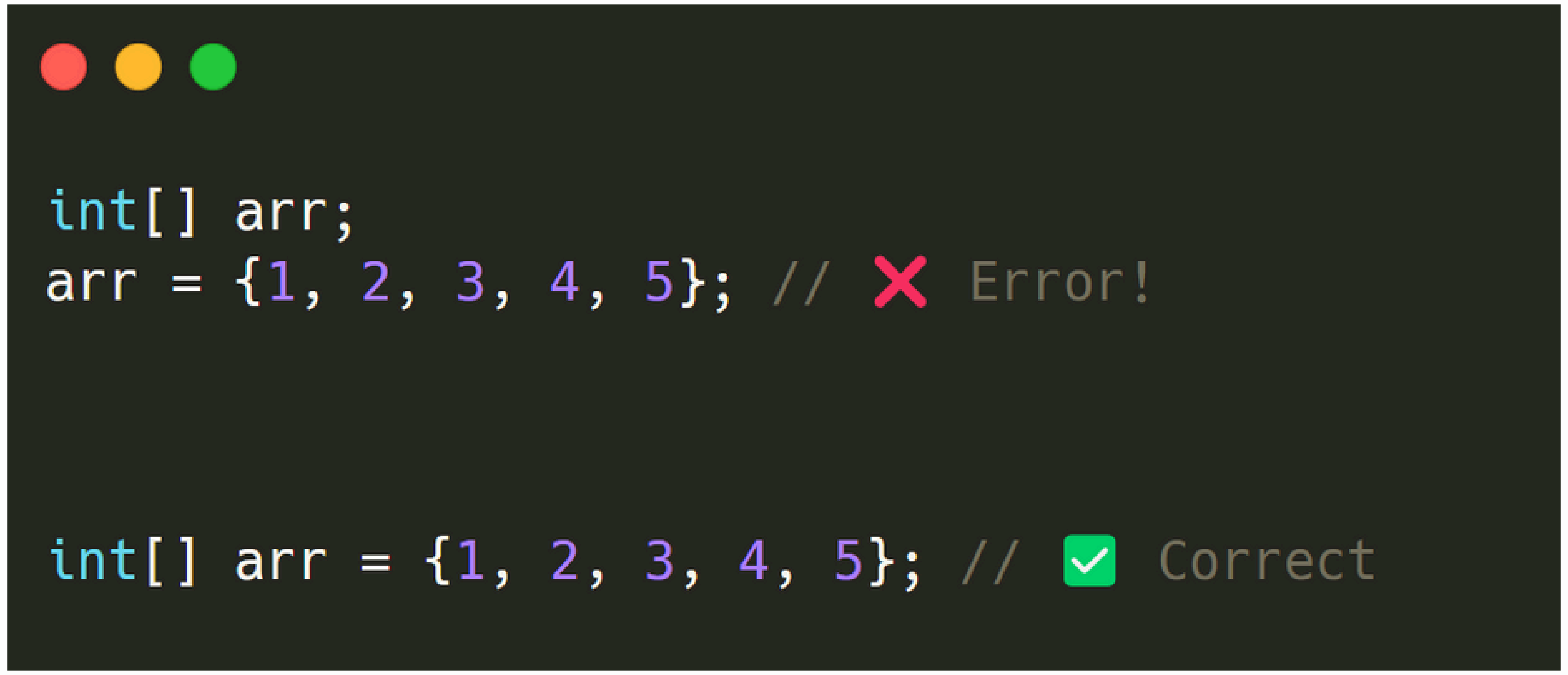
In a situation where the size of the array and variables of the array are already known, array literals can be used.

**In modern Java versions, you can omit new int[] if initialization is done at the same time as declaration:**



```java
int[] arr = { 1, 2, 3, 4, 5 };
```

- **Array literals cannot be used if the declaration and initialization are done separately.**



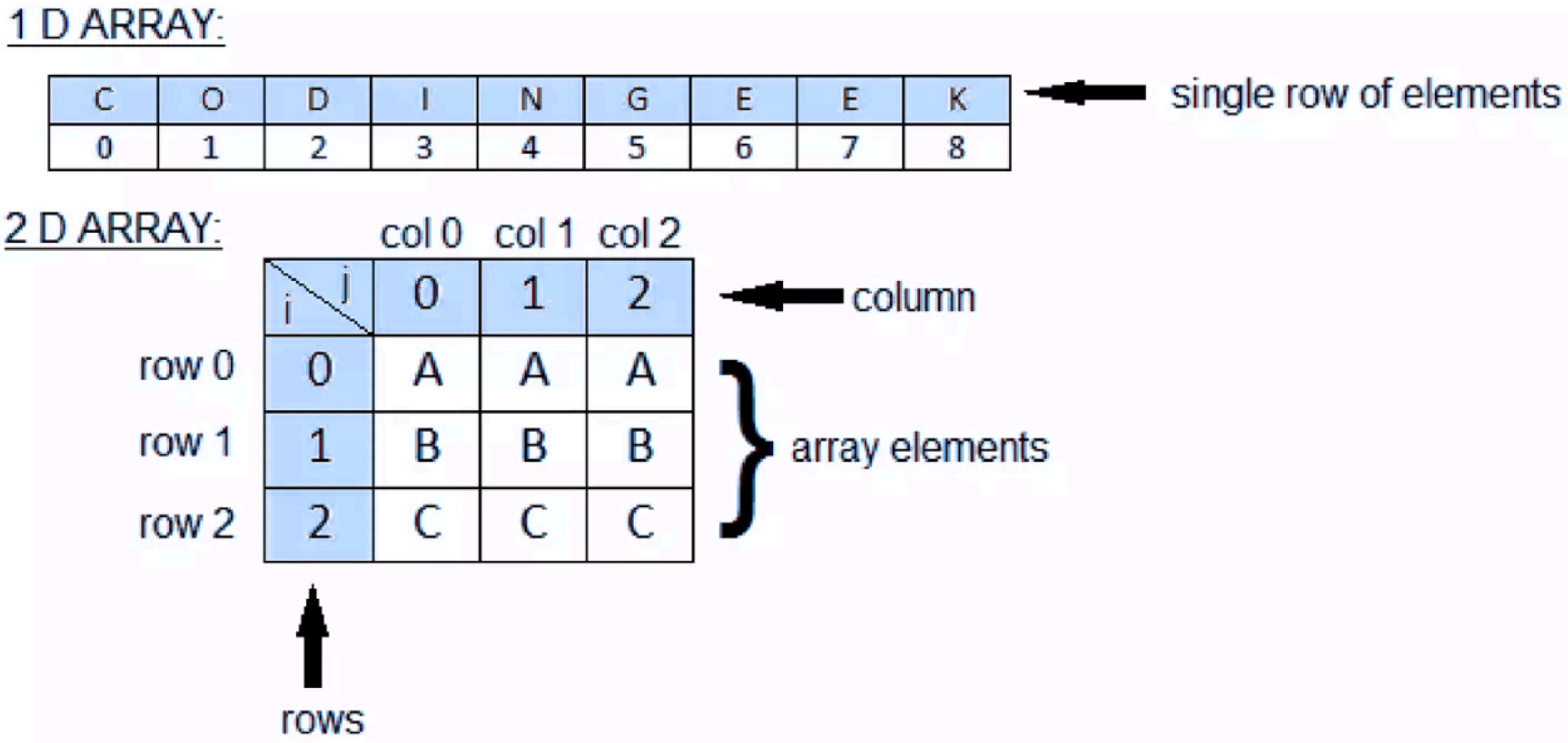
```java
int[] arr;
arr = {1, 2, 3, 4, 5}; // ❌ Error!


int[] arr = {1, 2, 3, 4, 5}; // ✅ Correct
```

# Types of Array in java :

There are two types of array:

*** Single Dimensional Array.**

*** Multidimensional Array.**



1 D ARRAY:

| C | O | D | I | N | G | E | E | K |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

single row of elements

2 D ARRAY:

| | | col 0 | col 1 | col 2 |
|---|---|---|---|---|
| | i \ j | 0 | 1 | 2 |
| row 0 | 0 | A | A | A |
| row 1 | 1 | B | B | B |
| row 2 | 2 | C | C | C |

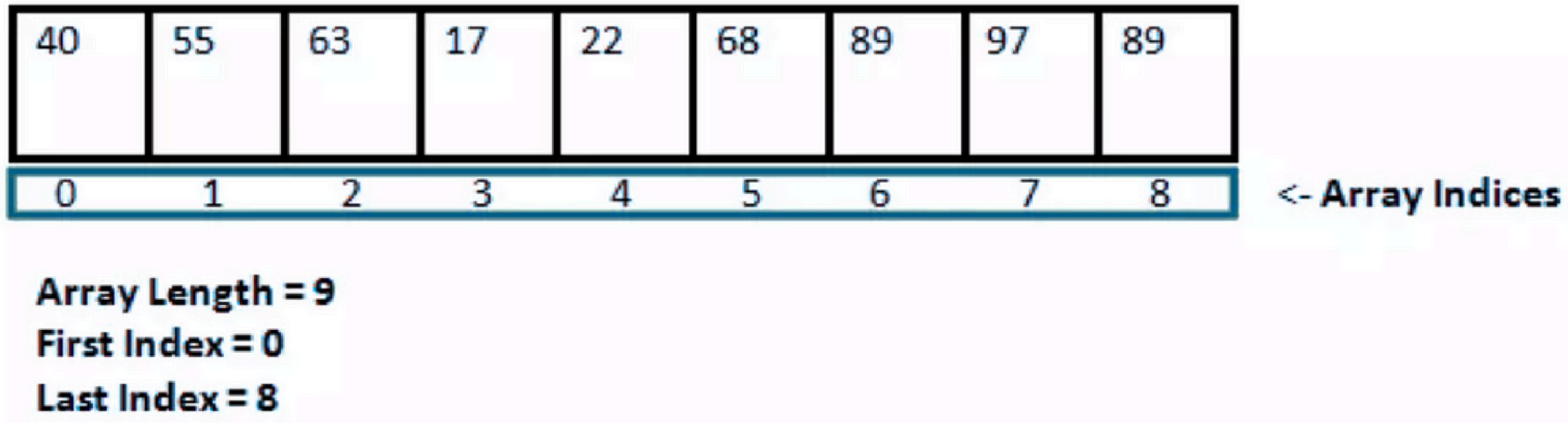
column

array elements

rows

# 1.One-Dimensional Array in Java (1-D) :

One-dimensional array(1-D): It is a single array that holds multiple values of the same type.



| 40 | 55 | 63 | 17 | 22 | 68 | 89 | 97 | 89 |  |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | <- Array Indices |

Array Length = 9
First Index = 0
Last Index = 8

## Declaring 1D Array :

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array.

## Syntax to declare an array in java will be like one of them :

```
dataType[ ]  arr ;  // preferred way.
dataType  [ ]arr ;
dataType  arr[ ] ;
```

# Multidimensional Array in Java (2D):

In such case, data is stored in row and column based index (matrix form).

## Syntax to Declare 2D Array in Java :

```
dataType[ ][ ]   arr ; (or)

dataType   [ ][ ]arr ; (or)

dataType   arr[ ][ ]; (or)

dataType   [ ]arr[ ];
```

Columns

Rows

| 2D ARRAY | 0 | 1 | 2 |
|---|---|---|---|
| 0 | a[0][0] | a[0][1] | a[0][2] |
| 1 | a[1][0] | a[1][1] | a[1][2] |
| 2 | a[2][0] | a[2][1] | a[2][2] |

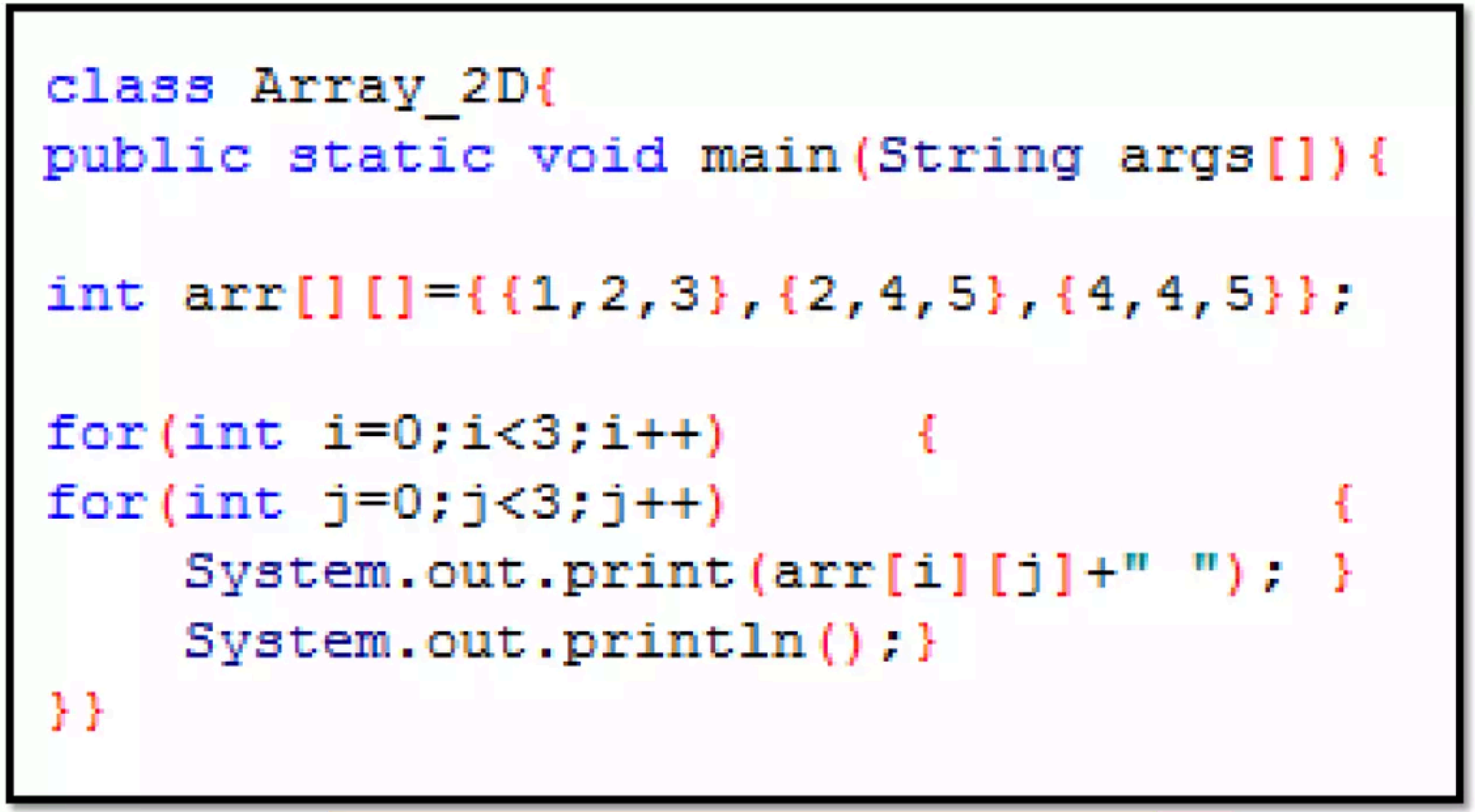
## Example to instantiate Multidimensional Array in Java:

```
int[ ][ ]   arr=new int[3][3];    //3 row and 3 column
```

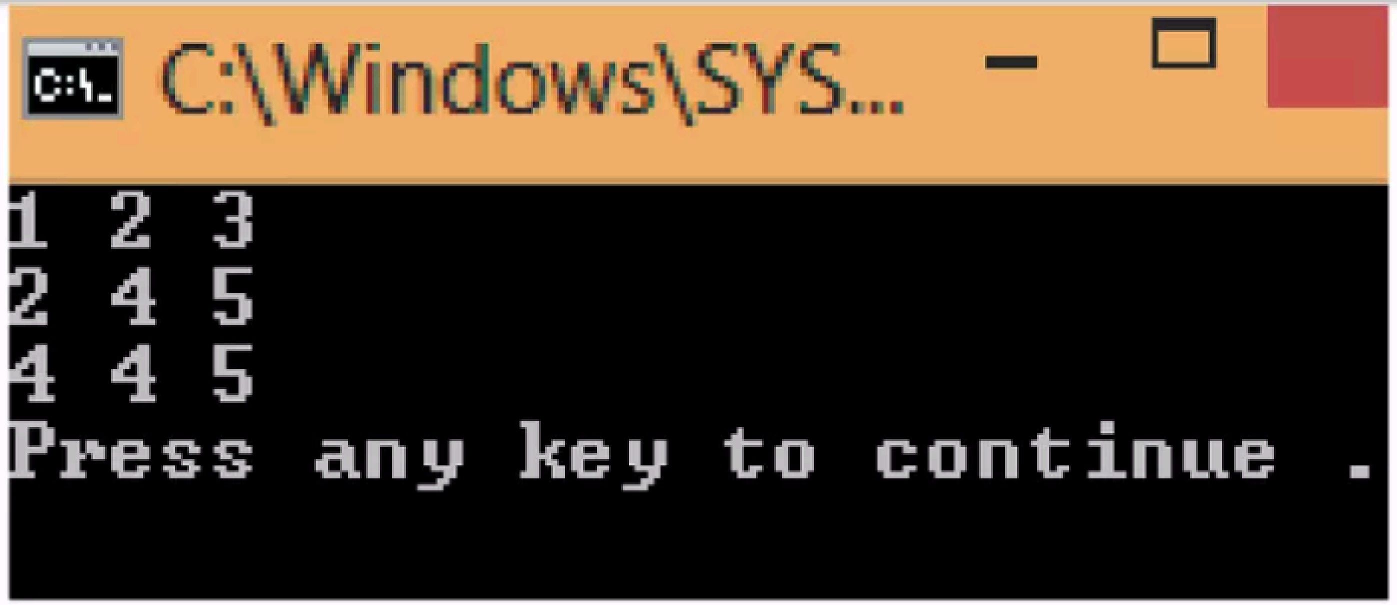
## Example to initialize Multidimensional Array in Java :

```
arr[0][0]=1;  arr[0][1]=2;  arr[0][2]=3;
arr[1][0]=4;  arr[1][1]=5;  arr[1][2]=6;
arr[2][0]=7;  arr[2][1]=8;  arr[2][2]=9;
```

# Example of Multidimensional (2D) Java Array :



```
class Array_2D{
public static void main(String args[]){

int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

for(int i=0;i<3;i++)      {
for(int j=0;j<3;j++)              {
    System.out.print(arr[i][j]+" "); }
    System.out.println();}
}}
```



```
1 2 3
2 4 5
4 4 5
Press any key to continue .
```

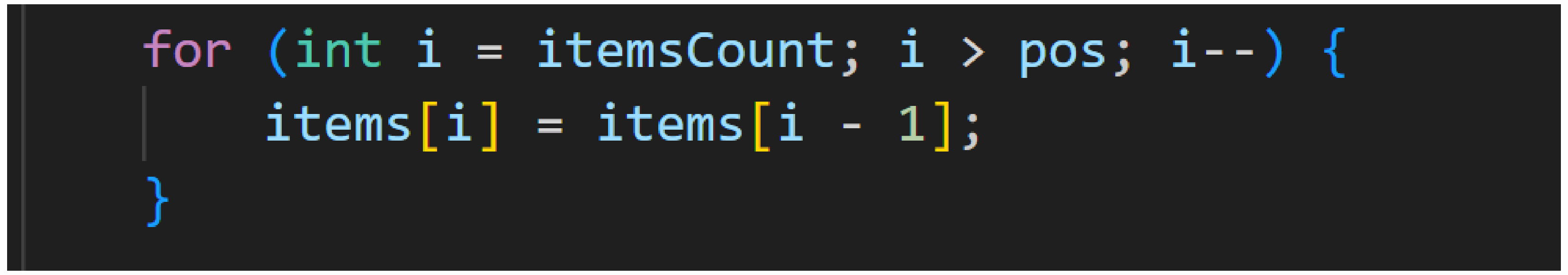
# Operations on arrays :

- **access** item **O(1).**
- **Traverse** Items.
- **append** item (insert at last) : append(dataType item).
- **Search** Specific item : search(dataTye item).
- **insert** at specific position : insert(int position , dataType item).
- **Delete** last item: delete(dataType item).
- **Delete** item at specific location : delete(int pos , dataType item).

Time Coding

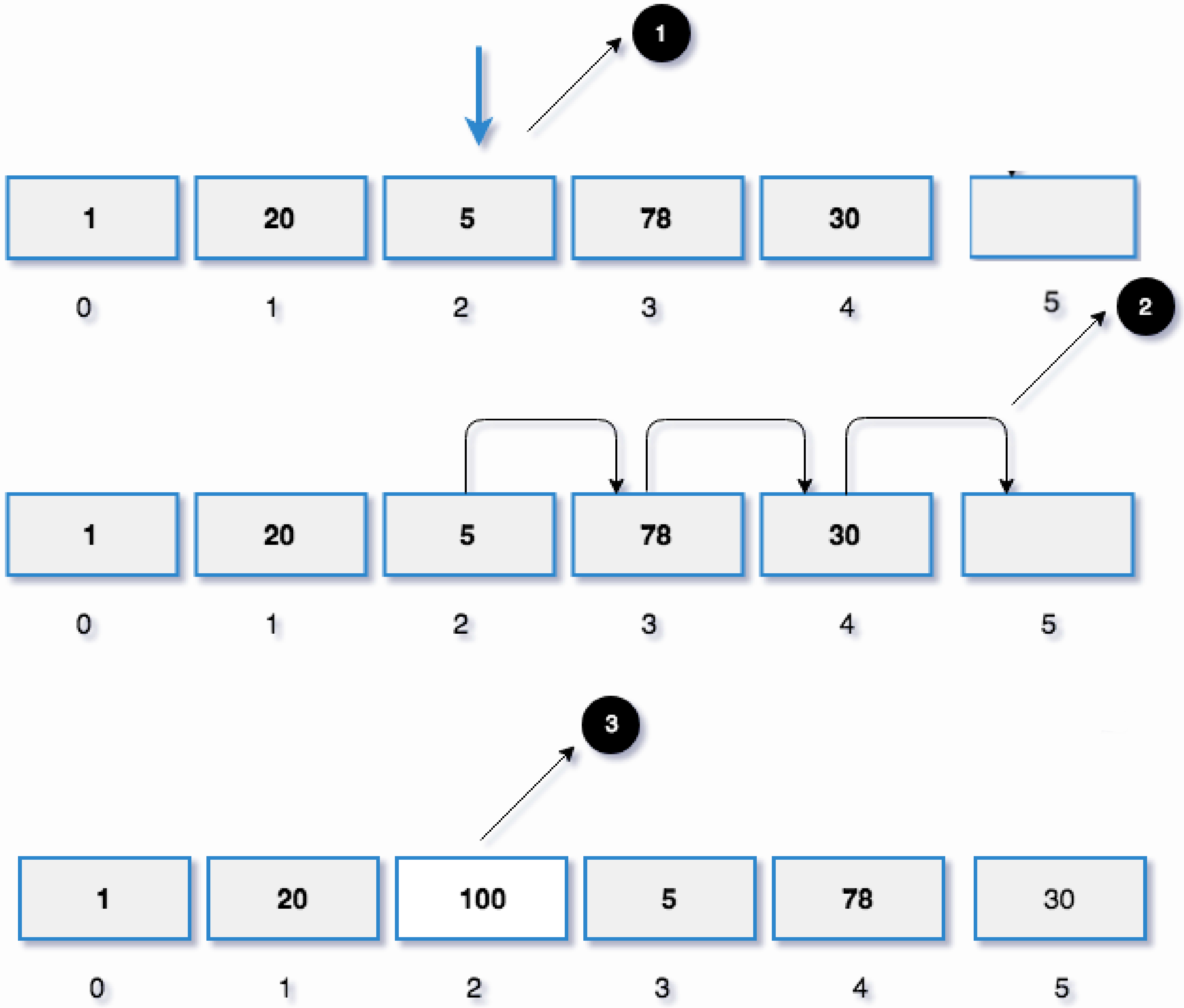
# Insert at a specific location :

- Check if there’s an empty location(s).
- Move right items a step : for all items in range index+1 to last item items[i]=items[i-1].



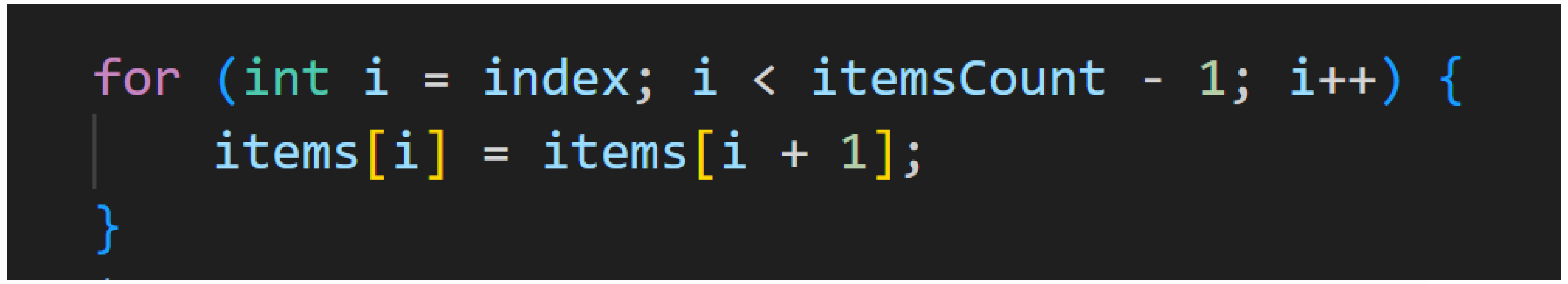
```
for (int i = itemsCount; i > pos; i--) {
    items[i] = items[i - 1];
}
```

- Now you can insert the new item at specific location.

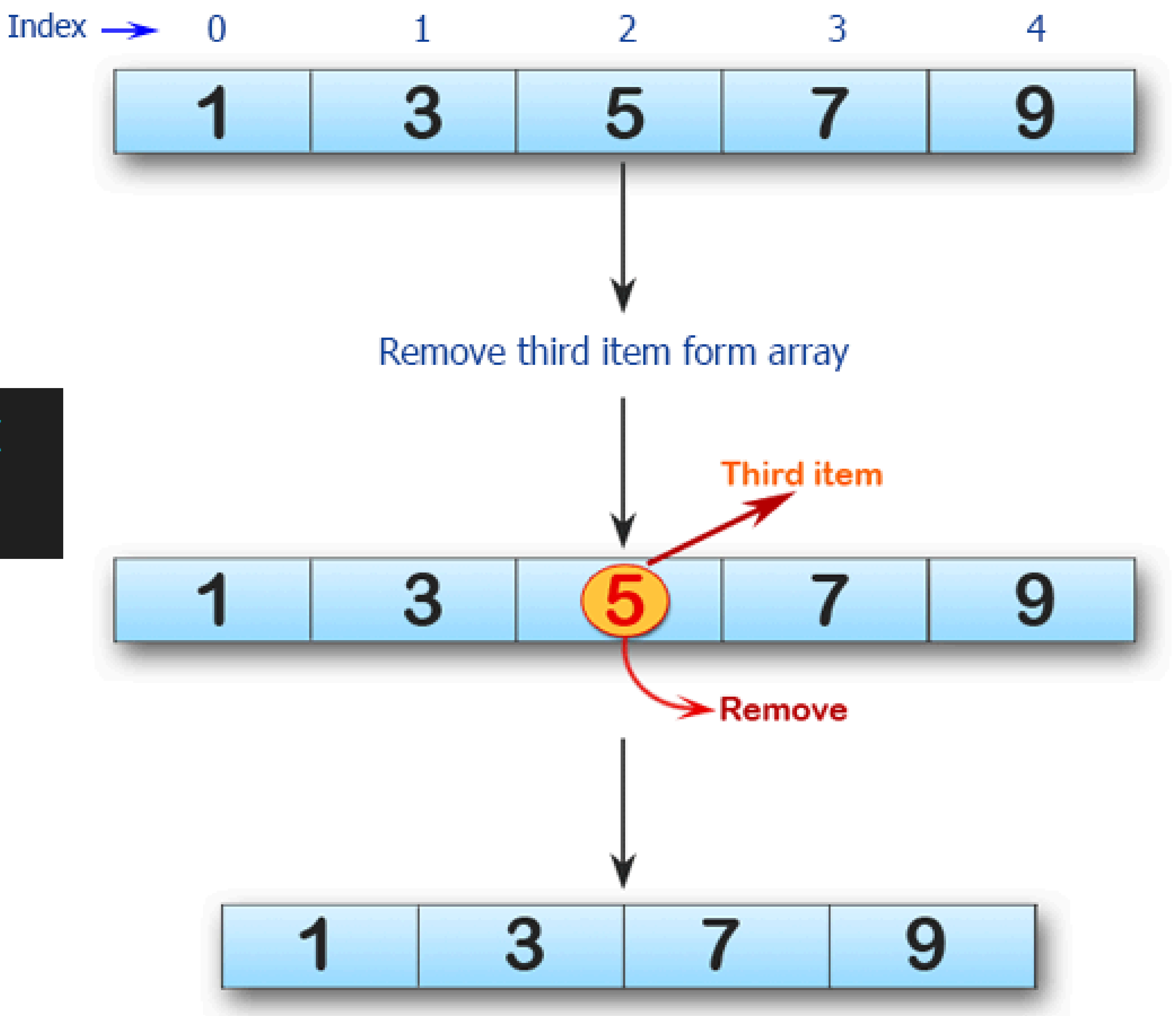


# Delete from specific location :

- Shift right side items:

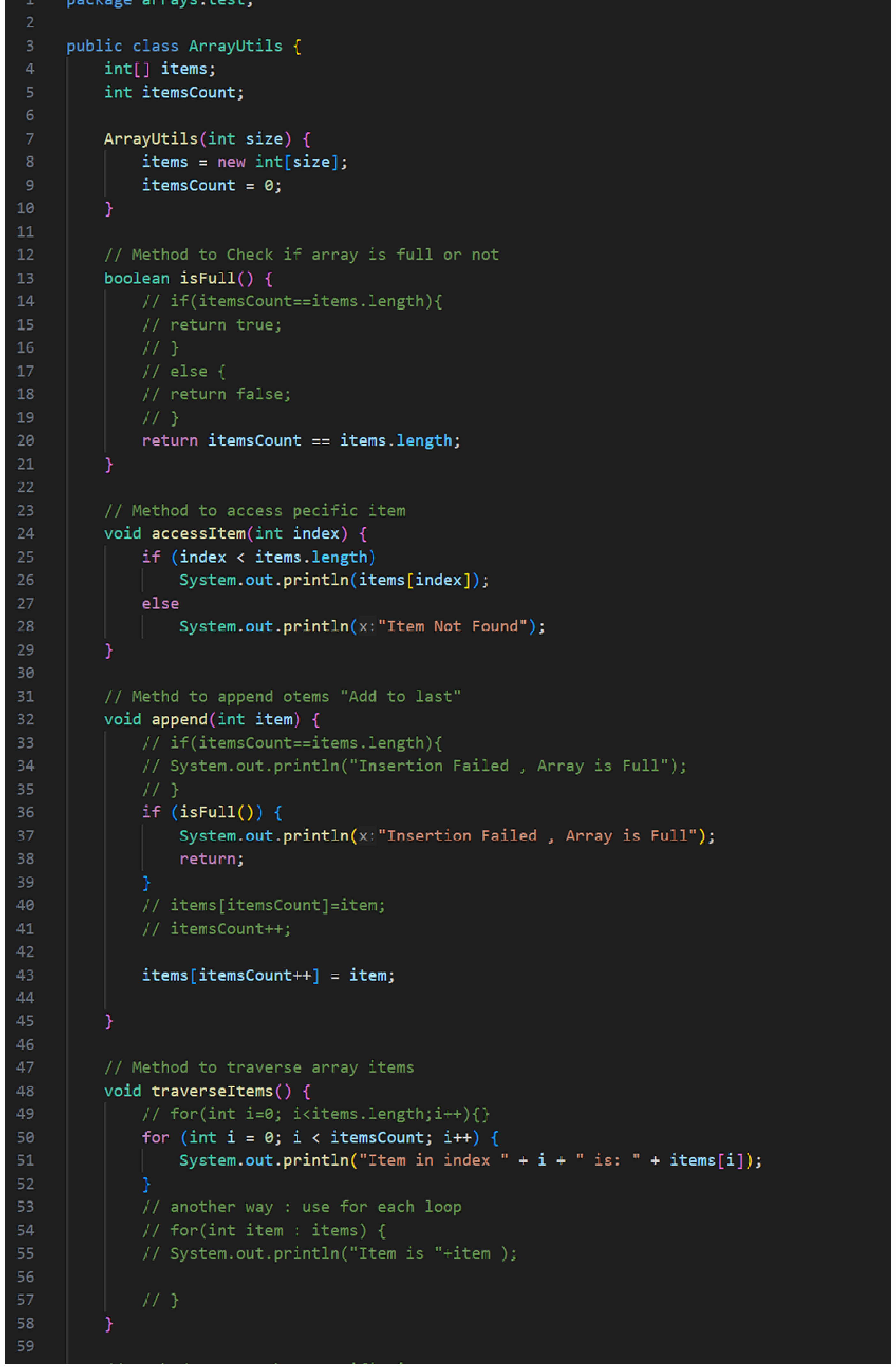


```
for (int i = index; i < itemsCount - 1; i++) {
    items[i] = items[i + 1];
}
```

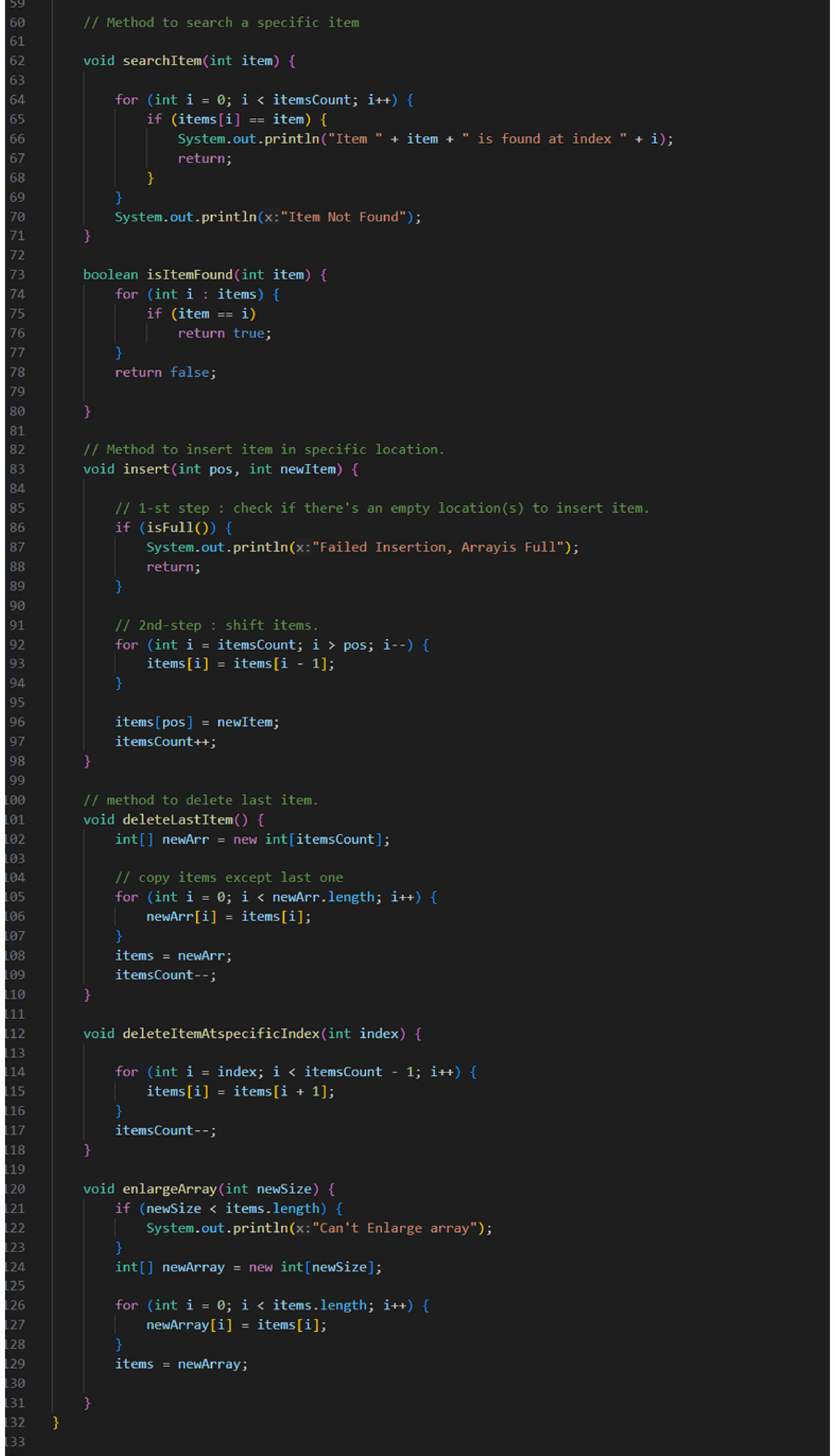


How to increase an array size?

# Array Functions Coding

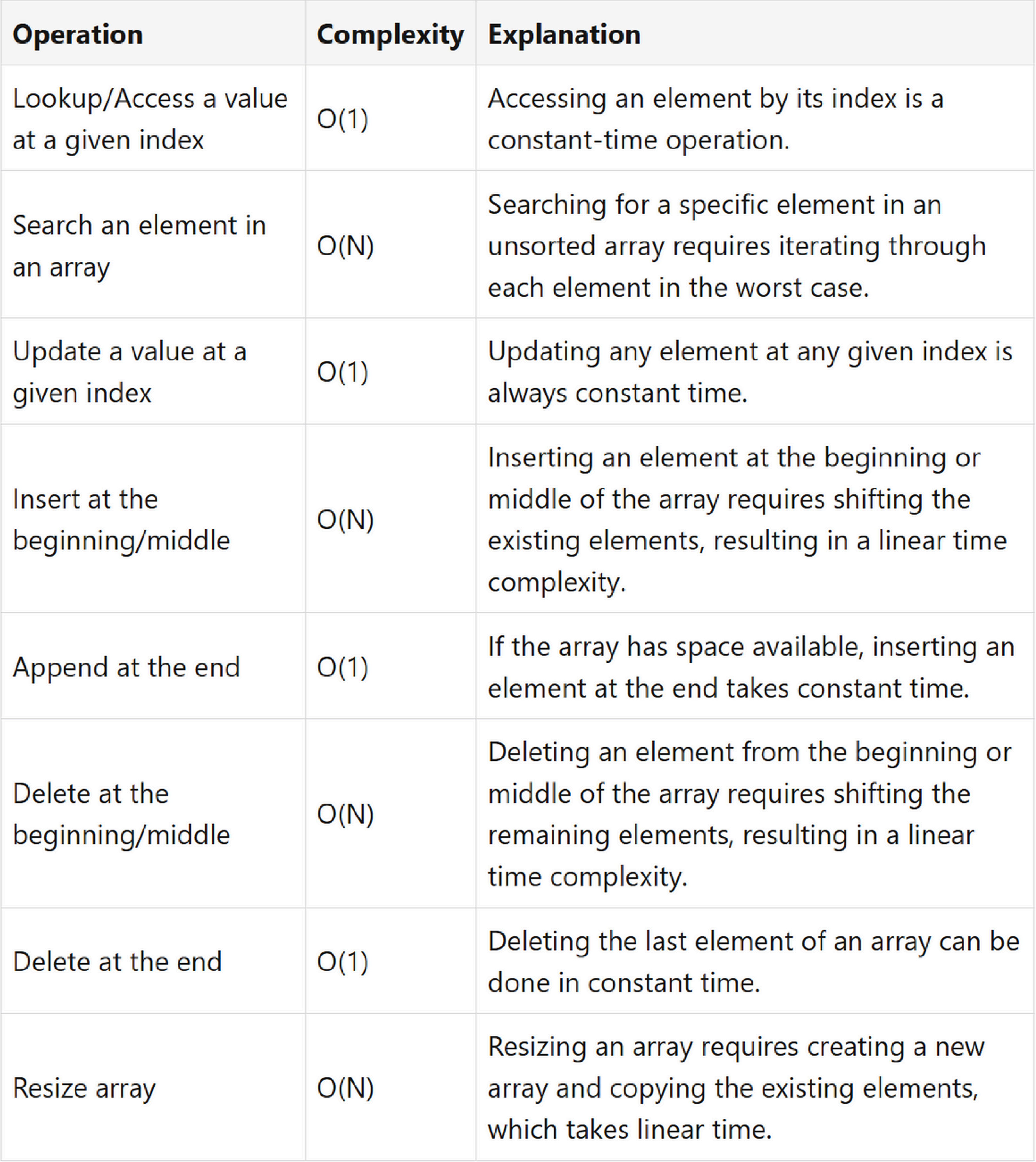


```java
package arrays.test;

public class ArrayUtils {
    int[] items;
    int itemsCount;

    ArrayUtils(int size) {
        items = new int[size];
        itemsCount = 0;
    }

    // Method to Check if array is full or not
    boolean isFull() {
        // if(itemsCount==items.length){
        // return true;
        // }
        // else {
        // return false;
        // }
        return itemsCount == items.length;
    }

    // Method to access pecific item
    void accessItem(int index) {
        if (index < items.length)
            System.out.println(items[index]);
        else
            System.out.println(x:"Item Not Found");
    }

    // Methd to append otems "Add to last"
    void append(int item) {
        // if(itemsCount==items.length){
        // System.out.println("Insertion Failed , Array is Full");
        // }
        if (isFull()) {
            System.out.println(x:"Insertion Failed , Array is Full");
            return;
        }
        // items[itemsCount]=item;
        // itemsCount++;

        items[itemsCount++] = item;

    }

    // Method to traverse array items
    void traverseItems() {
        // for(int i=0; i<items.length;i++){}
        for (int i = 0; i < itemsCount; i++) {
            System.out.println("Item in index " + i + " is: " + items[i]);
        }
        // another way : use for each loop
        // for(int item : items) {
        // System.out.println("Item is "+item );

        // }
    }

```



```java

    // Method to search a specific item

    void searchItem(int item) {

        for (int i = 0; i < itemsCount; i++) {
            if (items[i] == item) {
                System.out.println("Item " + item + " is found at index " + i);
                return;
            }
        }
        System.out.println(x:"Item Not Found");
    }

    boolean isItemFound(int item) {
        for (int i : items) {
            if (item == i)
                return true;
        }
        return false;

    }

    // Method to insert item in specific location.
    void insert(int pos, int newItem) {

        // 1-st step : check if there's an empty location(s) to insert item.
        if (isFull()) {
            System.out.println(x:"Failed Insertion, Arrayis Full");
            return;
        }

        // 2nd-step : shift items.
        for (int i = itemsCount; i > pos; i--) {
            items[i] = items[i - 1];
        }

        items[pos] = newItem;
        itemsCount++;
    }

    // method to delete last item.
    void deleteLastItem() {
        int[] newArr = new int[itemsCount];

        // copy items except last one
        for (int i = 0; i < newArr.length; i++) {
            newArr[i] = items[i];
        }
        items = newArr;
        itemsCount--;
    }

    void deleteItemAtspecificIndex(int index) {

        for (int i = index; i < itemsCount - 1; i++) {
            items[i] = items[i + 1];
        }
        itemsCount--;
    }

    void enlargeArray(int newSize) {
        if (newSize < items.length) {
            System.out.println(x:"Can't Enlarge array");
        }
        int[] newArray = new int[newSize];

        for (int i = 0; i < items.length; i++) {
            newArray[i] = items[i];
        }
        items = newArray;

    }
}

```

# Time Complexities for different operations



| Operation | Complexity | Explanation |
|---|---|---|
| Lookup/Access a value at a given index | O(1) | Accessing an element by its index is a constant-time operation. |
| Search an element in an array | O(N) | Searching for a specific element in an unsorted array requires iterating through each element in the worst case. |
| Update a value at a given index | O(1) | Updating any element at any given index is always constant time. |
| Insert at the beginning/middle | O(N) | Inserting an element at the beginning or middle of the array requires shifting the existing elements, resulting in a linear time complexity. |
| Append at the end | O(1) | If the array has space available, inserting an element at the end takes constant time. |
| Delete at the beginning/middle | O(N) | Deleting an element from the beginning or middle of the array requires shifting the remaining elements, resulting in a linear time complexity. |
| Delete at the end | O(1) | Deleting the last element of an array can be done in constant time. |
| Resize array | O(N) | Resizing an array requires creating a new array and copying the existing elements, which takes linear time. |

# Limitations of Array

- **Fixed Size:** Once an array is created, its size cannot be changed.
- **Insertion and Deletion are Costly:** Adding or removing elements requires shifting other elements.
  → Time complexity can be O(n).
- **Wasted Memory:** If the array is larger than needed, unused elements waste memory.
- **Contiguous Memory Requirement:** Arrays require a continuous block of memory,
  → which may cause issues for very large arrays.
- **No Built-in Dynamic Behavior:** Java arrays do not grow or shrink automatically — you have to manually create a new one and copy the data.

**cause of these limitations, we need a more flexible data structure that can:**

- **Grow and shrink in size dynamically,**
- **Allow easy insertion and deletion of elements.**

➡️ **That’s why we study Linked Lists next.**

# THANK YOU!