Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 1

INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

1.8 CONTROL FLOW, ARRAY

COMPUTER SCIENCE & ENGINEERING

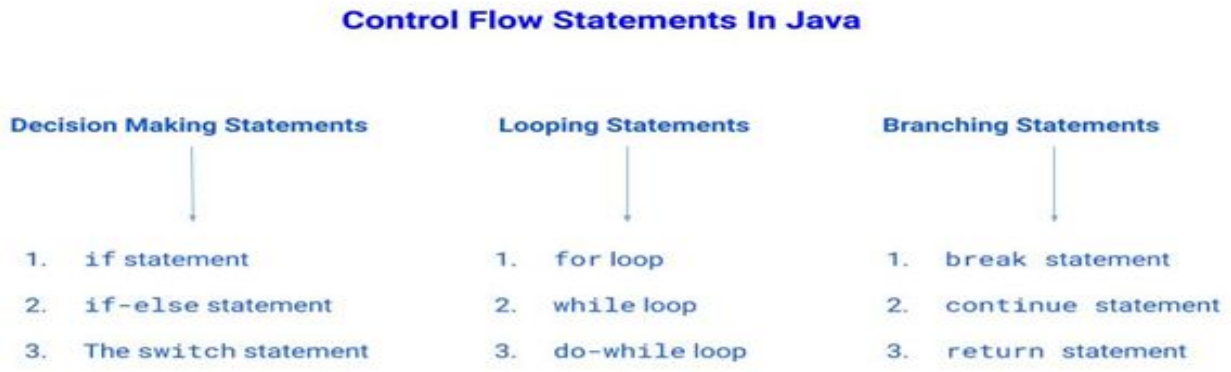
Introduction

- In Java, program is a set of statements and which are executed sequentially in order in which they appear. In that statements, some calculation have need of executing with some conditions and for that a control to that statements has been provided.

- Control flow statements, however, break up the flow of execution by employing decision making ,looping and branching, enabling your program to conditionally execute particular blocks of code.

Types of control flow statements

- In java , control structure is divided into three types:
  1. Selection statement / Decision Making statements
  2. Iteration statement / looping statements
  3. Jumps in statement / Branching statements



Selection statement / Decision Making statements

- Selection statement is also called as Decision making statements because it provides the decision making capabilities to the statements.
- In selection statement, there are two types:
  - if statement
  - switch statement.
- These two statements are allows the user to control the flow of a program with their conditions.

Java If Statements

- The Java if statement is used to test the condition.
- It Checks Boolean condition: true or false.
- There are various types of if statement in Java.

1. if statement
2. if-else statement
3. if-else-if ladder
4. nested if statement

Java If Statement

- The Java if statement tests the condition. It executes the *if block* if condition is true
- **Syntax:**

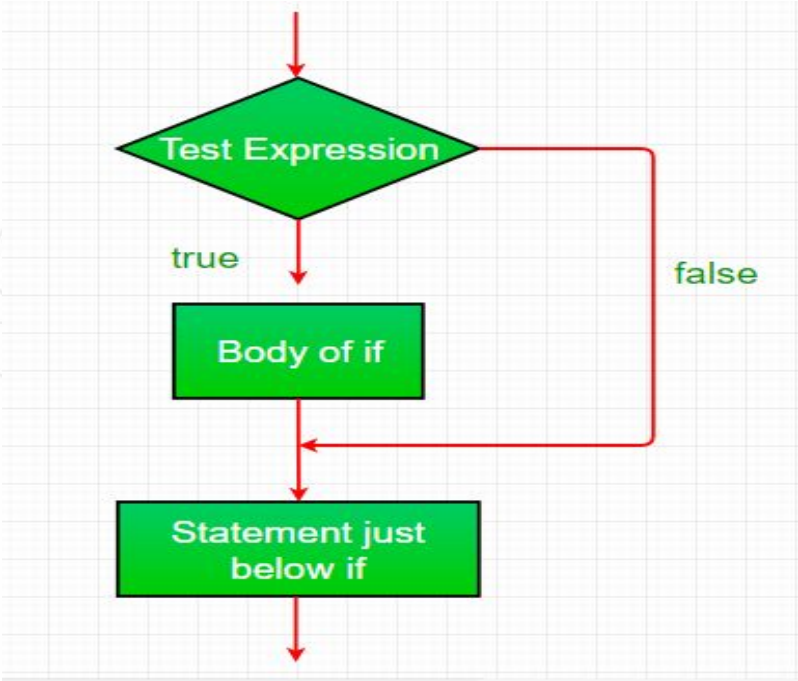
```
if(condition){
//code to be executed
}
```

- **Example:**

```
if(age>18){
        System.out.print("Age is greater than 18");
 }
```

Java If Statement

Flow Diagram



Java If Statements

- **Program**

```
//Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

Output:

```
Age is greater than 18
```

Java If - else Statement

- The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.
- **Syntax:**

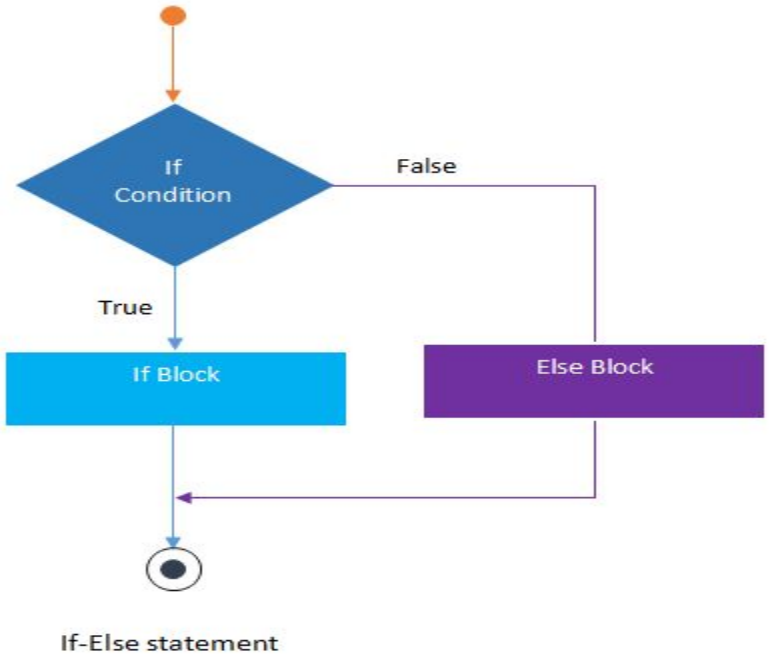
```
if(condition){
//code if condition is true
}else{
//code if condition is false
 }
```

- **Example :**

```
if(number%2==0){
    System.out.println("even number");
}else{
     System.out.println("odd number");          }
```

Java If - else Statement

- **Flow Diagram**



If-Else statement

Java If - else Statement

- **Program**

```
//A Java Program to demonstrate the use of if-else statement.
//It is a program of odd and even number.
public class IfElseExample {
public static void main(String[] args) {
   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not
   if(number%2==0){
      System.out.println("even number");
   }else{
      System.out.println("odd number");
   }
}
}
```

Output:

```
odd number
```

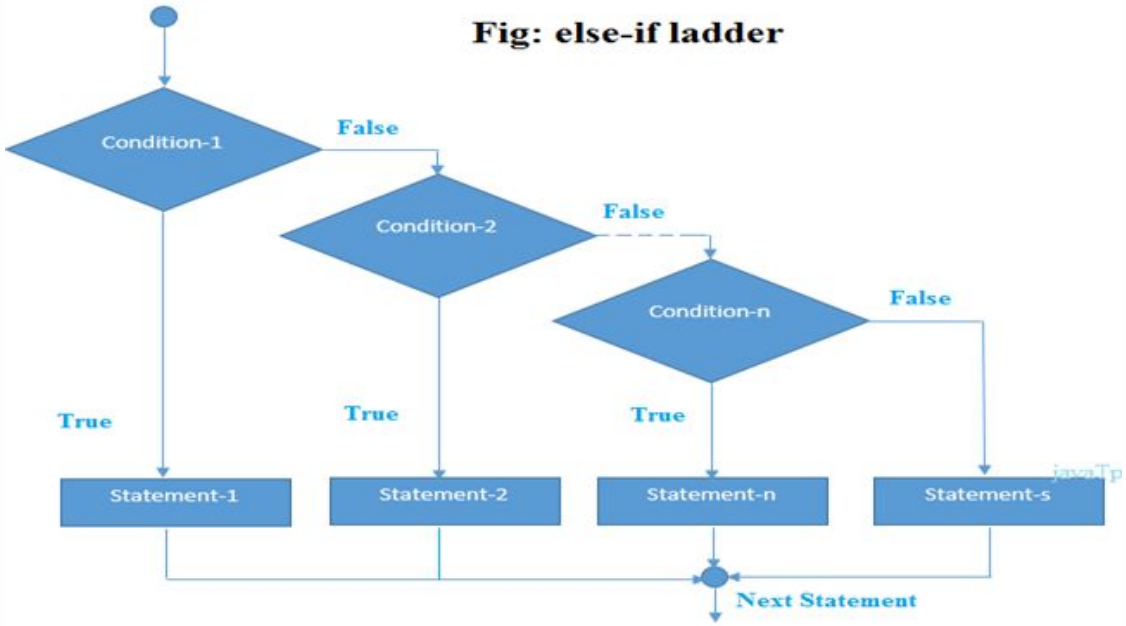
Java If - else ladder Statement

- The if-else-if ladder statement executes one condition from multiple statements.
- **Syntax**

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
 //code to be executed if condition3 is true
 }                ...
else{
//code to be executed if all the conditions are false
 }
```

Java If - else ladder Statement

- **Flow Diagram**



Java If - else ladder Statement

- **Program**

```
//Java Program to demonstrate the use of If else-if ladder.
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.
public class IfElseIfExample {
public static void main(String[] args) {
    int marks=65;
    if(marks<50){
        System.out.println("fail");      }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");     }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");     }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");     }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
        System.out.println("Invalid!");      }
}
}
```

Output:

```
C grade
```

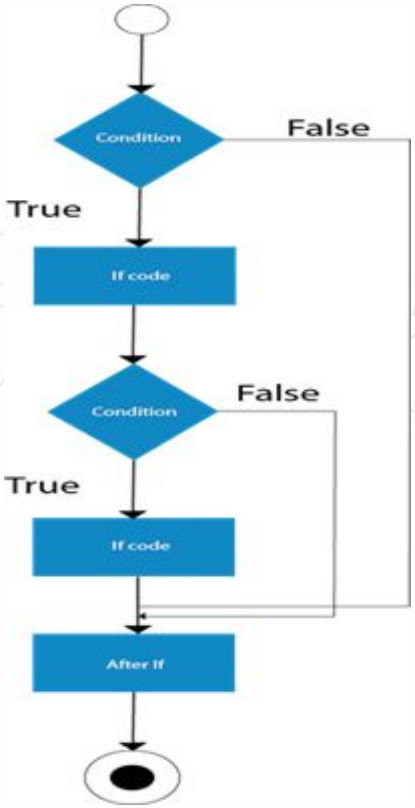
Java Nested If Statement

- **Java Nested if statement**
- **The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.**
- **Syntax:**

```
if(condition){
        //code to be executed
    if(condition){
        //code to be executed
    }
}
```

Java Nested If Statement

- **Flow Diagram**



Java Nested If Statement

- **Program**

Program to find largest number of three given numbers using if else.

```
class Test
{
    public static void main(String args[])
    {
        int x = 20;
        int y = 15;
        int z = 30;
        if (x > y)
        {
            if (x > z)
                System.out.println("Largest number is: "+x);
            else
                System.out.println("Largest number is: "+z);
        }
        else
        {
            if (y > z)
                System.out.println("Largest number is: "+y);
            else
                System.out.println("Largest number is: "+z);
        }
    }
}
```

Output: Largest number is: 30

Java Switch Statement

- The Java *switch statement* executes one statement from multiple conditions.
- It is like if –else-if ladder statement.
- The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use string in the switch statement.
- In other words, the switch statement tests the equality of a variable against multiple values.

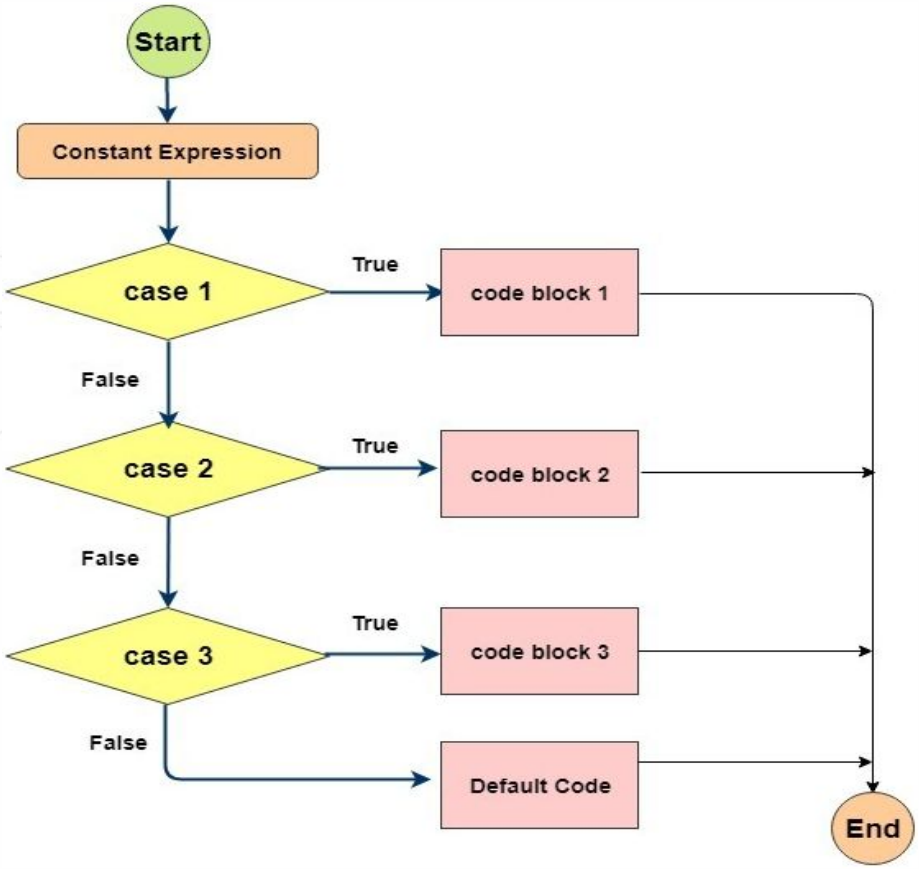
Java Switch Statement

- Syntax

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....
 default:
 code to be executed if all cases are not matched;
}
```

Java Switch Statement

- Flow Diagram



Java Switch Statement

- **Program**

Finding Month Example:

```
//Java Program to demonstrate the example of Switch statement
//where we are printing month name for the given number
public class SwitchMonthExample {
public static void main(String[] args) {
   int month=7;       //Specifying month number
   String monthString="";
    switch(month){
   case 1: monthString="1 - January";
   break;
   case 2: monthString="2 - February";
   break;
   case 3: monthString="3 - March";
   break;
   case 4: monthString="4 - April";
    ......
    .....
   default:System.out.println("Invalid Month!");
   }
   System.out.println(monthString);
}  }
```

Output: 7 - July

Iteration / Loop Statement

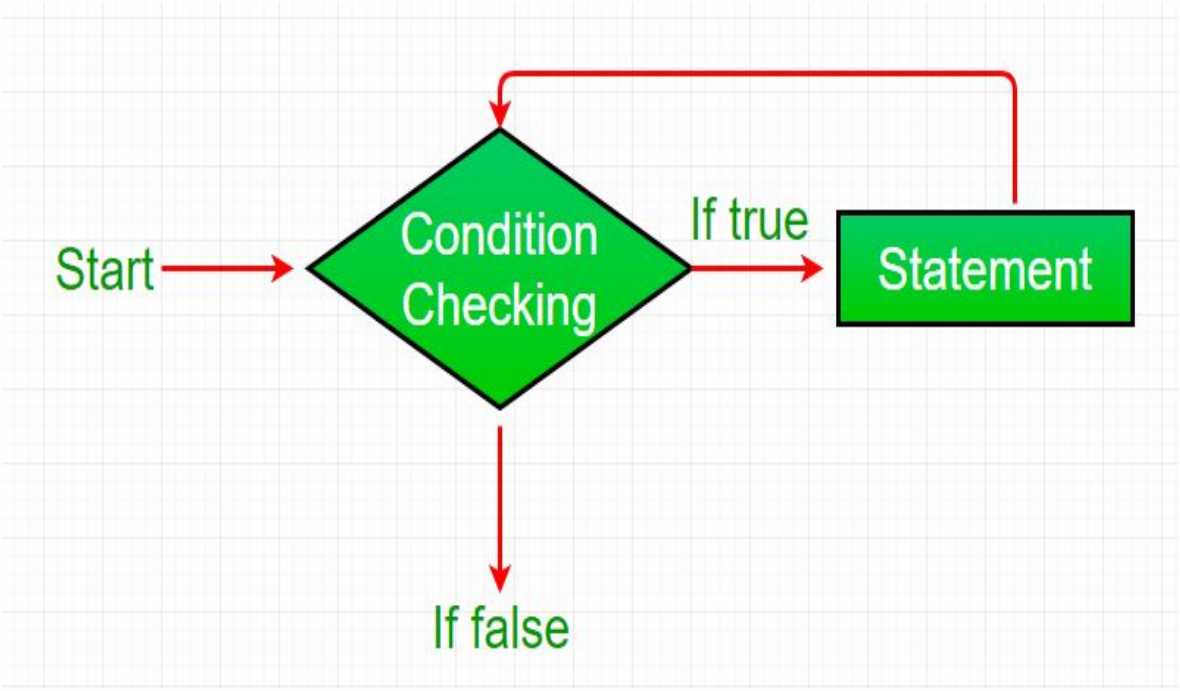
- The process of repeatedly executing a statements and is called as looping.
- The statements may be executed multiple times (from zero to infinite number).
- If a loop executing continuous then it is called as Infinite loop. Looping is also called as iterations.
- In Iteration statement, there are three types of operation:

  1.while loop

  2.do-while loop

  3. for loop

Iteration / Loop Statement



While loop Statement

- **Definition**

A while loop statement in java programming language repeatedly executes a target statement as long as a given condition is true.

- **Syntax**

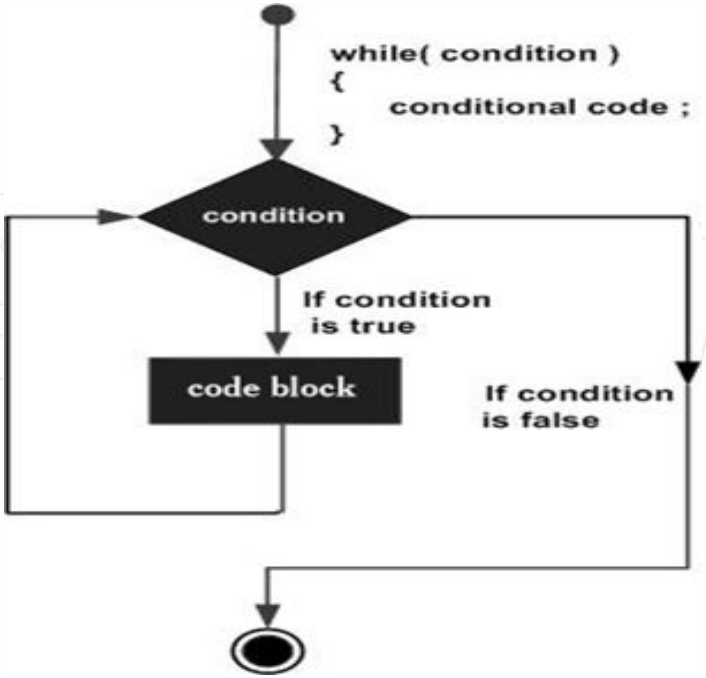
```
while(Boolean_expression)
{
   //Statements
}
```

- **Example:**

```
while( x < 20 ){
        System.out.print("value of x : " + x );
         x++;
        System.out.print("\n");          }
```

While loop Statement

- **Flow Diagram**



While loop Statement

- **Program**

```
class Test {
  public static void main(String args[]) {
    int x = 10;
    while( x < 20 ){
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
}
  }
}
```

Do While loop Statement

- **Definition**

  A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

- **Syntax**

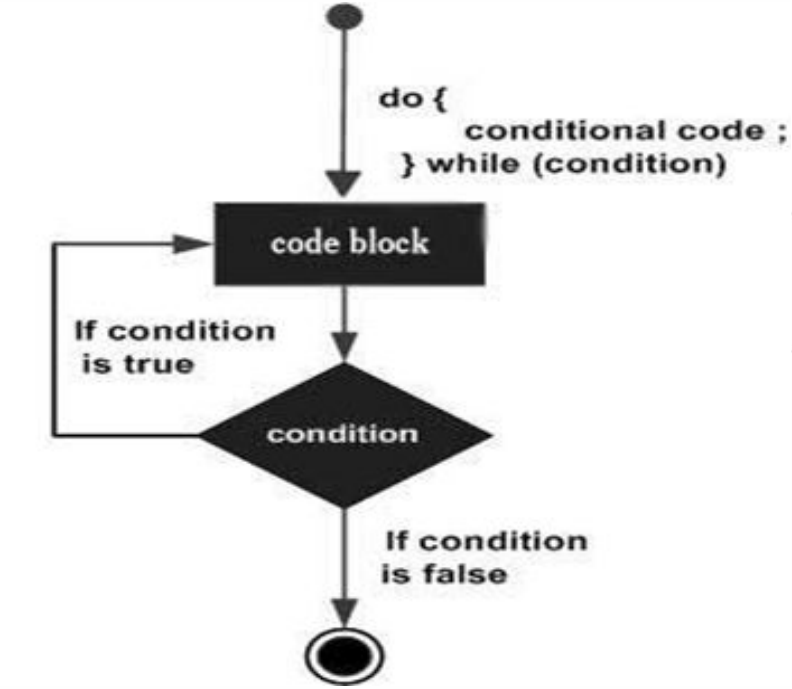
```
do
{    //Statements
} while(Boolean_expression);
```

Example

```
do{
   System.out.print("value of x : " + x );
   x++;
   System.out.print("\n");
   }while( x < 20 );
```

Do-While loop Statement

- **Flow Diagram**



Do-While loop Statement

- **Program**

```
public class Test {
  public static void main(String args[]){
    int x = 10;
    do{
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
    }while( x < 20 );
  }
}
```

- for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. A for loop is useful when the user knows how many times a task is to be repeated.

for loop Statement

- **Syntax**

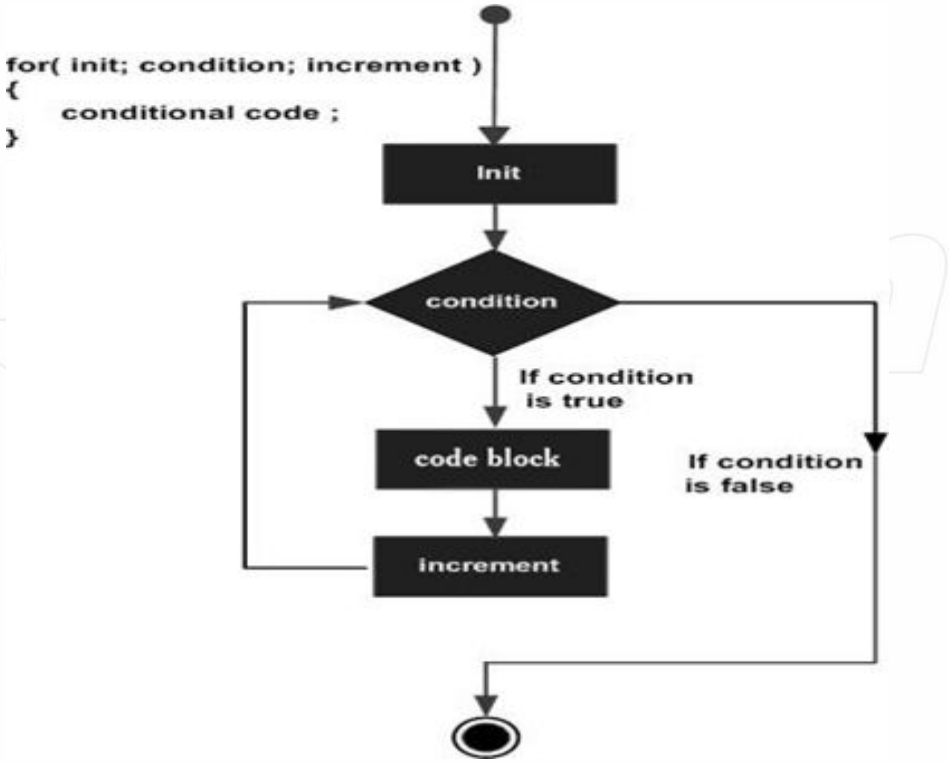
```
for(initialization; Boolean_expression; update)        {
    //Statements
}
```

- **Example:**

```
for (int i = 1; i <= 5 ++i)        {
        System.out.println("Number " + i);
}
```

for loop Statement

- Flow Diagram



for loop Statement

- **Program**

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

      for (int i = 1; i <= 5 ++i) {
         System.out.println("Number " + i);
      }
   }
}
```

Output:

```
Number 1
Number 2
Number 3
Number 4
Number 5
```

Jump Statement

- Statements or loops perform a set of operations continually until the control variable will not satisfy the condition.
- If a user want to break the loop when condition satisfies, then Java give a permission to jump from one statement to end of loop or beginning of loop as well as jump out of a loop.
- “break” keyword use for exiting from loop and “continue” keyword use for continuing the loop.

Break Statement

- **break statement**
- The break statement in Java programming language has the following two usages:
- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement
- **Syntax**

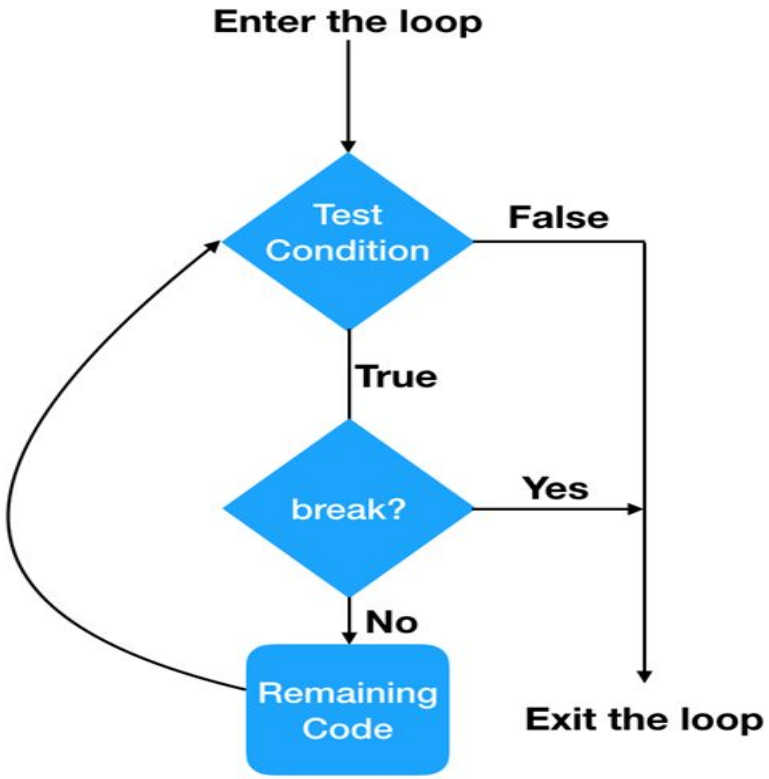
```
break;
```

- **Example**

```
for(int x : numbers ) {
    if( x == 30 ) {
break;
 }
```

Break Statement

- **Flow Diagram**



Break Statement

- **Program**

```
/Java Program to demonstrate the use of break statement
public class BreakExample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }
      System.out.println(i);
   }
} |
}
```

Output:

```
1
2
3
4
```

Continue Statement

- **continue**
- The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.
- In for loop, the continue keyword causes control to immediately jump to the update statement.
- In a while loop or do/while loop, control immediately jumps to the Boolean expression.
- **Syntax**

```
continue;
```

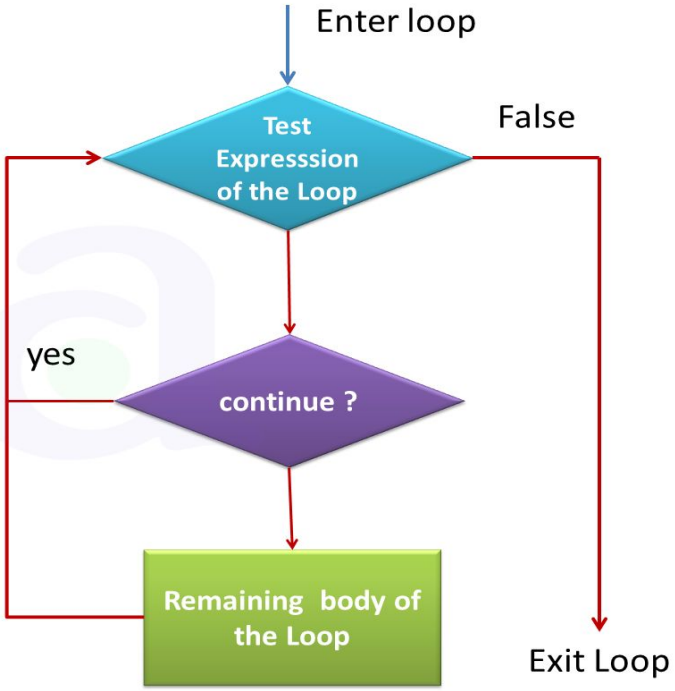
Continue Statement

- **Example**

```
for(int x : numbers ) {
    if( x == 30 ) {
     continue;
    }
```

Continue Statement

- **Flow Diagram**



Continue Statement

- **Program**

```
/Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=5;i++){
      if(i==3){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Output:

```
1
2
4
5
```

Return Statement

- **Definition**
- return is a reserved keyword in Java i.e, we can't use it as an identifier.
- It is used to exit from a method, with or without a value.
- **Syntax**
  - return;
- **Example:**

```
static int myMethod(int x)
{
    return 5 + x;
}
```

Return Statement

- **Program**

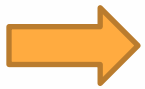
Example : Program for return statement

```
public class MyClass
 {
   static int myMethod(int x)
     {
          return 5 + x;
     }
public static void main(String[] args)
    {
      System.out.println(myMethod(3));
    }
}
// Outputs 8 (5 + 3)
```

Java : Arrays

- Arrays in Java are similar to that of C++ or any other programming language. An array is a data structure which holds the sequential elements of the same type.
- An array is a very common type of data structure wherein all elements must be of the same data type. Once defined, the size of an array is fixed and cannot increase to accommodate more elements. The first element of an array starts with index zero.
- In simple words, it's a programming construct which helps to replace this

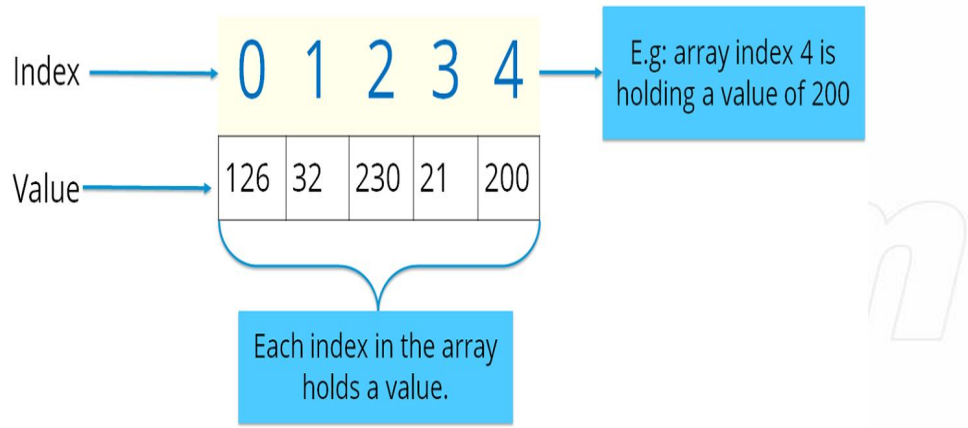
```
x0=0;
x1=1;
x2=2;
x3=3;
x4=4;
x5=5;
```



```
x[0]=0;
x[1]=1;
x[2]=2;
x[3]=3;
x[4]=4;
x[5]=5
```

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.,
- Java array can be also be used as a static field, a local variable or a method parameter.
- The **size** of an array must be specified by an int value and not long or short.
- The direct superclass of an array type is object.
- Every array type implements the interfaces cloneable and java.io.serializable

Each array has two components: index and value.



The indexing starts from zero and goes till (n-1) where n= size of the array. Let's say you want to store 10 numbers, then the indexing starts from zero and goes till 9.

Using an array in your program is a **3 step process**

1) Declaring your Array
2) Constructing your Array
3) Initialize your Array

Declaring your Array

Syntax

<elementType>[] <arrayName>;
or
<elementType> <arrayName>[];

Example:

```
int intArray[];
```

// Defines that intArray is an ARRAY variable which will store integer values

```
int []intArray;
```

Constructing an Array

arrayname = new dataType[]

Example:

intArray = new int[10];

// Defines that intArray will store 10 integer values

Declaration and Construction combined

int intArray[] = new int[10];

Initialize an Array

intArray[0]=1; // Assigns an integer value 1 to the first element 0 of the array

intArray[1]=2; // Assigns an integer value 2 to the second element 1 of the array

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

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

Access the Elements of an Array

Array elements can be accessed by referring to the index number.

The following program accesses the value of the first element in cars:

```
public class MyClass {

  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    System.out.println(cars[0]);

  }
}
```

Output:Volvo

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Array Element

To change the value of a specific element, refer to the index number:

```
public class MyClass {

   public static void main(String[] args) {
     String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
      cars[0] = "Opel";

       System.out.println(cars[0]);

  }
}
```

Output: opel

Array Length

To find out number of elements in an array , use the length property:

```
public class MyClass {

  public static void main(String[] args) {

    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    System.out.println(cars.length);

  }

}
```

Output:4

Loop Through an Array

It is possible to loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

The following example outputs all elements in the **cars** array:

```
public class MyClass {
  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    for (int i = 0; i < cars.length; i++) {

      System.out.println(cars[i]);

    }
  }

}
```

Output

```
Volvo

BMW

FORD

Mazda
```

Loop Through an Array with For-Each

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays:

Syntax

```
for (type variable : arrayname) {
  ...
}
```

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

```
public class MyClass {
 public static void main(String[] args) {
   String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
   for (String i : cars) {
     System.out.println(i);
    }
   }
}
```

First Array Program

```
class ArrayDemo{

  public static void main(String args[]){
    int array[] = new int[7];

    for (int count=0;count<7;count++){

      array[count]=count+1;

}

for (int count=0;count<7;count++){
   System.out.println("array["+count+"] = "+array[count]);

}

//System.out.println("Length of Array  =  "+array.length);

// array[8] =10;

  }

}
```

Output:

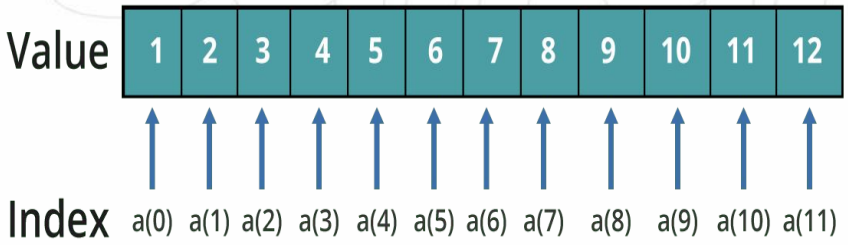
```
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
array[5] = 6
array[6] = 7
```

Single-dimensional Array:

In a single-dimension array, a list of variables of the same type can be accessed by a common name. You can initialize the array using the following syntax:

int a[] = new int[12];

You can refer to the below image where I have stored data with respect to the given index.

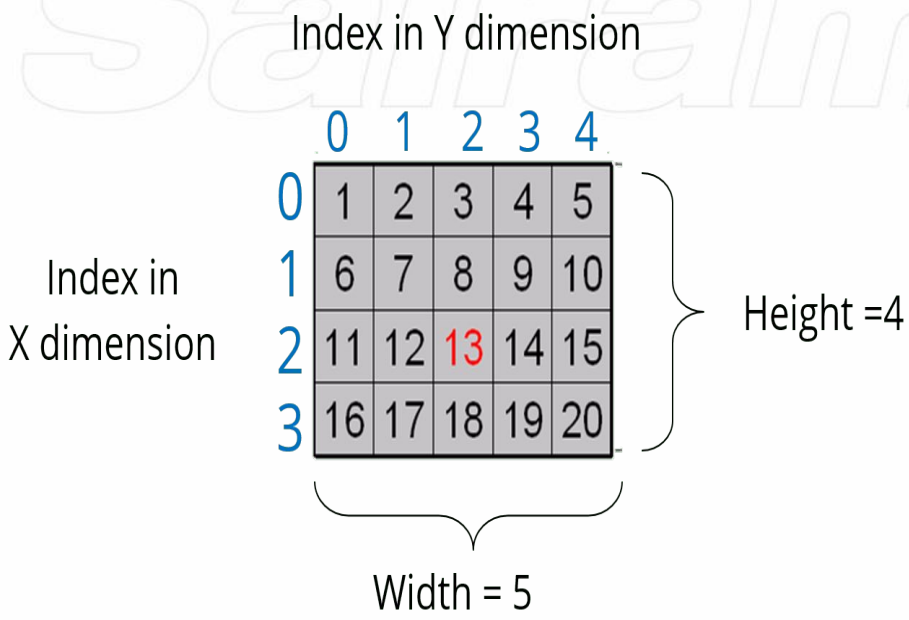


Multi–dimensional Array:

In a multi-dimension array, your data is stored in a matrix form. Here, you can initialize the array using the following syntax:

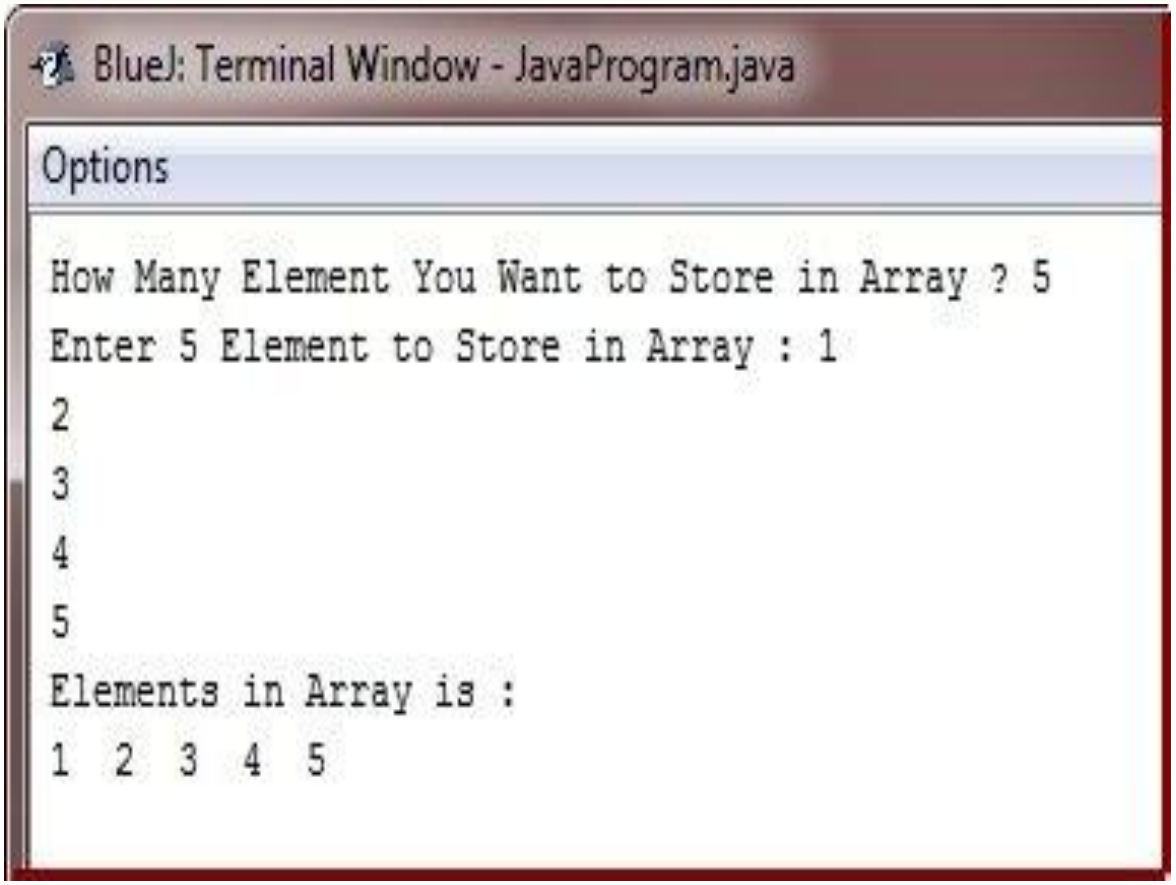
```
int table[][]= new int[4][5];
```

It is quite similar to the matrix that we use in mathematics. Refer to the below image where I have stored data with respect to different dimensions.



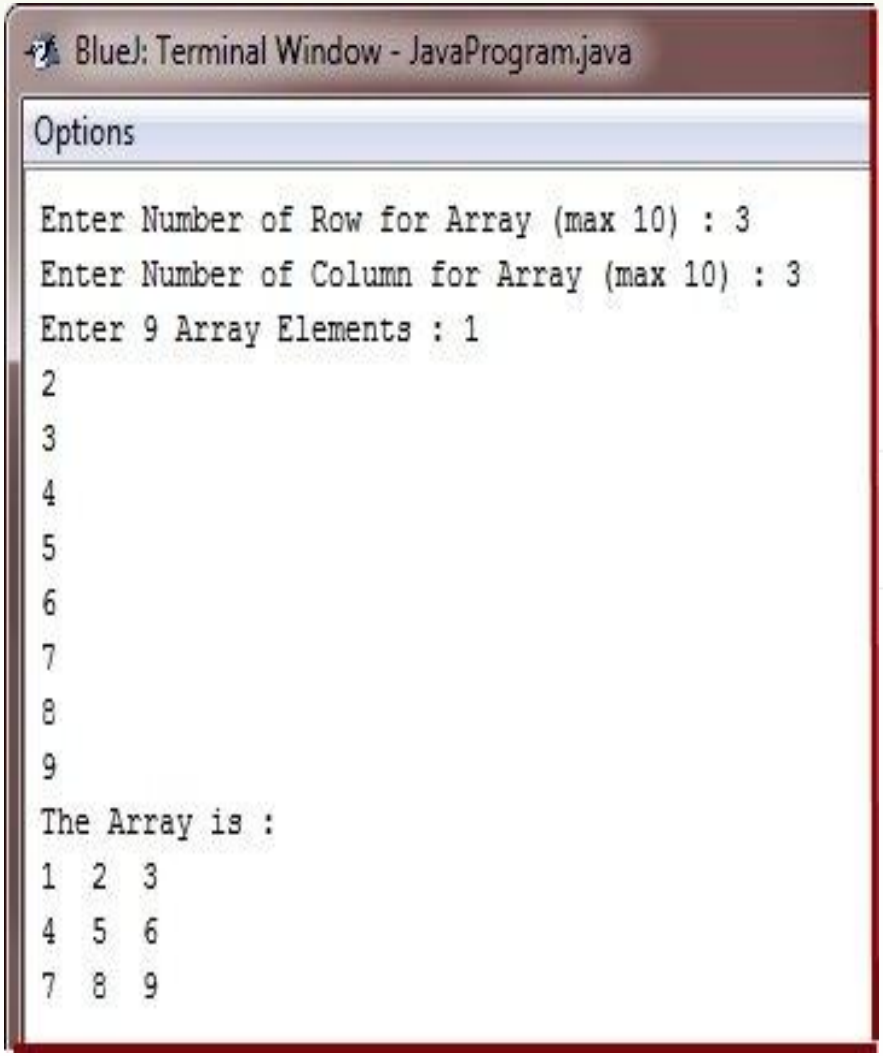
One dimensional Example

```
import java.util.Scanner;

public class JavaProgram    {
   public static void main(String args[])     {
     int arr[] = new int[50];
     int n, i;
     Scanner scan = new Scanner(System.in);

      System.out.print("How Many Element You Want to Store in Array ? ");
      n = scan.nextInt();

      System.out.print("Enter " + n + " Element to Store in Array : ");

       for(i=0; i<n; i++)            {
            arr[i] = scan.nextInt();

       }
      System.out.print("Elements in Array is :\n");
      for(i=0; i<n; i++)     {
          System.out.print(arr[i] + "  ");
     }    } }
```


BlueJ: Terminal Window - JavaProgram.java
Options
How Many Element You Want to Store in Array ? 5
Enter 5 Element to Store in Array : 1
2
3
4
5
Elements in Array is :
1 2 3 4 5

```
{
  public static void main(String args[])
  {
int row, col, i, j; int arr[][] = new int[10][10];
Scanner scan = new Scanner(System.in);
System.out.print("Enter Number of Row for Array (max 10) : ");
row = scan.nextInt(); System.out.print("Enter Number of Column for Array (max 10) : ");
col = scan.nextInt();    System.out.print("Enter " +(row*col)+ " Array Elements : ");
for(i=0; i<row; i++)
   {
         for(j=0; j<col; j++)
        {
                  arr[i][j] = scan.nextInt();

        }
   }
System.out.print("The Array is :\n");
for(i=0; i<row; i++)
{
  for(j=0; j<col; j++)
  {

    System.out.print(arr[i][j]+ "  ");
   }
     System.out.println();
   }
 } }
```

Multidimensional Example


BlueJ: Terminal Window - JavaProgram.java
Options
Enter Number of Row for Array (max 10) : 3
Enter Number of Column for Array (max 10) : 3
Enter 9 Array Elements : 1
2
3
4
5
6
7
8
9
The Array is :
1 2 3
4 5 6
7 8 9

MCQ LINK

- **Quiz link for Control Flow**

  https://forms.gle/EgYkM5NR9agMiQwk7