Marwadi University

Unit – 2 Selection, Iteration and Array

Prepared By

Prof. Ravikumar R N

Assistant Professor, CE Dept.

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

Contents

Marwadi University

- If statements
- Switch statement
- While statement
- For statement
- Do-while statement
- Break and continue keywords
- One dimensional and multidimensional arrays
- Jagged array
- Methods to copy array

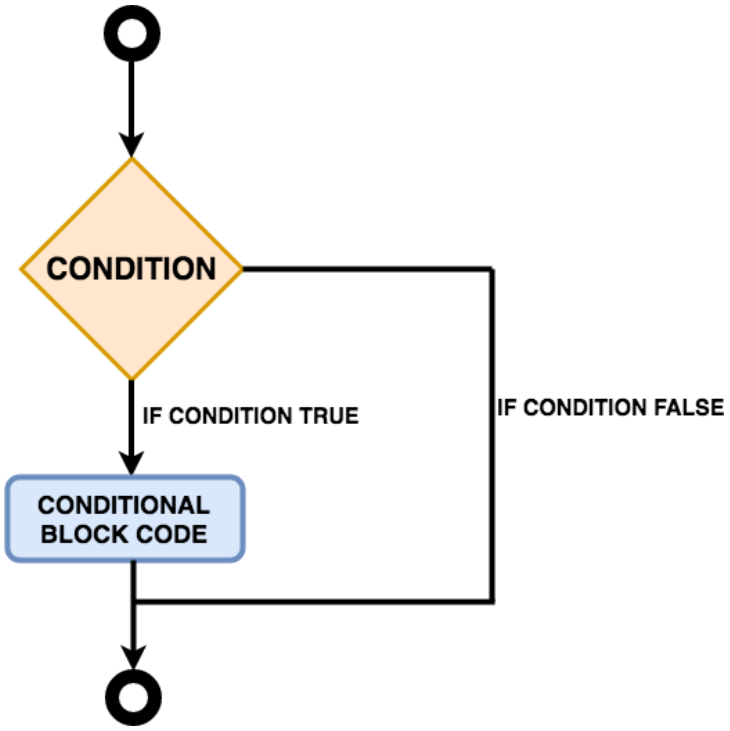
Selections

Marwadi University

1. If Statement
2. Two way if-else statement
3. Nested if and multi-way if else statement
4. Switch statement
5. Conditional Expression (Ternary)

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
}
```

Check Program: Unit – 2 → IfDemo.java

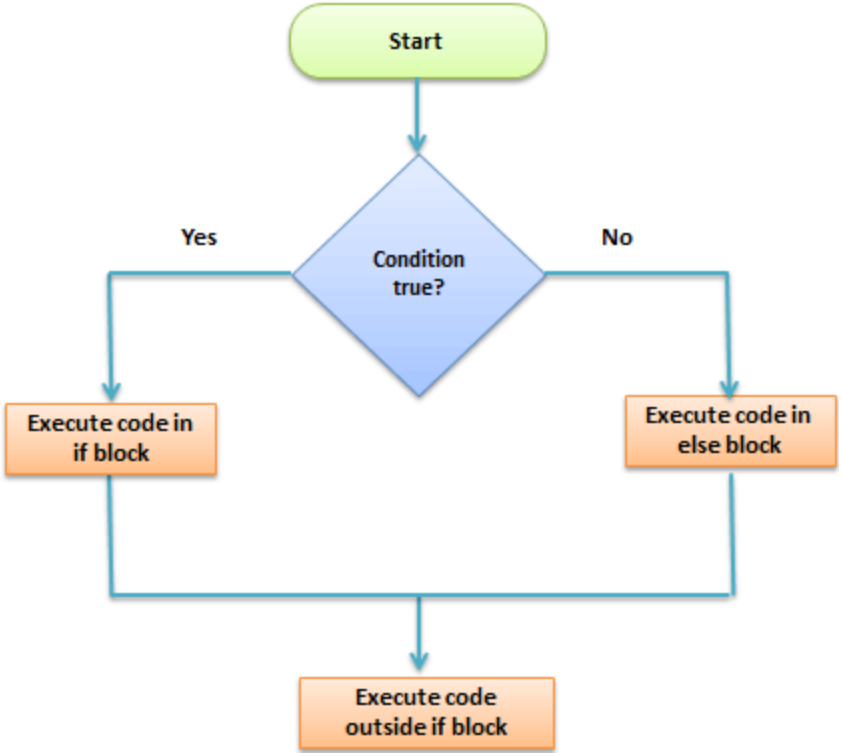
Two way If (if.. else Statement)

Marwadi University

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
  }
```



Check Program: Unit – 2 → IfElseDemo.java

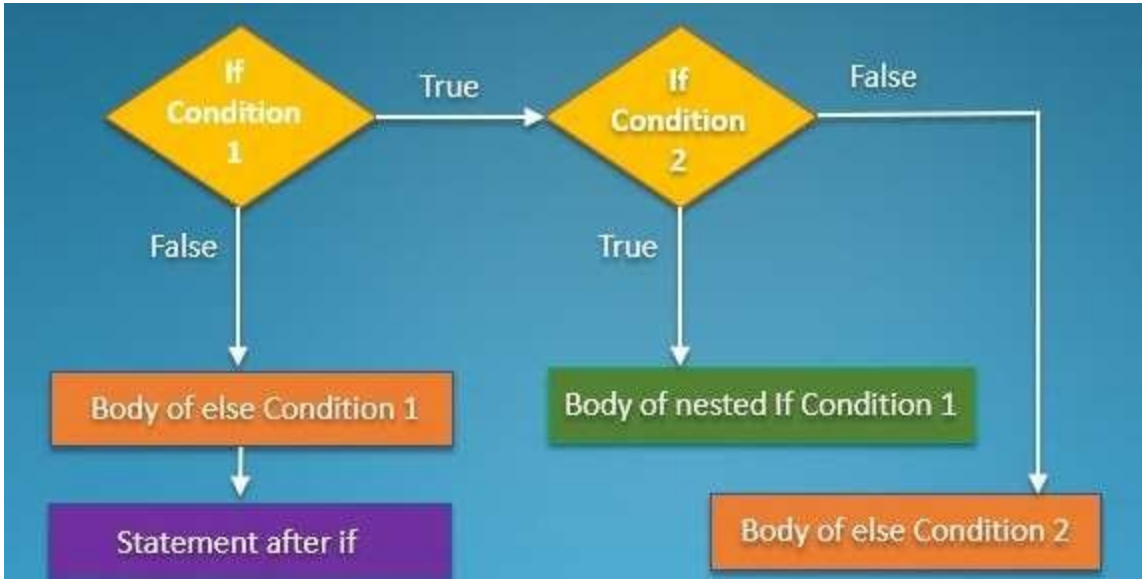
Nested if statement

Marwadi University

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
    }
}
```



Check Program: Unit – 2 → NestedIfDemo.java

Multi way if... ladder if

Marwadi University

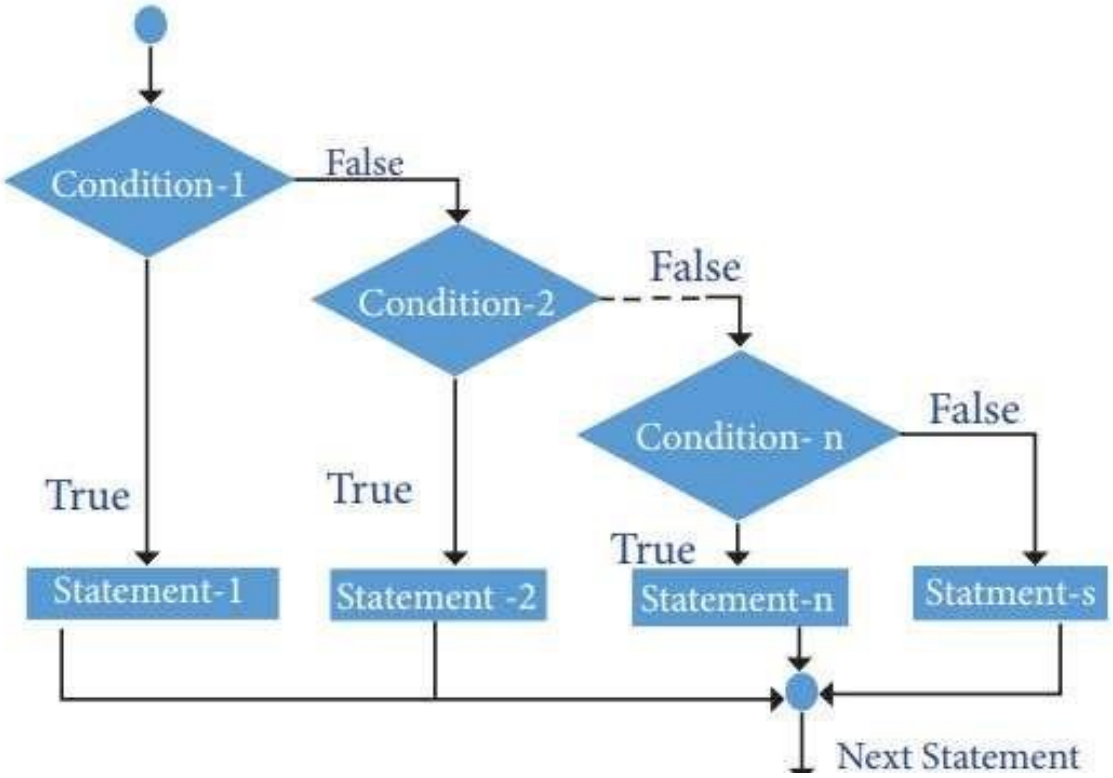
The if-else-if ladder statement executes one condition from multiple statements.

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
}
```

Multi way if... ladder if

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



Check Program: Unit – 2 → LadderIfDemo.java

Multi way if... ladder if

```
class IfElseLadder{
public static void main(String[] args)
{
        // initializing expression
        int i = 20;

        // condition 1
        if (i == 10)

System.out.println("i is 10\n");

        // condition 2
        else if (i == 15)

System.out.println("i is 15\n");

// condition 3
else if (i == 20)
System.out.println("i is 20\n");
else
System.out.println("i is not present\n");

System.out.println("Outside if-else-if");
        }
}
```

Java Switch Statement

Marwadi University

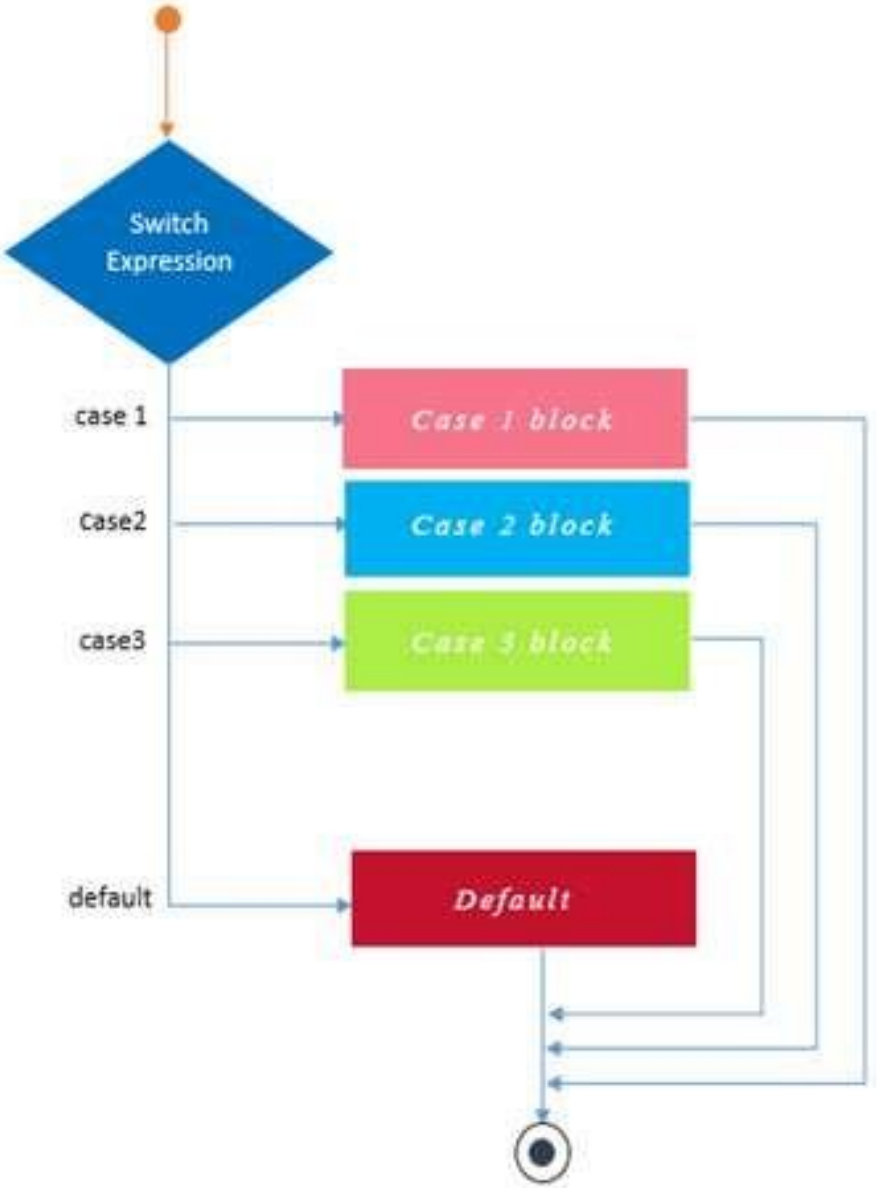
The Java switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

1) There can be one or N number of case values for a switch expression.
2) The case value must be of switch expression type only. The case value must be literal or constant. It doesn't allow variables.
3) The case values must be unique. In case of duplicate value, it renders compile-time error.
4) The Java switch expression must be of byte, short, int, long (with its Wrapper type), enums and string.
5) Each case statement can have a break statement which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
6) The case value can have a default label which is optional.

Check Program: Unit – 2 → SwitchDemo.java

Java Switch Statement



Check Program: Unit – 2 → SwitchDemo.java

Java Switch Statement

Marwadi University

```
public class SwitchDemo{
        public static void main(String[] args)
        {
                int day = 5;
                String dayString;

                // switch statement with int data type
                switch (day) {
                case 1:
                        dayString = "Monday";
                        break;
                case 2:
                        dayString = "Tuesday";
                        break;
                case 3:
                        dayString = "Wednesday";
                        break;
                default:
                        dayString = "Invalid day";
                }
                System.out.println(dayString);
        }
}
```

Loops in Java

Marwadi University

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more
readable.

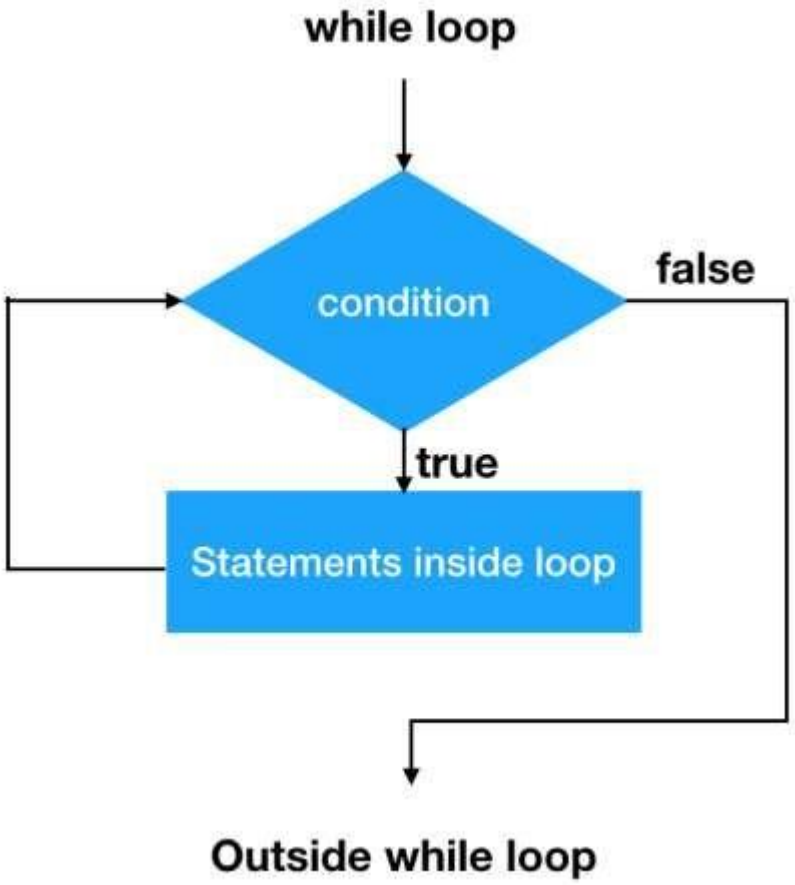
Java Supports :-
- While loop
- Do-while loop
- for loop

Java While Loop

The while loop loops through a block of code as long as a specified condition is true:

Syntax

```
while (condition) {
  // code block to be executed
}
```



Check Program: Unit – 2 → WhileDemo.java

Java While Loop

Marwadi University

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;
   }
}
}
```

```
class whileLoopDemo {
   public static void main(String args[])
   {
      // initialization expression
      int i = 1;

      // test expression
      while (i < 6) {
         System.out.println("Hello World");

         // update expression
         i++;
      }
   }
}
```

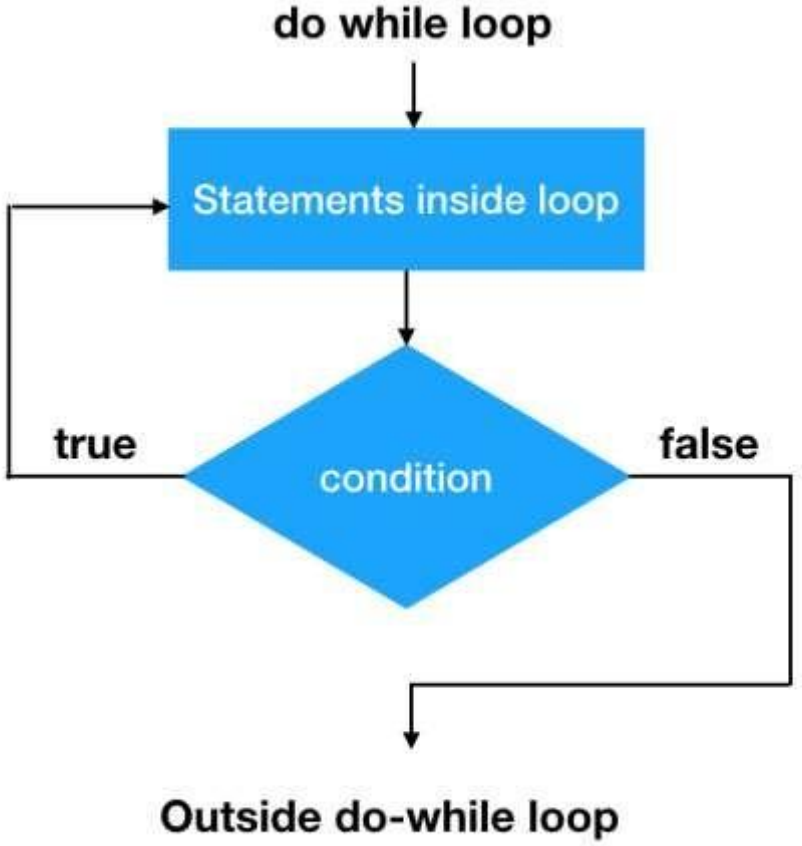
Java do.. While Loop

Marwadi University

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```



Check Program: Unit – 2 → DoWhileDemo.java

Java do.. While Loop

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

```
public class DoWhileExample2 {
public static void main(String[] args) {
    do{
        System.out.println("infinitive do while
loop");
    }while(true);
}
}
```

Check Program: Unit – 2 → DoWhileDemo.java

Java for Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

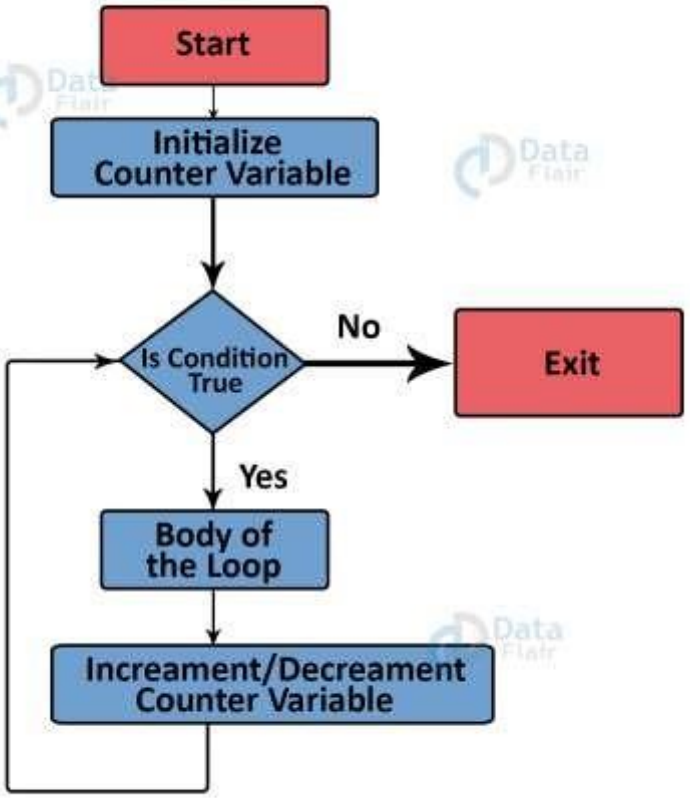
Syntax

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.



Check Program: Unit – 2 → ForDemo.java

Java for Loop

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

        // for loop
        for (int i = 1; i <= 10; ++i) {

            // if the value of i is 5 the loop terminates
            if (i == 5) {
                break;
            }
            System.out.println(i);
        }
    }
}
```

Check Program: Unit – 2 → ForDemo.java

Java Scanner, If, While

Marwadi University

```
import java.util.Scanner;

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

      Double number, sum = 0.0;

      // create an object of Scanner
      Scanner input = new Scanner(System.in);

      while (true) {
         System.out.print("Enter a number: ");

         // takes double input from user
         number = input.nextDouble();

         // if number is negative the loop terminates
         if (number < 0.0) {
            break;
         }

         sum += number;
      }
      System.out.println("Sum = " + sum);
   }
}
```

Check Program: Unit – 2 → ForDemo.java

KNOWLEDGE IS THE CURRENCY FOR THE 21st CENTURY

Java Break statement

hen a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Check Program: Unit – 2 → BreakDemo.java

Java 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);
    }
}
}
```

```
1
2
3
4
```

Java continue statement

The continue statement is used in loop control structure when you need to jump/skip to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Cannot be used with Switch Statement

Check Program: Unit – 2 → ContinueDemo.java

Java continue statement

Marwadi University

```
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the particular statement
      }
      System.out.println(i);
   }
}
}
```

```
1
2
3
4
6
7
8
9
10
```

Break	Continue
The break statement is used to terminate the loop immediately.	The continue statement is used to skip the current iteration of the loop.
break keyword is used to indicate break statements in java programming.	**continue** keyword is used to indicate continue statement in java programming.
We can use a break with the switch statement.	We can not use a continue with the switch statement.
The break statement terminates the whole loop early.	The continue statement brings the next iteration early.
It stops the execution of the loop.	It does not stop the execution of the loop.

Java Nested Loop

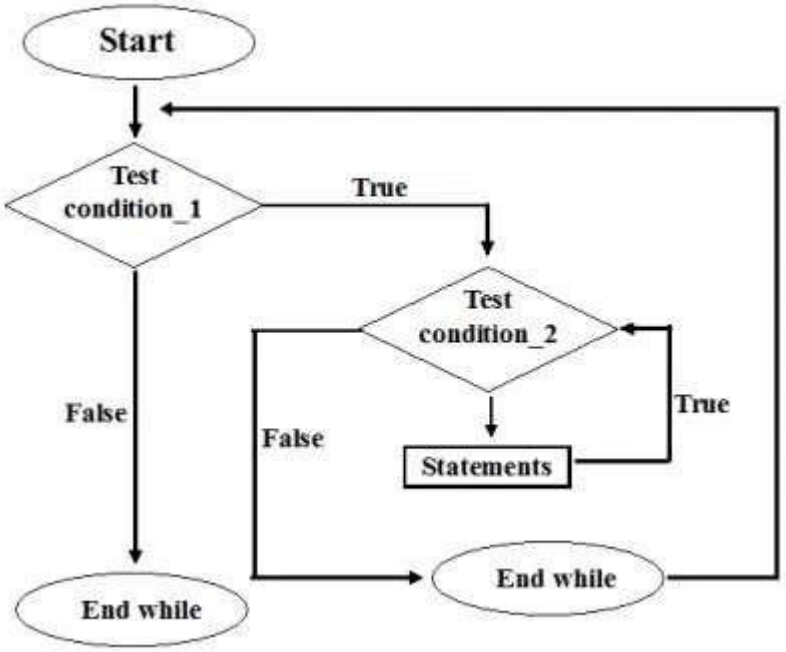
Marwadi University

If a loop exists inside the body of another loop, it's called a nested loop. Here's an example of the nested for loop.

```
int row,column;
// outer  loop
row = 1;
while(row <= 3) {
column=1;
  // inner loop
 while(column <=row) {
                System.out.println(row + " ");
         Column++;
  }
   System.out.println("\n");
   row++;
}
```

OUTPUT:-

```
1
2 2
3 3 3
```



Check Program: Unit – 2 → NestedForDemo.java

Java Nested Loop

Marwadi University

```
class Main {
  public static void main(String[] args) {
    int rows = 5;

    for (int i = 1; i <= rows; ++i) {

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

Questions

Marwadi University

- Will the Switch statement work without using Break?
- Do while vs while?
- For vs for each?
- Is it a good idea to use Continue statement in Switch case?
- “If” and multiple “else if” works without “else” part or not?
- Break statement is used to jump/skip statements?
- Inner “if” block condition executes only when outer if block condition is true?

Common mathematical methods

The **java.lang.Math** class contains various methods for performing basic numeric operations such as the logarithm, cube root, and trigonometric functions etc.

Min, Max, abs , log, sqrt method

Ex. Math.max(2.5, 3) returns 3.0

Math.min(2.5, 4.6) returns 2.5

Math.abs(-2) returns 2

Math.abs(-2.1) returns 2.1

Check Program : Unit – 2 → MathRandomDemo1.java

Common mathematical methods

Marwadi University

Method	*Description*
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.
`rint(x)`	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value.
`round(x)`	Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double.

Rounding methods

Ex. Math.ceil(2.1) returns 3.0

Math.floor(2.8) returns 2.0

Math.rint(2.5) returns 2.0

Math.round(2.6f) returns 3

Check Program : Unit – 2 → MathRandomDemo3.java

Common mathematical methods

Trigonometric methods in Math class

Ex. Math.toDegrees(Math.PI / 2) returns **90.0**

Math.toRadians(30) returns **0.5236** (same as π/6)

Math.atan(**1.0**) returns **0.785398** (same as π/4)

Method	*Description*
`sin(radians)`	Returns the trigonometric sine of an angle in radians.
`cos(radians)`	Returns the trigonometric cosine of an angle in radians.
`tan(radians)`	Returns the trigonometric tangent of an angle in radians.
`toRadians(degree)`	Returns the angle in radians for the angle in degree.
`toDegree(radians)`	Returns the angle in degrees for the angle in radians.
`asin(a)`	Returns the angle in radians for the inverse of sine.
`acos(a)`	Returns the angle in radians for the inverse of cosine.
`atan(a)`	Returns the angle in radians for the inverse of tangent.

Check Program : Unit – 2 → MathRandomDemo2.java

Common mathematical methods

Marwadi University

```
public class MathRandomDemo1
{
   public static void main(String[] args)
   {
      double x = 28;
      double y = 4;

      // return the maximum of two numbers
      System.out.println("Maximum number of x
                  and y is: " +Math.max(x, y));

      // return the square root of y
      System.out.println("Square root of y is: " + Math.sqrt(y));

      //returns 28 power of 4 i.e. 28*28*28*28
      System.out.println("Power of x and y is: " + Math.pow(x, y));

      // return the logarithm of given value
      System.out.println("Logarithm of x is: " + Math.log(x));
      System.out.println("Logarithm of y is: " + Math.log(y));

      // return the logarithm of given value when base is 10
      System.out.println("log10 of x is: " + Math.log10(x));
      System.out.println("log10 of y is: " + Math.log10(y));

      // return the log of x + 1
      System.out.println("log1p of x is: " +Math.log1p(x));

      // return a power of 2
      System.out.println("exp of a is: " +Math.exp(x));

      // return (a power of 2)-1
      System.out.println("expm1 of a is: " +Math.expm1(x));
   }
}
```

Check Program : Unit – 2 → MathRandomDemo1.java

Common mathematical methods

Marwadi University

```
public class JavaMathExample2
{
   public static void main(String[] args)
   {
      double a = 30;

      // converting values to radian
      double b = Math.toRadians(a);

      // return the trigonometric sine of a
      System.out.println("Sine value of a is: " +Math.sin(a));

      // return the trigonometric cosine value of a
      System.out.println("Cosine value of a is: " +Math.cos(a));

      // return the trigonometric tangent value of a
      System.out.println("Tangent value of a is: " +Math.tan(a));

      // return the trigonometric arc sine of a
      System.out.println("Sine value of a is: " +Math.asin(a));

      // return the trigonometric arc cosine value of a
      System.out.println("Cosine value of a is: "
+Math.acos(a));

      // return the trigonometric arc tangent value of a
      System.out.println("Tangent value of a is: "
+Math.atan(a));

      // return the hyperbolic sine of a
      System.out.println("Sine value of a is: " +Math.sinh(a));

      // return the hyperbolic cosine value of a
      System.out.println("Cosine value of a is: "
+Math.cosh(a));

      // return the hyperbolic tangent value of a
      System.out.println("Tangent value of a is: "
+Math.tanh(a));
   }
}
```

Check Program : Unit – 2 → MathRandomDemo2.java

Common mathematical methods

Marwadi University

```
public class MathRandomDemo3 {
 public static void main(String args[]) {
int i1 = 27;
  int i2 = -45;
System.out.println("Absolute value of i1: " + Math.abs(i1));

  System.out.println("Absolute value of i2: " + Math.abs(i2));

double d1 = 84.6;
  double d2 = 0.45;
  System.out.println("Round off for d1: " + Math.round(d1));

  System.out.println("Round off for d2: " + Math.round(d2));

System.out.println("Ceiling of '" + d1 + "' = " + Math.ceil(d1));
System.out.println("Ceiling of '" + d2 + "' = " + Math.ceil(d2));
 System.out.println("Floor of '" + d1 + "' = " +
Math.floor(d1));
 System.out.println("Floor of '" + d1 + "' = " +
Math.floor(d2));

System.out.println("exp(" + d2 + ") = " + Math.exp(d2));

  System.out.println("log(" + d2 + ") = " + Math.log(d2));

  System.out.println("pow(5, 3) = " + Math.pow(5.0, 3.0));

  System.out.println("sqrt(16) = " + Math.sqrt(16));
}
```

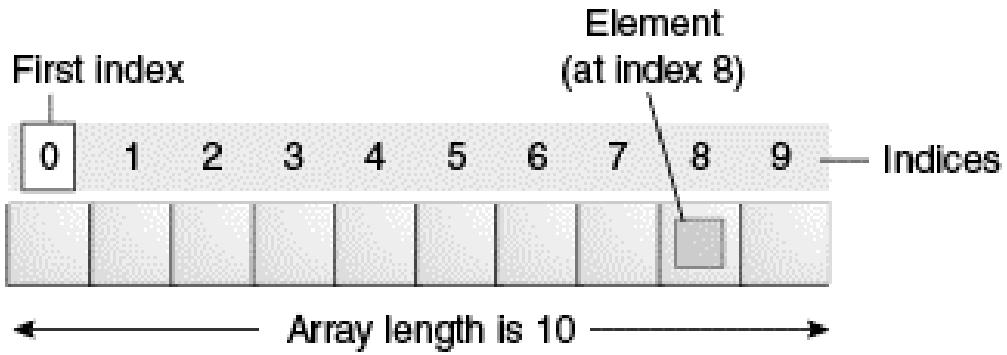
```
System.out.println("Minimum out of '" + i1 + "' and '" + i2 + "' = " + Math.min(i1, i2));

  System.out.println("Maximum out of '" + i1 + "' and '" + i2 + "' = " + Math.max(i1, i2));
```

Check Program : Unit – 2 → MathRandomDemo3.java

Array

- **An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.**
- **Java array is an object which contains elements of a similar data type.**
- **Benefits**
  - Code Optimization. Easier access to any element using the index.
  - Random access.
- **Disadvantages**
  - Fixed size. Can not be increased or decrease once declared.
  - Can store a single type of primitives only.
  - To delete an element in an array we need to traverse through out the array so this will reduce performance.

Array



Single Dimensional Array

Marwadi University

- A single array variable can reference a large collection of data.
- Once an array is created, its size is fixed. An array reference variable is used to access the elements in an array using an index.
- Instead of declaring individual variables, such as number0, number1, . . . , and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], . . . , and numbers[99] to represent individual variables.

Declaring 1-D Array Variables

- Syntax:

  datatype[ ] arr_ref_var;

  OR

  datatype arr_ref_var[ ]; **OR** datatype [ ]arr_ref_var;

- Ex. double myList[ ];
- Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array.
- you can create an array by using the new operator and assign its reference to the variable.

  Syntax:

  Arr_ref_var = new datatype [size];

Declaring 1-D Array Variables

Marwadi University

- Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement as:
  - datatype[ ] arr_ref_var = new datatype [Size];

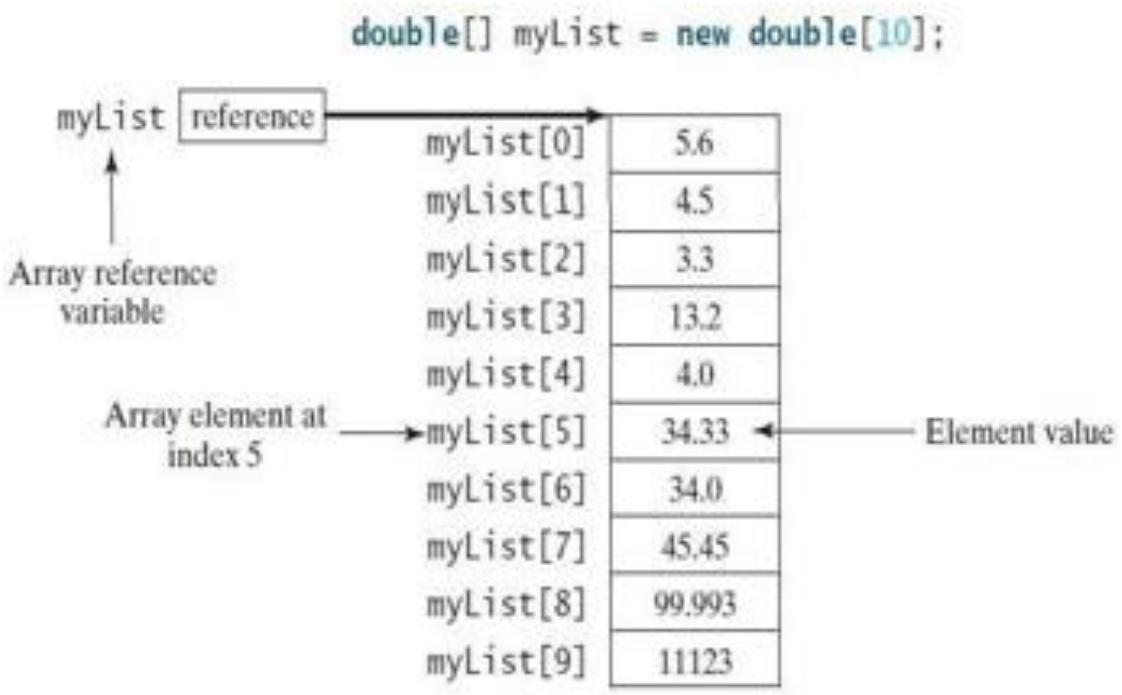
OR

  - datatype arr_ref_var[ ] = new datatype [Size];

- Ex. **double**[] myList = **new double**[**10**];

Assign values to Array

- Ex.
  - myList[**0**] = **5.6**;
  - myList[**1**] = **4.5**;
  - myList[**2**] = **3.3**;
  - myList[**3**] = **13.2**;
  - myList[**4**] = **4.0**;
  - myList[**5**] = **34.33**;
  - myList[**6**] = **34.0**;
  - myList[**7**] = **45.45**;
  - myList[**8**] = **99.993**;
  - myList[**9**] = **11123**;



Array size and default values

Marwadi University

- The size of an array cannot be changed after the array is created.
- Size can be obtained using arr_ref_var**.length**.
- For example, myList.length is 10.
- myList holds 10 double values, and the indices are from **0 to 9**.
- For example, myList[9] represents the **last** element in the array myList.
- Array Initializer:

  Syntax: elementType[] arrayRefVar = {value0, value1, ...,

  value*k*}; Example: double[] myList = {1.9, 2.9, 3.4, 3.5};

Processing Array


```
double[] myList = new double[10];
```

The following are some examples of processing arrays.

1. *Initializing arrays with input values:* The following loop initializes the array `myList` with user input values.

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();
```

2. *Initializing arrays with random values:* The following loop initializes the array `myList` with random values between `0.0` and `100.0`, but less than `100.0`.

```
for (int i = 0; i < myList.length; i++) {
  myList[i] = Math.random() * 100;
}
```

3. *Displaying arrays:* To print an array, you have to print each element in the array using a loop like the following:

```
for (int i = 0; i < myList.length; i++) {
  System.out.print(myList[i] + " ");
}
```

1D – Array Example

```
public class Array1D{
    public static void main(String args[])
     {
            double[ ] mylist = new double[10]; // Array declaration
             java.util.Scanner sc = new java.util.Scanner(System.in);

            System.out.println("Enter  " + mylist.length + " values ");
            //intialization array with input values
            for(int i=0;i<mylist.length;i++)
                  mylist[i] = sc.nextInt();

            //Display Array
            for(int i=0;i<mylist.length;i++)
                  System.out.print((int)mylist[i] + " ");
     }
}
```

For each Loop

Marwadi University

- **For** Java supports a convenient for loop, known as a foreach loop, which enables you to traverse the array sequentially without using an index variable.
- Syntax:

```
for (elementType element: arrayRefVar)
{
    // Process the element
}
```

- You can read the code as "for each element e in myList, do the following." If you wish to traverse in different order then you need index variable.

Ex.

```
for(double e : myList)
{
    System.out.println(e);
}
```

Two-Dimensional Array

- Data in a table or a matrix can be represented using a two- dimensional array.
- An element in a two-dimensional array is accessed through a row and column index.
- Syntax:

  elementType[ ][ ] arrayRefVar;

  OR

  elementType arrayRefVar[ ][ ]; **OR** elementType [ ][ ]arrayRefVar;

- Ex:

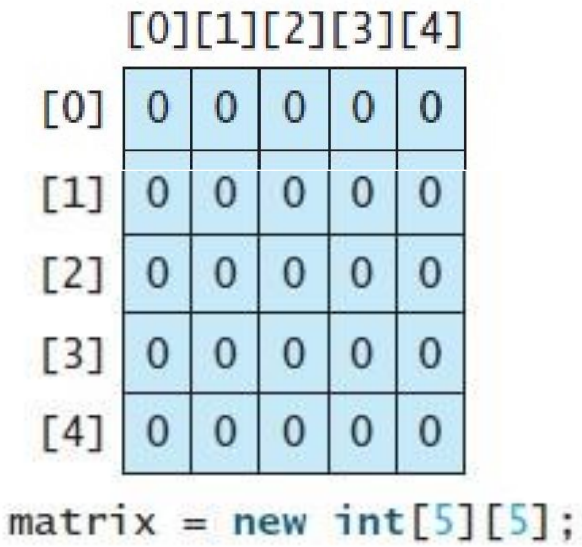
  int[ ][ ] matrix;

  OR

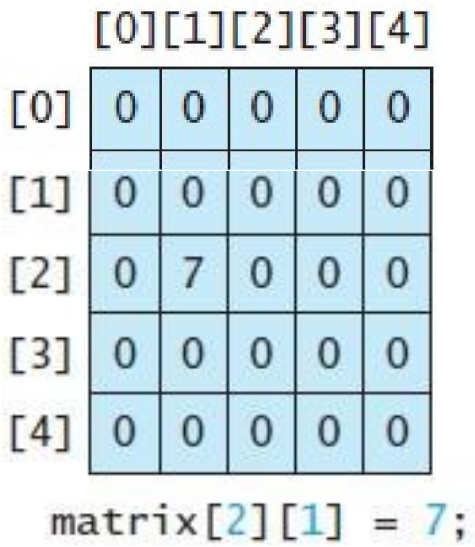
  int matrix[ ][ ];

Two-Dimensional Array

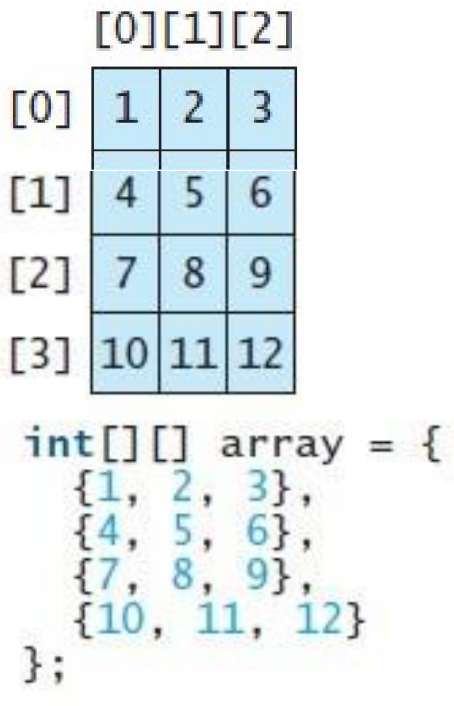
matrix = new int[5][5];



(a)



(b)



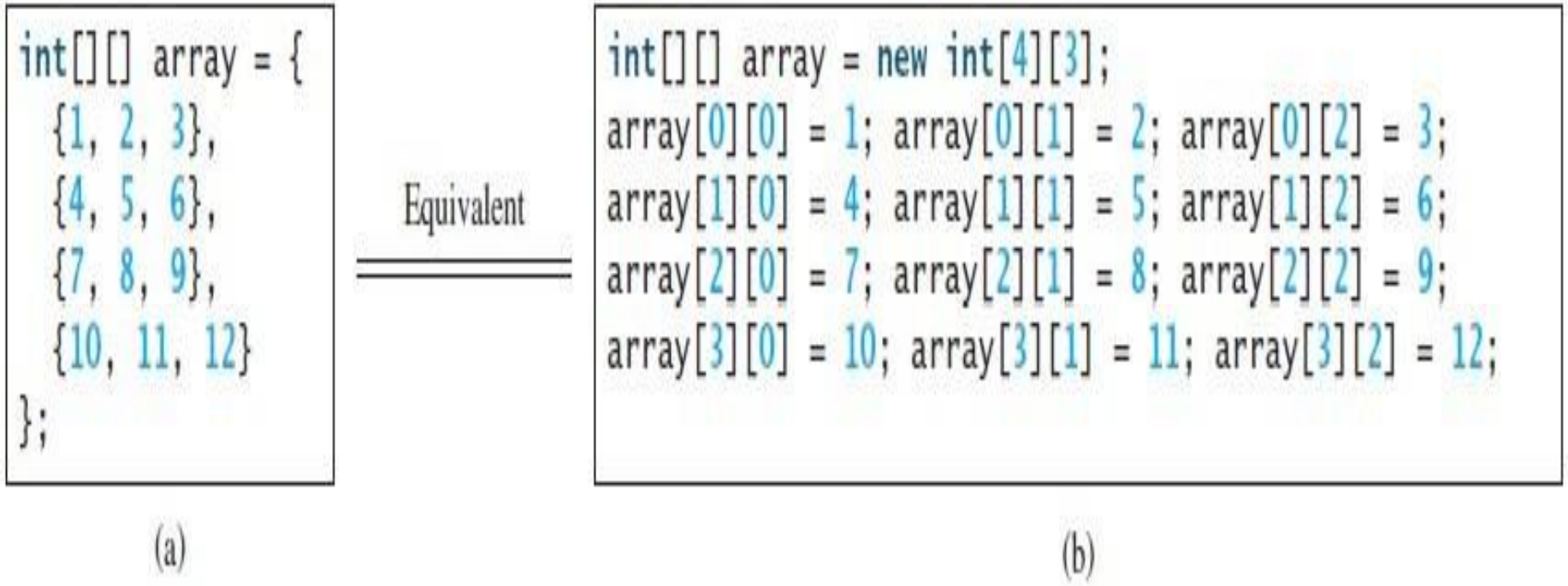
(c)

Two-Dimensional Array

Marwadi University

```
class Main
{
        public static void main(String args[])
        {
        int[][] a={{10,20},{30,40}};//declaration and initialization
        System.out.println("Two dimensional array elements are");
        System.out.println(a[0][0]);
        System.out.println(a[0][1]);
        System.out.println(a[1][0]);
        System.out.println(a[1][1]);
        }
}
```

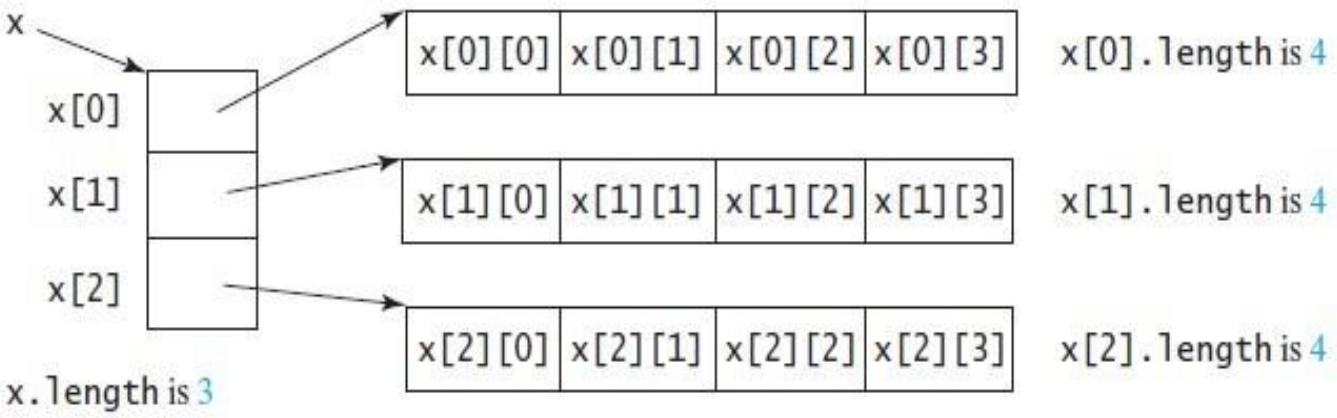
Two-Dimensional Array



Obtaining length of 2-D Array

Marwadi University

Example:
x = new int[3][4], x[0], x[1], and x[2] are one-dimensional arrays and each contains four elements. So, x.length is 3, and x[0].length, x[1].length, and x[2].length are 4.



Processing 2-D Arrays

Marwadi University

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

The following are some examples of processing two-dimensional arrays.

1. *Initializing arrays with input values.* The following loop initializes the array with user input values:

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
  matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

2. *Initializing arrays with random values.* The following loop initializes the array with random values between 0 and 99:

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

Processing 2-D Arrays

3. *Printing arrays.* To print a two-dimensional array, you have to print each element in the array using a loop like the following:

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    System.out.print(matrix[row][column] + " ");
  }

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

4. *Summing all elements.* Use a variable named `total` to store the sum. Initially `total` is `0`. Add each element in the array to `total` using a loop like this:

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
```

Passing 2-D Array to methods

Marwadi University

When passing a two-dimensional array to a method, the reference of the array is passed to the method.

```
import java.util.Scanner;

public class PassTwoDimensionalArray {
  public static void main(String[] args) {
    int[][] m = getArray(); // Get an array

    // Display sum of elements
    System.out.println("\nSum of all elements is " + sum(m));
  }

  public static int[][] getArray() {
    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // Enter array values
    int[][] m = new int[3][4];
    System.out.println("Enter " + m.length + " rows and "
      + m[0].length + " columns: ");
    for (int i = 0; i < m.length; i++)
      for (int j = 0; j < m[i].length; j++)
        m[i][j] = input.nextInt();

    return m;
  }

  public static int sum(int[][] m) {
    int total = 0;
    for (int row = 0; row < m.length; row++) {
      for (int column = 0; column < m[row].length; column++) {
        total += m[row][column];
      }
    }

    return total;
  }
}
```

2-D Array Example

```
class Main
{
        public static void main(String args[])
        {
        int[][] a={{10,20},{30,40},{50,60}};//declaration and initialization
        System.out.println("Two dimensional array elements are");
        for (int i = 0; i < 3; i++)
        {
    for (int j = 0; j < 2; j++)
          {
       System.out.println(a[i][j]);
         }
        }
        }
}
```

2-D Array to Method Example

Marwadi University

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



        int[][] a = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
        arr2method(a); // pass it to the method
    }

    public static void arr2method(int[][] a) {
        System.out.println("Elements are :");

        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                System.out.print(a[i][j] + "\t");
            }
            System.out.println("");
        }
    }
```

Multi Dimensional Array

A two-dimensional array consists of an array of one-dimensional arrays and a three dimensional array consists of an array of two- dimensional arrays.

```
double[][][] scores = new double[6][5][2];
```

You can also use the short-hand notation to create and initialize the array as follows:

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```

Multi Dimensional Array

Data_Type[][][] Name = new int[Tables][Row_Size][Column_Size];

Int [] [] [] sum = new int [2] [3] [4]

Tables: Total number of tables it can accept. 2D Array is always a single table with rows and columns. In contrast, Multi Dimensional array in Java is more than one table with rows and columns.

Row_Size: Number of Row elements. For example, Row_Size = 3, then the 3D array holds 3 rows.

Column_Size: Column elements it can store. Column_Size = 4, then the 3D array holds 4 Columns.

Multi Dimensional Array

Marwadi University

```
public class Main
{
   public static void main(String[] args) {
      //initialize 3-d array
      int[][][] intArray = { { { 1, 2, 3}, { 4, 5, 6 } ,  { 7, 8, 9 } } };
      System.out.println ("3-d array is given below :");
      //print the elements of array
      for (int i = 0; i < 1; i++)
         for (int j = 0; j < 3; j++)
         for (int z = 0; z < 3; z++)
            System.out.println (intArray [i][j][z]);
   }
}
```

Multi Dimensional Array

Marwadi University

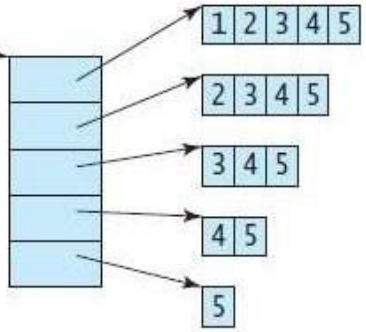
```
public class Main {
   public static void main(String[] args) {
      //initialize 3-d array
      int[][][] myArray = { { { 1, 2, 3 }, { 4, 5, 6 } },  { { 1, 4, 9 }, { 16, 25, 36 } },
            { { 1, 8, 27 }, { 64, 125, 216 } } };
      System.out.println("3x2x3 array is given below:");
      //print the 3-d array
      for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 2; j++) {
      for (int k = 0; k < 3; k++) {
      System.out.print(myArray[i][j][k] + "\t");
      }
      System.out.println();
               }
      System.out.println();
      }    }  }
```

Jagged Array

- Each row in a two-dimensional array is itself an array. Thus, the rows can have different lengths. An array of this kind is known as a **jagged array**.
- If you don't know the values in a jagged array in advance, but do know the sizes.

```
int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];

triangleArray[0][3] = 50;
triangleArray[4][0] = 45;
```

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



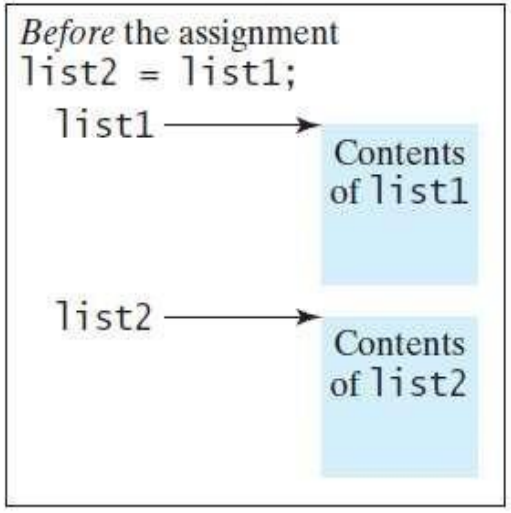
Copying Array

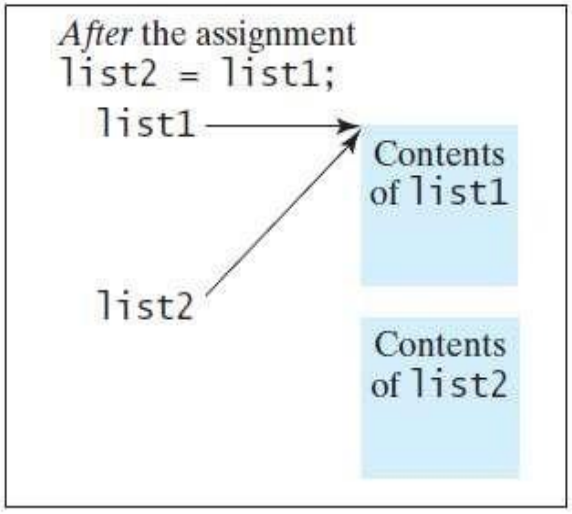
- To copy the contents of one array into another, you have to copy the array's individual elements into the other array.

Method : 1

```
list2 = list1;
```

- This statement does not copy the contents of the array referenced by list1 to list2, but instead it copies the reference value from **list1** to **list2**.





Copying Array

Marwadi University

- The other 3 ways to copy Array:

  Method 2: Use a loop to copy individual elements one by one.

  Method 3: Use the static arraycopy method in the System class.

  Method 4: Use the clone method to copy arrays

Method 2: Loop to copy individual element

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray . length];
for (int i = 0; i < sourceArray.length; i++)
{
    targetArray[i] = sourceArray[i];
}
```

Copying Array – Method 1

Marwadi University

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

        int [] arr1 = {1, 2, 3, 4, 5, 6};
        int [] arr2 = arr1;    // copying arrays

        // change value of first array
        arr1[0] = -1;

        // printing the second array
        for (int x: arr2) {
            System.out.print(x + ", ");
        // Check hashCode for arr1 and arr2 it must be same.
        }
    } }
```

Copying Array – Method 2

Marwadi University

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

      int [] source = {1, 2, 3, 4, 5, 6};
      int [] destination = new int[6];

      for (int i = 0; i < source.length; ++i) {
         destination[i] = source[i];
      }
      destination[0]= -1;
       for(int a: source) {System.out.print(a + " " ); }//Actual array

       System.out.println();
       for(int x: destination) {   System.out.print(x + " "); }//Copied
       System.out.println("\n" + source.hashCode());
       System.out.println(destination.hashCode());
   }}
```

Copying Array

Marwadi University

Method 3: Another approach is to use the **arraycopy** method in the **java.lang.System** class to copy arrays instead of using a loop.

Syntax: arraycopy(sourceArray, srcPos, targetArray, tarPos, length);

Ex: System.arraycopy(sourceArray, **0**, targetArray, **0**, sourceArray.length);

Method 4: Another approach is to use the **clone** method in the

Copying Array – Method 3

Marwadi University

```
//arraycopy
import java.util.Arrays;
class Main {
   public static void main(String[] args) {
      int[] n1 = {2, 3, 12, 4, 12, -2};

      // Creating n2 array of having length of n1 array
      int[] n2 = new int[n1.length];

      // copying entire n1 array to n2
      System.arraycopy(n1, 0, n2, 0, n1.length);
      System.out.println("n2 = " + Arrays.toString(n2));
   }
}
```

Copying Array – Method 4

Marwadi University

```
class Main {
    public static void main(String args[])     {
    int num_Array[] = {5,10,15,20,25,30};
     int clone_Array[] = num_Array.clone();

   System.out.println("Original num_Array:");
   for (int i = 0; i <num_Array.length; i++) {
          System.out.print(num_Array[i]+" ");
      }
      System.out.println();
      System.out.println("Cloned num_Array:");
     for (int i = 0; i <clone_Array.length; i++) {
          System.out.print(clone_Array[i]+" ");
      } } }
```

Jagged Array Example

```
//Java Program to illustrate the jagged array
class JaggedArrayDemo{
    public static void main(String[] args){
        //declaring a 2D array with odd columns
        int arr[][] = new int[3][];
        arr[0] = new int[3];
        arr[1] = new int[4];
        arr[2] = new int[2];
        //initializing a jagged array
        int count = 0;
        for (int i=0; i<arr.length; i++)
            for(int j=0; j<arr[i].length; j++)
                arr[i][j] = count++;

        //printing the data of a jagged array
        for (int i=0; i<arr.length; i++){
            for (int j=0; j<arr[i].length; j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();//new line
        }
    }
}
```

Jagged Array Example

```
class Main
{
   public static void main(String[] args)
   {
      // Declare a 2-D array with 3 rows
     int myarray[][] = new int[3][];

     // define and initialize jagged array

     myarray[0] = new int[]{1,2,3};
     myarray[1] = new int[]{4,5};
     myarray[2] = new int[]{6,7,8,9,10};

     // display the jagged array
     System.out.println("Two dimensional Jagged Array:");
     for (int i=0; i<myarray.length; i++)
     {
       for (int j=0; j<myarray[i].length; j++)
          System.out.print(myarray[i][j] + " ");
       System.out.println();
      }
   }
}
```

Command-Line Argument

Marwadi University

- Can a main method receive arguments?
- **Yes**
- The main method can receive string arguments from the command line.
- Write a program to make calculator using Command Line argument.

```
class Main{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

Summary

Marwadi University

- If statements
- Switch statement
- While statement
- For statement
- Do-while statement
- Break and continue keywords
- One dimensional and multidimensional arrays
- Jagged array
- Methods to copy array

END OF UNIT - 2