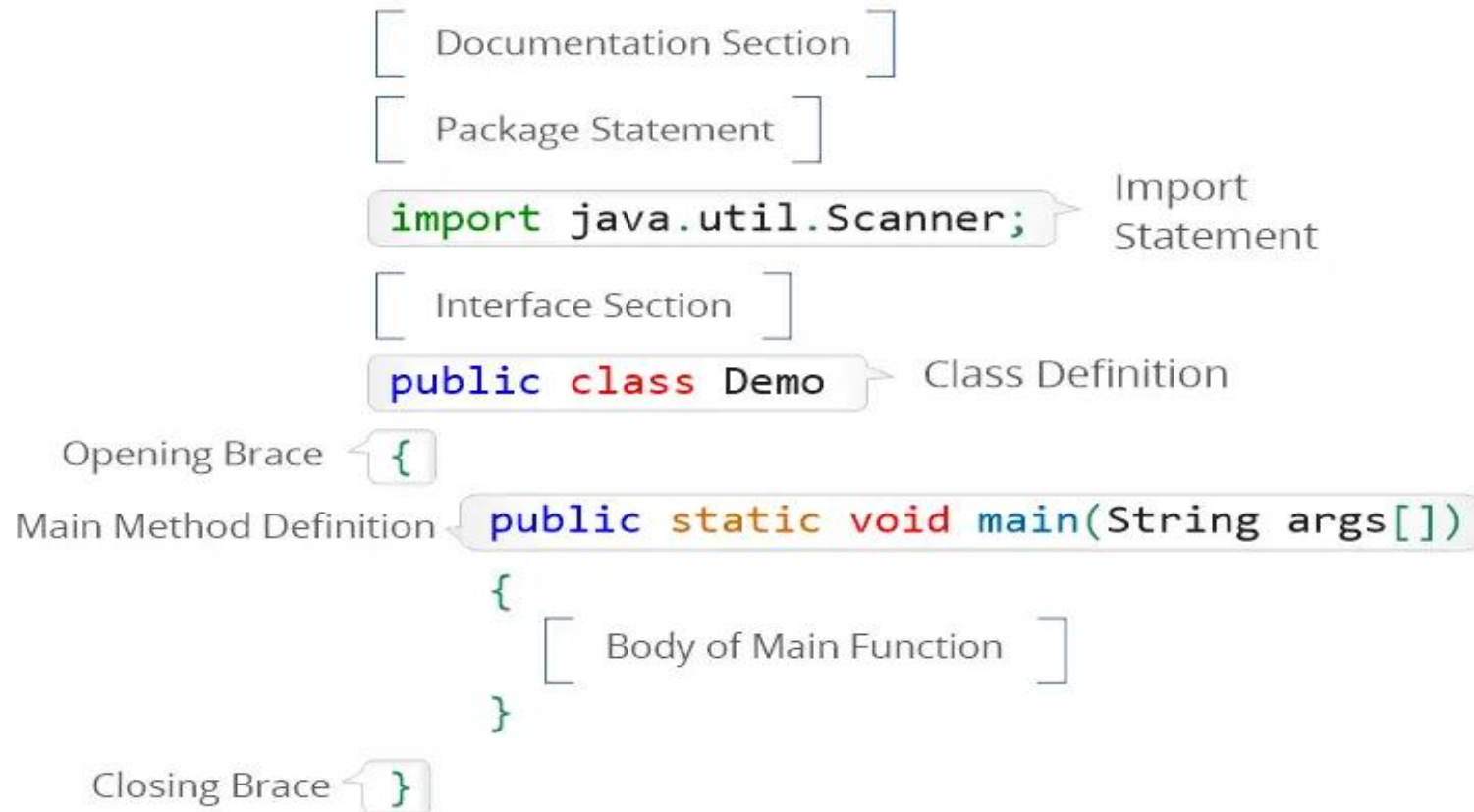


# **CA 105(A)**

# **Java Programming (Core Java)**

**UNIT-I An Introduction to Java**

# Basic Structure of a Java Program



The diagram illustrates the basic structure of a Java program with the following components and annotations:

- `[ Documentation Section ]`
- `[ Package Statement ]`
- `import java.util.Scanner;` (Import Statement)
- `[ Interface Section ]`
- `public class Demo` (Class Definition)
- `{` (Opening Brace)
- `public static void main(String args[])` (Main Method Definition)
- `{` (Opening Brace for Main Method Body)
- `[ Body of Main Function ]`
- `}` (Closing Brace for Main Method Body)
- `}` (Closing Brace for Class)

# Basic Structure of a Java Program

- **Example**
- This is how a simple basic structure of a Java program looks like.

```
public class MyFirstJavaProgram
{
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```

# Basic Structure of a Java Program

- Compile and run the program.
- Please follow the steps given below:
- Open notepad and add the *code* as above. Save the file as: MyFirstJavaProgram.java.
- Open a command prompt window and go o the directory where you saved the class.
- C :> javac MyFirstJavaProgram.java and press enter to compile your code.
- If there are no errors in your code, the command prompt will take you to the next line.
- C :> java MyFirstJavaProgram
- HelloWorld Output Displayed on Screen.

# Basic Structure of a Java Program

## Documentation Section

- It is used to improve the readability of the program. It consists of comments in Java which include basic information such as the method's usage or functionality to make it easier for the programmer to understand it while reviewing or debugging the code. This statement is optional.

## Package Statement

- There is a provision in Java that allows us to declare our classes in a collection called package. There can be only one package statement in a Java program and it has to be at the beginning of the code before any class or interface declaration. This statement is optional.

## Import Statement

- Many predefined classes are stored in packages in Java, an import statement is used to refer to the classes stored in other packages. An import statement is always written after the package statement but it has to be before any class declaration.

## Interface Section

- This section is used to specify an interface in Java. It is an optional section which is mainly used to implement Multiple Inheritance in Java. An interface is a lot similar to a class in Java but it contains only constants and method declarations.

# Basic Structure of a Java Program

## Class Definition

- A Java program may contain several class definitions, classes are an essential part of any Java program. Every program in Java will have at least one class with the main method. The class that contains the main method must be declared as public.

For example:

- `public class Test` - This creates a class called Test. You should make sure that the class name starts with a capital letter, and the public word means it is accessible from any other classes.

## Braces (Both Opening and Closing Brace)

- The curly braces are used to group all the commands together. To make sure that the commands belong to a class or a method.

# Basic Structure of a Java Program

- **Main Method Definition**

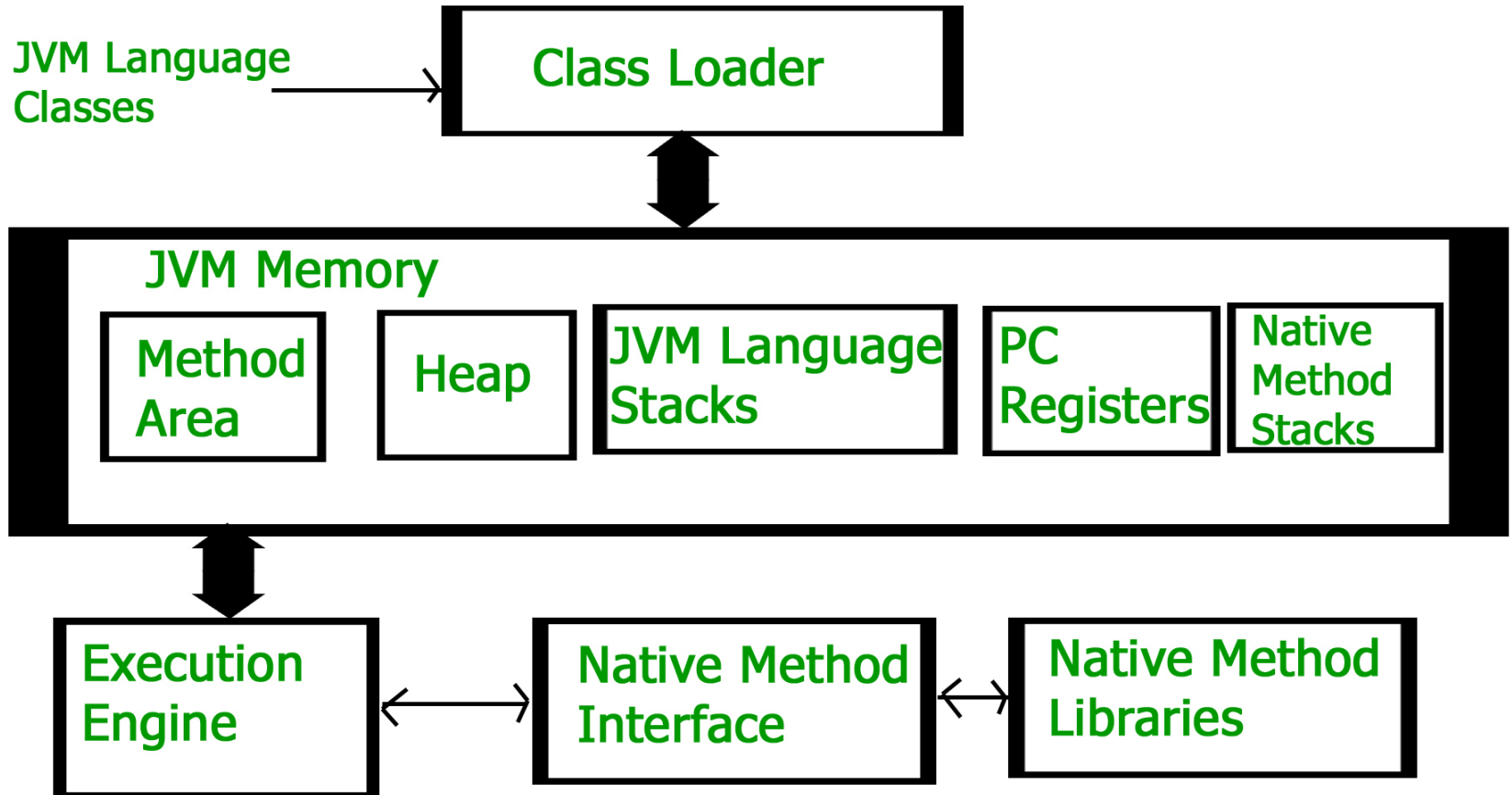
- In Java, the main method is treated as the entry point of the program, The main method is called by the operating system when the user runs the program.
- For Example: `public static void main(String args[])`
- **public** - When the main method is declared public, it means that it can be used outside of the declared class as well.
- **static** - The word static means that we want to access a method without making object of the class within which the method is declared. We call the main method without creating any objects.
- **void** - The word void indicates that it does not return any value. The main method is declared as void because it does not return any value.
- **main** - The main is the method, which is an essential part of any Java program.
- **String args[]** - It is an array where each element is a string, which is named as args. If you run the Java code through a console, you can pass the input parameter. The main() takes it as an input.

# The Java Virtual Machine

- JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the **main** method present in a java code. JVM is a part of JRE(Java Runtime Environment).
- Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.
- When we compile a *.java* file, *.class* files(contains byte-code) with the same class names present in *.java* file are generated by the Java compiler. This *.class* file goes into various steps when we run it. These steps together describe the whole JVM.



# The Java Virtual Machine



# The Java Virtual Machine

- **Class Loader Subsystem**

- It is mainly responsible for three activities.

1. Loading
2. Linking
3. Initialization

## **1. Loading:**

- The Class loader reads the “.class” file, generate the corresponding binary data and save it in the method area. For each “.class” file, JVM stores the following information in the method area.
- The fully qualified name of the loaded class and its immediate parent class.
- Whether the “.class” file is related to Class or Interface or Enum.
- **Modifier, Variables and Method information etc.**
- After loading the “.class” file, JVM creates an object of type Class to represent this file in the heap memory.

# The Java Virtual Machine

- **2. Linking:** Performs verification, preparation, and (optionally) resolution.
- ***Verification:***
  - It ensures the correctness of the *.class* file i.e. it checks whether this file is properly formatted and generated by a valid compiler or not. If verification fails, we get run-time exception *java.lang.VerifyError*.
  - This activity is done by the component *ByteCodeVerifier*. Once this activity is completed then the class file is ready for compilation.
- ***Preparation:***
  - JVM allocates memory for class static variables and initializing the memory to default values.
- ***Resolution:***
  - It is the process of replacing symbolic references from the type with direct references. It is done by searching into the method area to locate the referenced entity.

# Initialization:

## 3) Initialization: -

- In this phase, all static variables are assigned with their values defined in the code and static block(if any).
- This is executed from top to bottom in a class and from parent to child in the class hierarchy.

# Java Comments

- **Java Comments:-**
- Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.
- **1) Single-line Comments**
  - Single-line comments start with two forward slashes (//).
  - Any text between // and the end of the line is ignored by Java (will not be executed).
- Example:-
- **// This is a comment**
- `System.out.println("Hello World");`
- **2) Java Multi-line Comments**
  - Multi-line comments start with /\* and ends with \*/.
  - Any text between /\* and \*/ will be ignored by Java.
- **Example**
- `/* The code below will print the words Hello World to the screen,`
- `and it is amazing */`
- `System.out.println("Hello World");`
- **3) Java Documentation Comment**
  - Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API.
  - These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.
  - The documentation comments are placed between /\*\* and \*/.

# Data types

- Data types specify the different sizes and values that can be stored in the variable. There are **two types** of data types in Java:
- **1) Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **2) Non-primitive data types:** The non-primitive data types include [Classes](#), [Interfaces](#), and [Arrays](#).

## ➤ Variable

- A variable is the name of a reserved area allocated in memory.
- In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

### Types of Variables

- There are three types of variables in [Java](#):
- 1) local variable
- 2) instance variable
- 3) static variable

# Variable

- **1) Local Variable**

- A variable declared inside the body of the method is called local variable.
- You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

- **2) Instance Variable**

- A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](#).
- It is called an instance variable because its value is instance-specific and is not shared among instances.

- **3) Static variable**

- A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class.
- Memory allocation for static variables happens only once when the class is loaded in the memory.

# Keywords

- Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code.
- These are predefined words by Java so they cannot be used as a variable or object name or class name.

## List of Java Keywords

boolean	byte	char	double	float
short	void	int	long	while
for	do	switch	break	continue
case	default	if	else	try
catch	finally	class	abstract	extends
final	import	new	instance of	private
interface	native	public	package	implements
protected	return	static	super	synchronized
this	throw	throws	transient	volatile



# Java Control Structures

- Java compiler executes the code from top to bottom.
- The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code.
- Such statements are called control flow statements.
- Java provides **three types of control flow statements**.
  - **1) Decision Making statements**
    - if statements
    - switch statement
  - **2) Loop statements**
    - do while loop
    - while loop
    - for loop
    - for-each loop
  - **3) Jump statements**
    - break statement
    - continue statement

# Java Control Structures

## ➤ 1) Decision Making statements

➤ "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

- 1. Simple if statement
- 2. if-else statement
- 3. if-else-if ladder
- 4. Nested if-statement

- **1. Simple if statement:-**

- It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

- **Syntax** of if statement is given below.

- **if**(condition) {
- statement 1; //executes when condition is true
- }

- **Example:-**

- **public class** Student {
- **public static void** main(String[] args) {
- **int** x = 10;
- **int** y = 12;
- **if**(x+y > 20) {
- System.out.println("x + y is greater than 20");
- }
- }
- }

- **Output:**

- x + y is greater than 20

-

- **2) if-else statement**
- The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

- **Syntax:**

- `if(condition) {`
- `statement 1; //executes when condition is true`
- `}`
- `else{`
- `statement 2; //executes when condition is false`
- `}`

- **Consider the following example.**

- `public class Student {`
- `public static void main(String[] args) {`
- `int x = 10;`
- `int y = 12;`
- `if(x+y < 10) {`
- `System.out.println("x + y is less than 10");`
- `} else {`
- `System.out.println("x + y is greater than 20");`
- `}`
- `}`
- `}`

- **Output:**

- `x + y is greater than 20`

.

- **3) if-else-if ladder:**
- The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.
- **Syntax of if-else-if statement is given below.**
- **if**(condition 1) {
- statement 1; //executes when condition 1 is true
- }
- **else if**(condition 2) {
- statement 2; //executes when condition 2 is true
- }
- **else** {
- statement 2; //executes when all the conditions are false
- }
- **Consider the following example.:-**
- **public class** Student {
- **public static void** main(String[] args) {
- String city = "Delhi";
- **if**(city == "Meerut") {
- System.out.println("city is meerut");
- }**else if** (city == "Noida") {
- System.out.println("city is noida");
- }**else if**(city == "Agra") {
- System.out.println("city is agra");
- }**else** {
- System.out.println(city);
- }
- }
- }
- }
- **Output:** Delhi
-

- **4. Nested if-statement**

- In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

- **Syntax of Nested if-statement is given below.**

- **if**(condition 1) {
- statement 1; //executes when condition 1 is true
- **if**(condition 2) {
- statement 2; //executes when condition 2 is true
- }
- **else**{ statement 2; //executes when condition 2 is false } }

- **Consider the following example.:-**

- **public class** Student {
- **public static void** main(String[] args) {
- String address = "Delhi, India";
- 
- **if**(address.endsWith("India")) {
- **if**(address.contains("Meerut")) {
- System.out.println("Your city is Meerut");
- **}else if**(address.contains("Noida")) {
- System.out.println("Your city is Noida");
- **}else {**
- System.out.println(address.split(",")[0]);
- }
- **}else {**
- System.out.println("You are not living in India"); } } } **Output:** Delhi

- **Switch Statement:-**
- In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements.
- **The syntax to use the switch statement is given below.**
- **switch** (expression){
- **case** value1:
- statement1;
- **break**;
- .
- .
- .
- **case** valueN:
- statementN;
- **break**;
- **default**:
- **default** statement;
- }

**Consider the following example to understand the flow of the switch statement.**

```
public class Student implements Cloneable {  
public static void main(String[] args) {  
    int num = 2;  
    switch (num){  
    case 0:  
        System.out.println("number is 0");  
        break;  
    case 1:  
        System.out.println("number is 1");  
        break;  
    default:  
        System.out.println(num);  
    }  
    }  
}
```

**Output: 2**

# Java Control Structures

## ➤ 1) Loop statements

- In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.
  - However, loop statements are used to execute the set of instructions in a repeated order.
  - The execution of the set of instructions depends upon a particular condition.
- In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.
    1. for loop
    2. while loop
    3. do-while loop



## 1. for loop:-

- In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
- We use the for loop only when we exactly know the number of times, we want to execute the block of code.
- **for**(initialization, condition, increment/decrement) {
- //block of statements
- }
- **Example:-**
- **public class** Calculation {
- **public static void** main(String[] args) {
- // TODO Auto-generated method stub
- **int** sum = 0;
- **for(int** j = 1; j<=10; j++) {
- sum = sum + j;
- }
- System.out.println("The sum of first 10 natural numbers is " + sum);
- }
- }
- **Output:-**The sum of first 10 natural numbers is 55

## 2. While loop:-

- The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop.
- Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.
- It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

- The **syntax** of the while loop is given below.

- **while**(condition){

- //looping statements

- }

- **Example:-Calculation .java**

- **public class** Calculation {

- **public static void** main(String[] args) {

- // TODO Auto-generated method stub

- **int** i = 0;

- System.out.println("Printing the list of first 10 even numbers \n");

- **while**(i<=10) {

- System.out.println(i);

- i = i + 2;

- }

- }

- }

- **Output:**

- Printing the list of first 10 even numbers

- 0

- 2

- 4

- 6

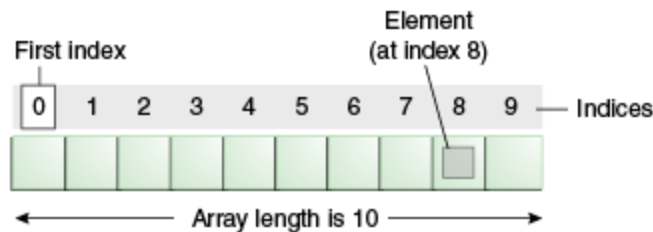
- 8

### 3. do-while loop:-

- The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.
- The **syntax** of the do-while loop.
  - **do**
  - {
  - //statements
  - } **while** (condition);
- **Example:- Calculation.java**
- **public class** Calculation {
- **public static void** main(String[] args) {
- // TODO Auto-generated method stub
- **int** i = 0;
- System.out.println("Printing the list of first 10 even numbers \n");
- **do** {
- System.out.println(i);
- i = i + 2;
- }**while**(i<=10);
- }
- }
- **Output:-**
- Printing the list of first 10 even numbers
- 0
- 2
- 4
- 6
- 8
- 10

# Arrays

- An array is a **collection** of **similar type of elements** which has **contiguous** memory location.
- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the **first** element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- We can store primitive values or objects in an array in Java.
- Like C/C++, we can also create **single** dimensional or **multidimensional** arrays in Java.



## Advantages:-

**Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.

**Random access:** We can get any data located at an index position.

## Disadvantages

**Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

- There are **two types** of array.
- 1. Single Dimensional Array
- 2. Multidimensional Array
- Single Dimensional Array in Java
- **Syntax to Declare an Array in Java**
- dataType[] arr; (or)
- dataType []arr; (or)
- dataType arr[];
- **Instantiation of an Array in Java**
- arrayRefVar=**new** datatype[size];
- **Example of Java Array**
- Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.
- //Java Program to illustrate how to declare, instantiate, initialize
- //and traverse the Java array.
- **class** Testarray{
- **public static void** main(String args[]){
- **int** a[]=**new int**[5];//declaration and instantiation
- a[0]=10;//initialization
- a[1]=20;
- a[2]=70;
- a[3]=40;
- a[4]=50;
- //traversing array
- **for**(**int** i=0;i<a.length;i++)//length is the property of array
- System.out.println(a[i]);
- }}

Output:

10  
20  
70  
40  
50

- **Declaration, Instantiation and Initialization of Java Array:-**
- We can declare, instantiate and initialize the java array together by:
- **int a[]={33,3,4,5};**//declaration, instantiation and initialization
- Let's see the simple example to print this array.
- **//Java Program to illustrate the use of declaration, instantiation**
- **//and initialization of Java array in a single line**
- **class Testarray1{**
- **public static void** main(String args[]){
- **int a[]={33,3,4,5};**//declaration, instantiation and initialization
- **//printing array**
- **for(int i=0;i<a.length;i++)**//length is the property of array
- **System.out.println(a[i]);**      **Output:**
- **}}**      33
- 3
- 4
- 5

## 2) Multidimensional Array in Java:-

- In such case, data is stored in row and column based index (also known as matrix form).
- **Syntax to Declare Multidimensional Array in Java**
- `dataType[][] arrayRefVar; (or)`
- `dataType [][]arrayRefVar; (or)`
- `dataType arrayRefVar[][]; (or)`
- `dataType []arrayRefVar[];`
- **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 Java Array

- Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

//Java Program to illustrate the use of multidimensional array

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
    for(int j=0;j<3;j++){
        System.out.print(arr[i][j]+" ");
    }
    System.out.println();
}
}}
```

**Output:**

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



### Jagged Array:-

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

- `//Java Program to illustrate the jagged array`
- `class TestJaggedArray{`
- `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`
- `}`
- `}`
- `}`

### Output:

```
0 1 2
3 4 5 6
7 8
```