Introduction to java:

Introduction:

- **Java** is a general-purpose, class-based, object-oriented programming language
- Java is a multi-platform, object-oriented, and network-centric language.
- It is among the most used programming language. Java is also used as a computing platform.
- A general-purpose programming language made for developers to *write once run anywhere* that is compiled Java code can run on all platforms that support Java.
- It is designed to have as few implementation dependencies as possible.
- It is considered as one of the fast, secure, and reliable programming languages preferred by most organizations to build their projects.
- It is widely used for developing Java applications in laptops, data centers, game consoles, scientific supercomputers, cell phones, etc.

Why Use Java?

- Java works on different platforms such as Windows, Mac, Linux, Raspberry Pi, etc.
- It is one of the most popular programming language in the world.
- It is easy to learn and simple to use.
- It is open-source and free.
- It is secure, fast and powerful.
- It has a huge community support i.e. tens of millions of developers.
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa.

History:

Here are important landmarks from the history of the Java language:

- James Gosling who is known as the Father of Java developed the Java platform at Sun Microsystems, and the Oracle Corporation later acquired it.

- The Java language was initially called OAK.
- Originally, it was developed for handling portable devices and set-top boxes. Oak was a massive failure.
- In 1995, Sun changed the name to "Java" and modified the language to take advantage of the burgeoning www (World Wide Web) development business.
- Later, in 2009, Oracle Corporation acquired Sun Microsystems and took ownership of three key Sun software assets: Java, MySQL, and Solaris.

Features of Java

The most important features of the Java language are given below.

1. Platform Independent: Compiler converts source code to bytecode and then the JVM executes the bytecode generated by the compiler. This bytecode can run on any platform be it Windows, Linux, macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of bytecode. Hence java language is a platform-independent language.

2. Object-Oriented Programming Language: Organizing the program in the terms of collection of objects is a way of object-oriented programming, each of which represents an instance of the class. The four main concepts of Object-Oriented programming are:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

3. Simple: Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, and explicit memory allocation.

4. Robust: Java language is robust that means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, therefore the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.

5. Secure: In java, we don't have pointers, and so we cannot access out-of-bound arrays i.e it shows **ArrayIndexOutOfBoundsException** if we try to do so. That's why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

6. Distributed: user can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating

distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

7. Multithreading: Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

8. Portable: As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

Components Of Java Programming Language

Java platform is a software or collection of programs that help us to execute applications written in Java programming language. A Java platform consists of a Java compiler, a set of libraries, and an execution engine.
Java platform is independent of any particular OS which makes Java programming language a platform-independent language.

Java platform consists of the following components.

- Java language
- The Java Development Kit (JDK)
- The Java Runtime Environment (JRE)
- The Java Compiler
- The Java Virtual Machine (JVM)

Java Development kit (JDK)

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. Java developers can use it on Windows, macOS, Solaris, and Linux. JDK helps them to code and run Java programs. It is possible to install more than one JDK version on the same computer.

Why use JDK?

Here are the main reasons for using JDK:

- JDK contains tools required to write Java programs and JRE to execute them.
- It includes a compiler, Java application launcher, Appletviewer, etc.
- Compiler converts code written in Java into bytecode.
- Java application launcher opens a JRE, loads the necessary class, and executes its main method.

Java Virtual Machine (JVM):

Java Virtual Machine (JVM) is an engine that provides a runtime environment to drive the Java Code or applications. It converts Java bytecode into machine language. JVM is a part of the Java Run Environment (JRE). In other programming languages, the compiler produces machine code for a particular system. However, the Java compiler produces code for a Virtual Machine known as Java Virtual Machine.

Java Runtime Environment (JRE)

JRE is a piece of software that is designed to run other software. It contains the class libraries, loader class, and JVM. In simple terms, if you want to run a Java program, you need JRE. If you are not a programmer, you don't need to install JDK, but just JRE to run Java programs.

Why use JRE?

Here are the main reasons of using JRE:

- JRE contains class libraries, JVM, and other supporting files. It does not include any tool for Java development like a debugger, compiler, etc.
- It uses important package classes like math, swing, util, lang, awt, and runtime libraries.
- If users have to run Java applets, then JRE must be installed in your system.

Types of Java Applications

Java applications has to classified into

- Standalone applications
- Web applications
- Enterprise applications
- Mobile applications

Standalone Applications

A program is run by as separate computer process without adding an existing files process is known as standalone application. It is also known as desktop applications. AWT (Abstract Window Toolkit) and swing are used to create stand alone application.

Web Applications

Web application is a client server software application which is run by the client. Web application includes mail, online retail sales, Wikipedia information's, Servlet, jsp, jsf which are used to create web applications.

Enterprise Application

Enterprise application is related to middle ware applications. To enable software application and hardware systems we are using technologies and services across the enterprises. So we are calling it as enterprise application. Enterprise applications are mainly designed for corporate sides such as banking business systems.

Mobile Application

Mobile application is developed for run the mobile phones and tablets.

Types of Java Platforms

There are four different types of Java programing language platforms:

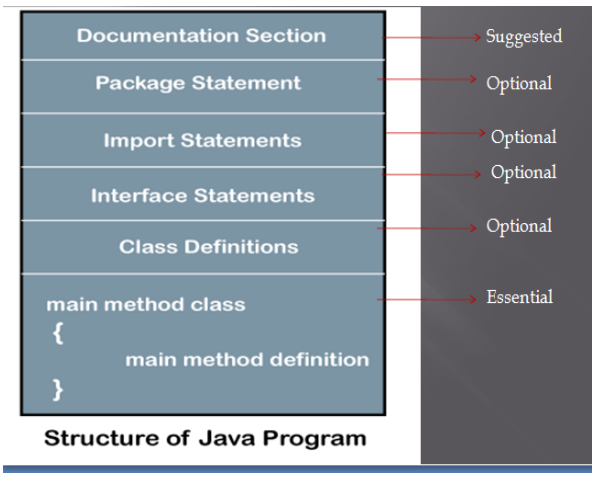
1. Java Platform, Standard Edition (Java SE): Java SE's API offers the Java programming language's core functionality. It defines all the basis of type and object to high-level classes. It is used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.

2. Java Platform, Enterprise Edition (Java EE): The Java EE platform offers an API and runtime environment for developing and running highly scalable, large-scale, multi-tiered, reliable, and secure network applications.

3. Java Programming Language Platform, Micro Edition (Java ME): The Java ME platform offers an API and a small-footprint virtual machine running Java programming language applications on small devices, like mobile phones.

4. Java FX: JavaFX is a platform for developing rich internet applications using a lightweight user-interface API. It user hardware-accelerated graphics and media engines that help Java take advantage of higher-performance clients and a modern look-and-feel and high-level APIs for connecting to networked data sources.

Structure of Java Program



- **Documentation Section:**

The documentation section is an important section but optional for a Java program. It includes basic information about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program. It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, comments are used. The comments may be single-line, multi-line, and documentation comments.

- **Single-line Comment:** It starts with a pair of forwarding slash **(//)**.
  For example:
  //First Java Program
- **Multi-line Comment:** It starts with a **/*** and ends with ***/.** Multiline comment write between these two symbols.
  For example:

/*It is an example of
multiline comment*/

- **Documentation comments**: The JDK javadoc tool uses *doc comments* when preparing automatically generated documentation. Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format.

  For example

/ documentation */**

This is a documentation comment and in general it is called doc comment.

- **Package Declaration:** The package declaration is optional. It is placed just after the documentation section.The first statement allowed in java file is a package statement.Note that there can be only one package statement in a Java program. It must be defined before any class and interface declaration. keyword package is used to declare the package name and informs compiler that classes are defined here belongs to this packages.

For e.g Package student;

- **Interface Section:** An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when user wish to implement the multiple inheritance feature in the program. It contains only constants and method declarations.

For e.g. **interface** car

```
{
void start();
void stop();
}
```

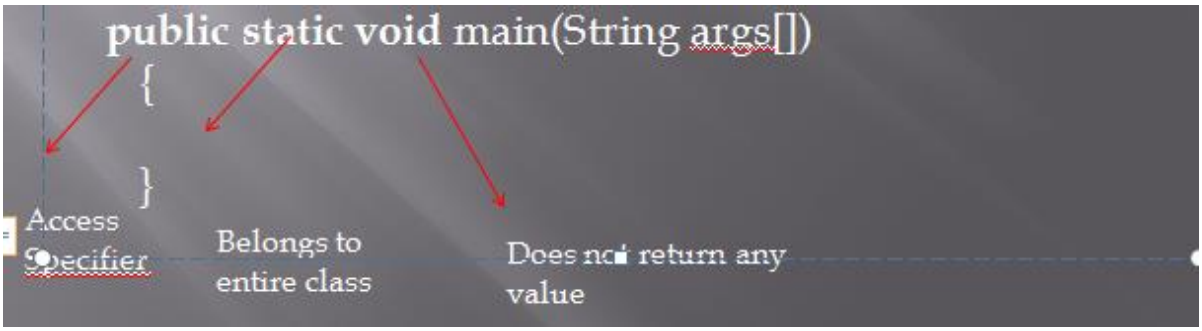
- **Class Definition:** A Java program may contains multiple class definitions. Classes are the primary and essential elements of a java program. These classes are used to map the objects of real world problems.class keyword is used to define the class

For example

```
class Student   //class definition
        {
         variables section
        Methods section
        }
```

- **Main Method Class:** In this section, user define the main() method. It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, user create objects and call the methods.

Following statement is used to define the main() method:



The requirement for Java Program

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- Create the java program
- Compile and run the java program
- Create a program using any text editor.

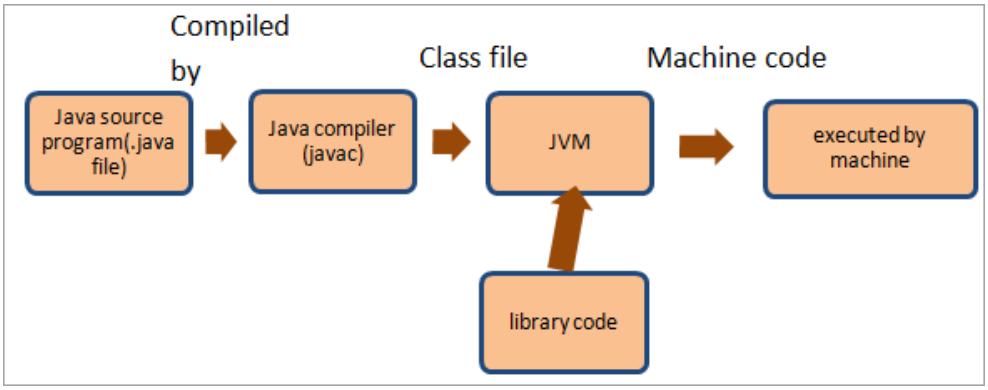
  For example in notepad type following program

```
class Test
{
   public static void main(String args[])
{
    System.out.println("Hello !");
  System.out.println("Welcomes to the world of Java");
  System.out.println("Let us learn Java");
   }
}
```

Steps to Compile and Run Program

- First save this file as Test.java
- To compile give command in cmd window javac Test.java
- To execute the program give the command java Test in cmd window

The following diagram shows the flow of a Java program.



Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** − Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

  For Example: *class MyFirstJavaClass*

- **Method Names** − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example: *public void myMethodName()*

- **Program File Name** − Name of the program file should exactly match the class name.

Java Tokens: Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Tokens are the smallest individual units in a program. In simple terms, a Java program is a collection of tokens, comments and white spaces. Java language includes following types of tokens. They are:

- Keywords
- Identifiers
- Literals
- Operators.
- Separators

Keywords: These are the pre-defined reserved words of any programming language. Each keyword has a special meaning. It is always written in lower case. For e.g. else , float etc.

Identifiers: Identifiers are programmer-designed tokens. They are used for naming variables, objects, labels, packages and interfaces in a program.

Java identifiers follow the following rules:

1. They can have alphabets, digits, and the underscore and dollar sign
2. They must not begin with a digit.
3. Uppercase and lowercase letters are distinct.
4. They can be of any length.

For e.g. average, Student

Literals: Literals in Java are a sequence of characters i.e. digits, letters, and other characters that represent constant values to be stored in variables. Java language specifies five major types of literals. They are:**:**

- Integer literals
- Floating point literals
- Character literals
- String literals
- Boolean literals

for e.g. 23 is integer literal

9.45 is floating point literal

True is a Boolean literal

'a' is character literal and

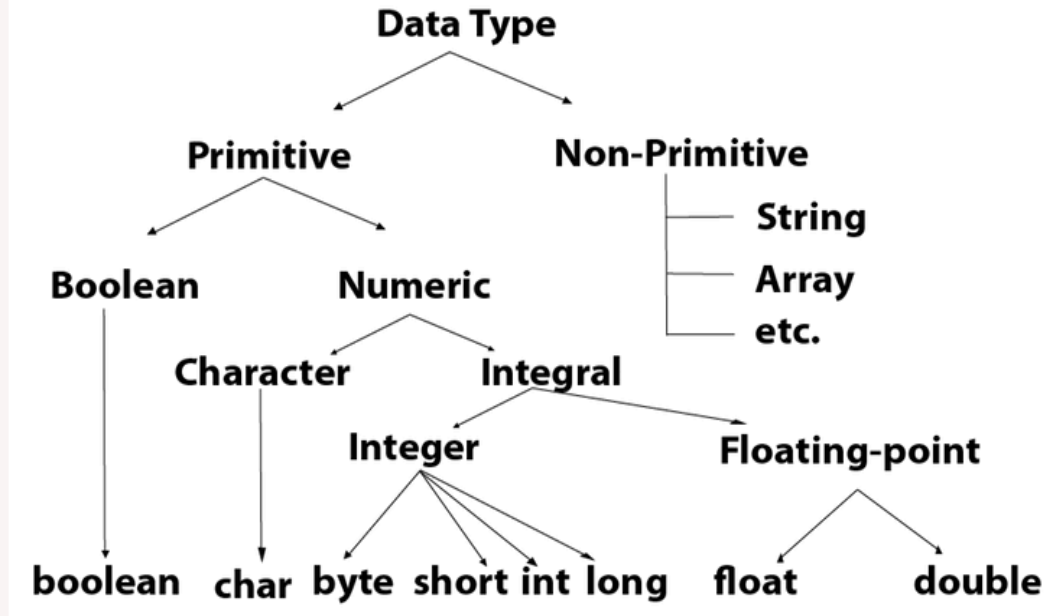
"BCA" is string literal

Data Types in Java

Every variable in Java has a data type. Data types specify the size and type of values that can be stored. Java language is rich in its data types. The variety of data types available allows the programmer to select the type appropriate to the needs of the application. There are two types of data :

- **Primitive data types:** It is also called intrinsic data type.The primitive data types also classified into Numeric includes byte, short, int, long, float and double and Non – Numeric include Boolean and char.

- **Non-primitive data types:** It is also called derived data type. The non-primitive data types include Classes, Interfaces, and Arrays.



Java Primitive Data Types: In Java language, primitive data types are the building blocks of data manipulation. There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Numeric Data types

1. **Byte:** The byte data type is an 8-bit signed integer. The byte data type is useful for saving memory in large arrays.
   Syntax:
   byte byteVar;
   Example
   byte myNum = 100;
   System.out.println(myNum);
   Size: 1 byte (8 bits)
   Values: -128 to 127
   Default Value: 0
2. **Short:**The short data type is a 16-bit signed integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters.

Syntax: short shortVar;

Example

short myNum = 5000;

System.out.println(myNum);

Size: 2 byte (16 bits)

Values: -32, 768 to 32, 767 (inclusive)

Default Value: 0

3. **Int**: It is a 32-bit signed integer.

Syntax:

int intVar;

Example

int myNum = 100000;

System.out.println(myNum);

Size: 4 byte (32 bits)

Values: -2, 147, 483, 648 to 2, 147, 483, 647 (inclusive)

Default Value: 0

4. **Long:**The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

Example

long myNum = 15000000000L;

System.out.println(myNum);

5. **Floating Point Types:**

Float : The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that end the value with an "F"

Example

float myNum = 5.75F;

Double: The double data type can store fractional numbers from 1.7e−308 to 1.7e+308. Note that you should end the value with a "d"

Example

double myNum = 19.99d;

System.out.println(myNum);

Non-Numeric Data Type

1. **Booleans**: A boolean data type is declared with the boolean keyword and can only take the values true or false:

Example

boolean t = true;

boolean f = false;

2. **Characters:** The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';
```

3. **Strings:**The String data type is used to store a sequence of characters i.e.text. String values must be surrounded by double quotes:

Example

```
String greeting = "Hello World";
```

Data type,Size &Description

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Variable in Java:

A variable is an identifier that denotes a storage location used to store a data value. The general rules for constructing names for variables are:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with $ and _
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as int or boolean) cannot be used as names

Declaration of Variables:

To create a variable, you must specify the type and assign it a value:

Syntax

```
type variable = value;
```

Where *type* is one of Java's types such as int or String, and

variable is the name of the variable such as **x** or name. The equal sign is used to assign values to the variable.

To declare more than one variable of the same type, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
System.out.println(x + y + z);
```

Reading Data From Keyboard

- Java Scanner class allows the user to take input from the console.
- It belongs to java.util package .
- It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java Program
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.
    - Syntax

```
Scanner sc=new Scanner(System.in);
```

Methods of Java Scanner Class

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

Example of Float input from user Program

```
import java.util.*;
class User
{
public static void main(String[] args)
{
Scanner sc= new Scanner(System.in);    //System.in is a standard input stream
System.out.print("Enter first number- ");
float a= sc.nextFloat();
System.out.print("Enter second number- ");
float b= sc.nextFloat();
System.out.print("Enter third number- ");
float c= sc.nextFloat();
float d=a+b+c;
System.out.println("Total= " +d);
}
}
```

Types of Variables

There are three types of variables in Java:

- **local variable**: A variable declared inside the body of the method is called local variable. user can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- **instance variable**: A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
- **static variable:** A variable which is declared as static is called static variable. Once create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example

Example to understand the types of variables in java

```
class A
{
int data=50;  //instance variable
static int m=100;  //static variable
void method()
{
int n=90;//local variable
}
}//end of class
```

Operators:

In programming, operators are the special symbol that tells the compiler to perform a special operation. Java provides different types of operators that can be classified according to the functionality they provide. There are eight types of operator

- Arithmetic Operators: + , - , / , * , %
- Assignment Operators: = , += , -= , *= , /= , %= , ^=
- Relational Operators: ==, != , < , >, <= , >=
- Unary Operators: ++ , - - , !
- Logical Operators: && , ||
- Ternary Operators: (Condition) ? (Statement1) : (Statement2);
- Bitwise Operators: & , | , ^ , ~
- Shift Operators: << , >> , >>>
- Special Operators: instanceof and dot (.) operator

instanceof : The instanceof is an object reference operator and returns true if the object of left-hand side is an instance of the class given on the right-hand . this operator allows us to determine whether the object belongs to a particular class or not
Example:
person instanceof student
is **true** if the object person belongs to the class student; otherwise it is **false**.

Dot Operator

The dot operator (.) is used to access the instance variable and methods of class objects.
Examples:
personl.age // Reference to the variable age
personl.salary() // Reference to method salary()
It is also used to access classes and sub-package from a package.

Expressions

Every expression consists of at least one operator and an operand. Operand can be a literal, variable or a method invocation i.e. an expression is a collection of variables, values, operators and method calls which evaluate to a single value.

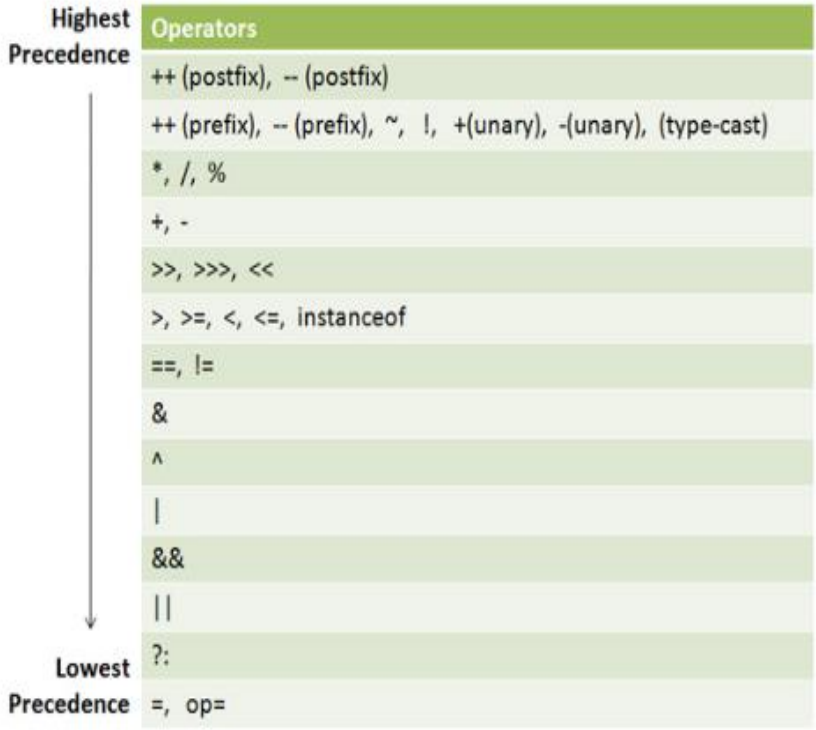
For example

```
int a = 10;  //Assignment expression
System.out.println("Value = "+x);
int result = a + 10;  //Assignment exp
if(val1 <= val2)  //Boolean expression
b = a++;  //Assignment exp
```

Operator precedence

All the operators in Java are divided into several groups and are assigned a precedence level. For e.g. 10 – 3 * 5 in this expression first multiplication is done then subtraction according to operator precedence.

The operator precedence chart for the operators in Java is shown below:



Operators
++ (postfix), -- (postfix)
++ (prefix), -- (prefix), ~, !, +(unary), -(unary), (type-cast)
*, /, %
+, -
>>, >>>, <<
>, >=, <, <=, instanceof
==, !=
&
^
|
&&
||
?:
=, op=

Associative Rules

When an expression contains operators from the same group, associative rules are applied to determine which operation should be performed first.

e.g. 10-6+2 o/p is 6 as per associative rule

The associative rules of Java are shown below:

Operator Group	Associativity	Type of Operation
! ~ ++ -- + -	right-to-left	unary
* / %	left-to-right	multiplicative
+ -	left-to-right	additive
<< >> >>>	left-to-right	bitwise
< <= > >=	left-to-right	relational
== !=	left-to-right	relational
&	left-to-right	bitwise
^	left-to-right	bitwise
|	left-to-right	bitwise
&&	left-to-right	boolean
||	left-to-right	boolean
?:	right-to-left	conditional
= += -= *= /= %= &= ^= |= <<= >>= >>>=	right-to-left	assignment
,	left-to-right	comma

Use of parenthesis in expressions:

parenthesis have the highest priority (precedence) over all the operators in Java.

E.g. (20 * 5) + (10 / 2) – (3 * 10)

There is another popular use of parenthesis. User will use them in print statements.

For example consider the following piece of code:

```
int a=10, b=20;
System.out.println("Sum of a and b is: "+a+b);
```

o/p : Sum of a and b is: 1020

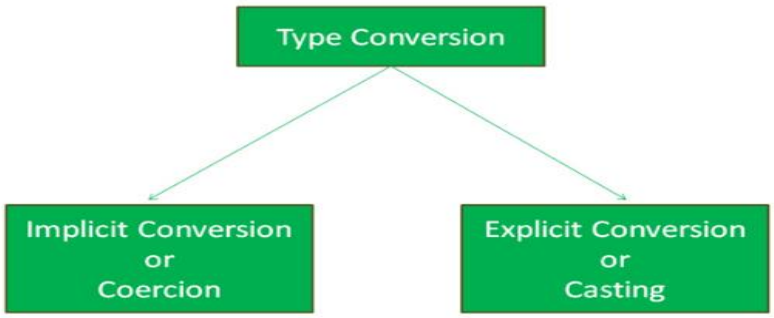
When one of the operand inside a print statement is a string, the + operator acts as a concatenation operator. To make it behave as an arithmetic operator we should enclose a+b is parenthesis as shown below:

```
int a=10, b=20;
System.out.println("Sum of a and b is: "+(a+b));
```

o/p : Sum of a and b is: 30

Type Conversions in Expressions

Converting a value from one data type to another data type is known as type conversion.



Implicit Conversion:

This type of conversion is performed automatically by Java due to performance reasons. Implicit conversion is not performed at all times. There are two rules to be satisfied for the conversion to take place. They are:

1. The source and destination types must be compatible with each other.
2. The size of the destination type must be larger than the source type.

For example, Java will automatically convert a value of *byte* into *int* type in expressions since they are both compatible and *int* is larger than *byte* type.

Since a smaller range type is converted into a larger range type this conversion is also known as **widening conversion**. Characters can never be converted to *boolean* type. Both are incompatible.

Explicit Conversion or Casting

There may be situations where user wants to convert a value having a type of size less than the destination type size. That is why this type of conversion is known as explicit conversion or casting as the programmer does this manually.

Syntax for type casting is as shown below:

(**type-name**) expression

An example for type casting is shown below

```
int a = 10;
byte b = (int) a;
```

- Since here converting a source type having larger size into a destination type having less size, such conversion is known as **narrowing conversion.**
- A type cast can have unexpected behavior. For example, if a *double* is converted into an *int*, the fraction component will be lost.

Rules for mixed data type expression

Type promotion rules of Java for expressions are listed below:

- All *char, short* and *byte* values are automatically promoted to *int* type.
- If at least one operand in an expression is a *long* type, then the entire expression will be promoted to *long*.
- If at least one operand in an expression is a *float* type, then the entire expression will be promoted to *float.*
- If at least one operand in an expression is a *double* type, then the entire expression will be promoted to *double*.

Decision Making and Branching

A Java program is a set of statements, which are normally executed sequentially in the order in which they appear. There is a number of situations, where user may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met. This involves a kind of decision making. When a program breaks the sequential flow and jumps to another part of the code is called branching .When the branching is based on a particular condition, it is known as conditional branching. If branching takes place without any decision, it is known as unconditional branching.Java language possesses such decision making capabilities and support the following statements known as control or decision making statements.

1. if statement
2. switch statement
3. Conditional operator statement

Simple IF Statement

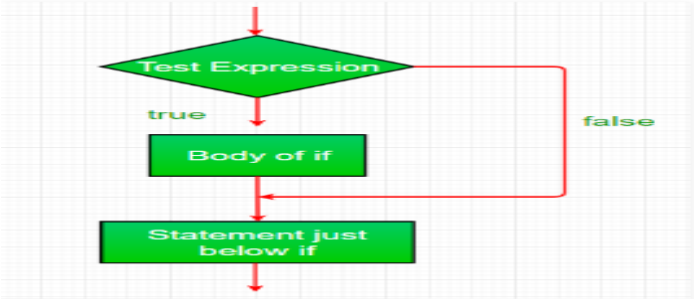
if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax is as follows

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

Here first condition is checked if condition is true then block of statements executed otherwise control goes to next statement just below if.

Flowchart is as follows



Example of simple if statement

```
// Java program to illustrate If statement
class IfDemo
{
   public static void main(String args[])
   {
      int i = 10;
       if (i > 15)
         System.out.println("10  is less than 15");
       // This statement will be executed
      // as if considers one statement by default
      System.out.println("I am Not  in  if");
   }
}
```

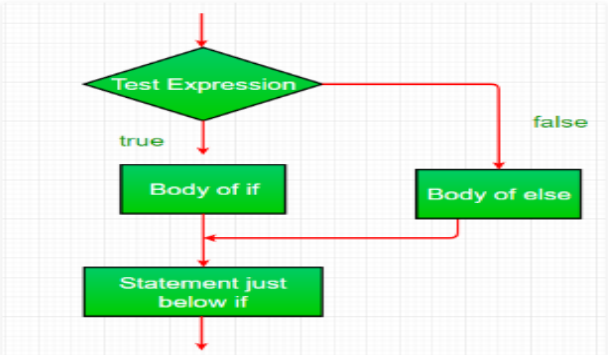
if…else statement

The if…else statement is an extension of simple if statement. The general form is Flowchart

```
if (condition)
{
     // Executes this block if
     // condition is true
}
else
{
     // Executes this block if
     // condition is false
}
```

Here first condition is checked if condition is true then if block of statements executed and if condition is false then else block of statement is executed.

Flowchart is



Example of if…else statement

```
// Java program to illustrate if-else statement
class IfElseDemo
{
   public static void main(String args[])
   {
      int i = 10;

      if (i < 15)
         System.out.println("i is smaller than 15");
      else
         System.out.println("i is greater than 15");
   }
}
```

Nesting of if...else statement

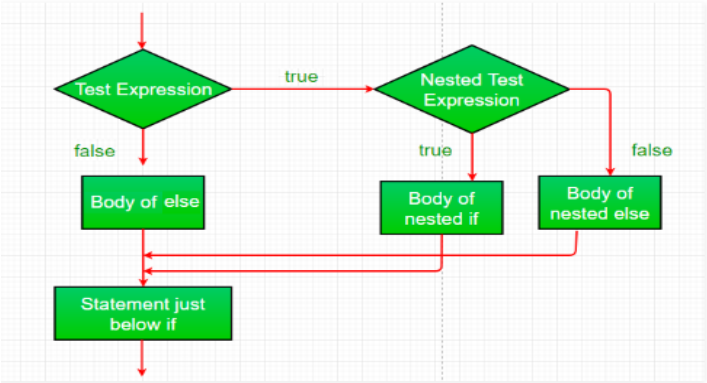
When series of decisions are involved then use more than one if..else statement in nested form

Syntax :

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

Here first condition1 is checked. If condition 1 is true then again condition 2 is executed. In this way serious of decision is checked through nesting if...else statement.

Flowchart



Example:

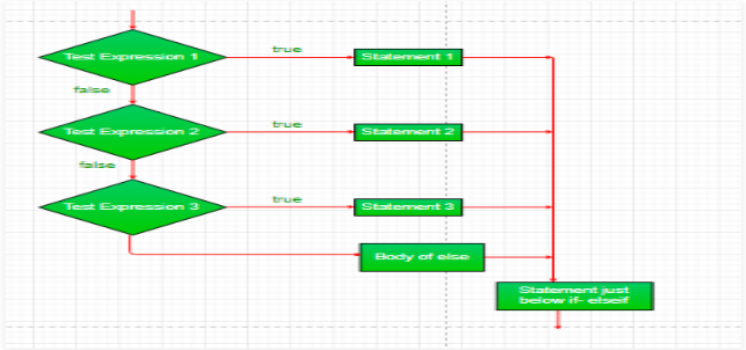
```
// Java program to illustrate nested-if statement
class NestedIfDemo
{
   public static void main(String args[])
   {
      int i = 10;
        if (i == 10)
      {
         // First if statement
         if (i < 15)
            System.out.println("i is smaller than 15");
          // Nested - if statement
         // Will only be executed if statement above
         // it is true
         if (i < 12)
            System.out.println("i is smaller than 12 too");
         else
            System.out.println("i is greater than 15");
      }
   }
}
```

else–if ladder

There is another way of putting if together when multipath decisions are involved. multipath decision is a chain of if in which the statement associated with each else if. It takes following syntax

```
if (condition)
      statement;
else if (condition)
      statement;
.
.
else
      statement;
```

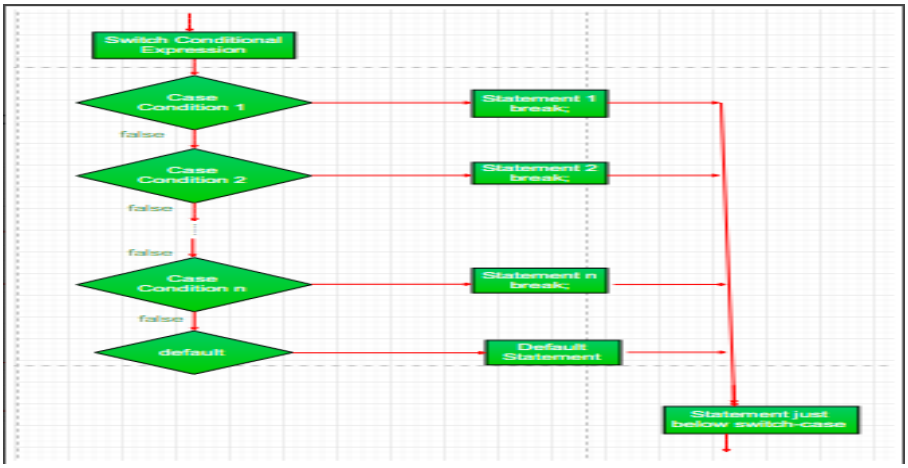
Flowchart is as follows:



Switch Statement:

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
Syntax and Flowchart:

```
switch (expression)
{
   case value1:
      statement1;
      break;
   case value2:
      statement2;
      break;
   .
   .
   case valueN:
      statementN;
      break;
   default:
      statementDefault;
}
```



Example:

```
public class Grades
 {
   public static void main(String args[]) {
     Scanner sc = new Scanner(System.in);
     System.out.println("Enter average of your marks (less than 100)::");
     int average = sc.nextInt();
     char grade;
     if(average>=80){
       grade = 'A';
     }else if(average>=60 && average<80){
       grade = 'B';
     }
     else if(average>=40 && average<60){
       grade = 'C';
     }
     else {
       grade = 'D';
     }
switch(grade) {
        case 'A' :
          System.out.println("Excellent!");
          break;
        case 'B' :
        case 'C' :
          System.out.println("Well done");
          break;
        case 'D' :
          System.out.println("You passed");
        case 'F' :
          System.out.println("Better try again");
          break;
        default :
        System.out.println("Invalid grade");
     }
     System.out.println("Your grade is " + grade);
   }
}
```

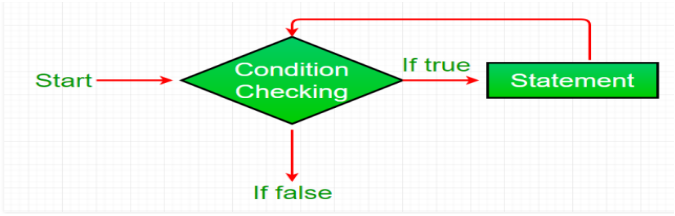
While loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

The while is entry controlled loop statement.

Syntax :

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



Example

```
// Java program to illustrate  use of while loop
class whileDemo
{
   public static void main(String args[])
   {
      int  x = 1;
        // Exit when x becomes greater than 4
      while (x <= 4)
      {
         System.out.println("Value of x:" + x);

         // Increment the value of x for
         // next iteration
         x++;
      }
   }
}
```

```
//Use of command line argument
```

```
class Ctest
{
public static void main(String ar[])
 {
   int count, i=0;
   String st;
   count = ar.length;
   System.out.println("no. of arguments="+count);
   while(i<count)
    {
      st=ar[i];
      i=i+1;
      System.out.println(i+":"+ "java is "+st);
}
}
}
```

Output

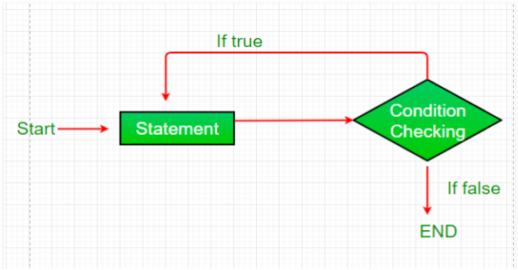
```
C:\Users\GT Asus\Desktop\java pract>java Ctest
no. of arguments=0

C:\Users\GT Asus\Desktop\java pract>java Ctest Simple OOP Distrubuted robust sec
ure
no. of arguments=5
1:java is Simple
2:java is OOP
3:java is Distrubuted
4:java is robust
5:java is secure
```

do while

do while loop is similar to while loop with only difference that it checks the condition after executing the statements, and therefore is known as **Exit Control Loop.**

Syntax and Flowchart is as follows

```
{
     statements..
}
while (condition);
```



```
// multiplication table for demonstrating Nested Do ..While  Loop
class DoWhile
{
public static void main(String ar[])
 {
    int r,c,y;
   System.out.println("\n Multipliction Table\n");
    r=1;
do
{
c=1;
do
{
 y=r*c;
    System.out.print(" "+y);
    c+=1;
}
    while(c<=5);
   System.out.println("\n");
   r+=1;
}
while(r<=5);
}
}
```

Output

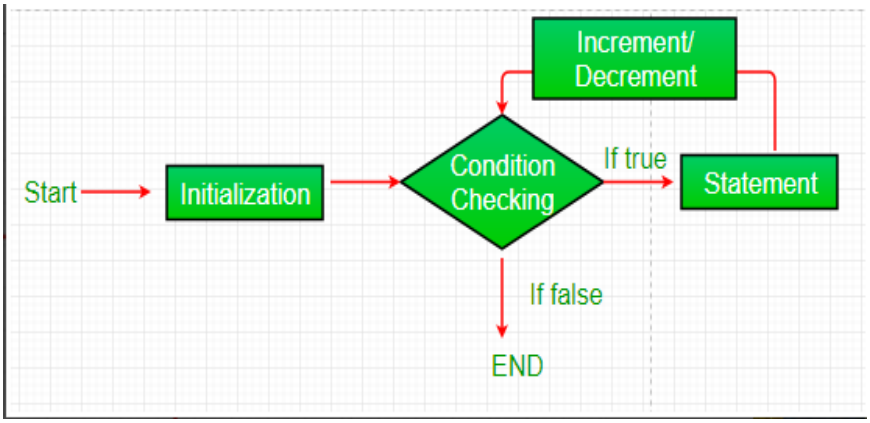
```
C:\Users\GT Asus\Desktop\java pract>javac DoWhile.java
C:\Users\GT Asus\Desktop\java pract>java DoWhile
 Multipliction Table
 1 2 3 4 5
 2 4 6 8 10
 3 6 9 12 15
 4 8 12 16 20
 5 10 15 20 25
```

For Loop

for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

For loop is another entry controlled loop. In this statement the initialization, condition and increment/decrement in one line. This loop statement is providing a shorter, easy to debug structure of looping.

```
for (initialization condition; testing condition;
                                increment/decrement)
{
    statement(s)
}
```



Additional Features of for Loop

The for loop in Java has several capabilities that are:

1.More than one variable can be initialized at a time in the for statement.

Ex:– for(p=1, n=0; n<10;++n)

2. The increment section may also have more than one part.

Ex:–for(n=1, m=50; n<=m;n=n+1, m =m–1)

The multiple arguments in initialization section & in the increment section are separated by commas.

3. The test–condition may have any compound relation & the testing need not be limited only to the loop control variable.

Ex:–for(i=1; i<10&&sum<100;i++)

4. It is also possible to use expressions in the assignment statements of initialization & increment sections.

Ex:– for(x=(m+n)/2; x>0;x=x/2)

5. Another unique aspect of for loop is that one or more sections can be omitted, if necessary.

Ex:–m=5;

For(;m!=100;)

{

printf(“%d\n”,m);

m=m+5;

}

6. We can set uptime delay loops using the null statement as follows.

Ex:– for (i=1000;j>0;j=j–1);

This loop is executed 1000 times without producing any output. It simply causes a time delay.

Nested Loops: Nested loop means a loop statement inside another loop statement. Therefore nested loops are also called as “loop inside loop“.

Syntax for Nested For loop:

```
for ( initialization; condition; increment ) {

   for ( initialization; condition; increment ) {

      // statement of inside loop
   }

   // statement of outer loop
}
```

```
//Program For Pattern
import java.util.*;
public class Pattern5
{
public static void main(String[] args)
{
int i, j, rows;
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of rows you want to print: ");
 rows = sc.nextInt();
for (i = 1; i <= rows; i++)
{
for (j = 1; j <= i; j++)
{
System.out.print(i+" ");
}
System.out.println();
}
}
}
```

break and continue statements

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 the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement

The syntax of a break is a single statement inside any loop −

```
break;
```

Example of Break Statement

```
public class Test1
 {
  public static void main(String args[])
{
     int [] numbers = {10, 20, 30, 40, 50};
     for(int x : numbers )
{
        if( x == 30 )
          break;
         System.out.print( x );
        System.out.print("\n");
     }
  }
}
```

continue statement

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 a 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.
- The syntax of a continue is a single statement inside any loop −

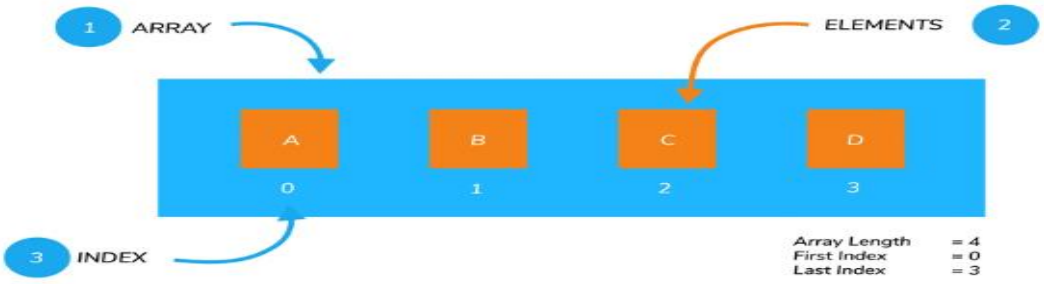
```
continue;
```

continue example

```
public class Test2
 {
   public static void main(String args[])
{
     int [] numbers = {10, 20, 30, 40, 50};
     for(int  x : numbers )
 {
        if( x == 30 ) {
          continue;
        }
        System.out.print( x );
        System.out.print("\n");
     }
   }
}
```

Arrays in Java:

A variable is a location in our program with a name and value. This value could be any data type, like int. An array is another variable type or a container object with a fixed number of values that are all of a single type. In other words, array is a collection of similar data types. When an array is created, the size of the array i.e length is also fixed.



Features of Array

- Arrays are dynamically allocated
- Arrays are objects in Java
- A Java array variable is declared like other variables
- The variables are ordered, with the index beginning at 0
- The superclass of the array type is Object
- The size of an array is specified with an int value

One-Dimensional Arrays :

The general form of a one-dimensional array declaration is

```
type var-name[];
 OR
type[] var-name;
```

for e.g. list of integers.

```
int[] myArray
```

Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc. or user-defined data types.

When an array is declared, only a reference of array is created. To actually create or give memory to array by using **new** and assign it to myArray. The general form of *new* as it applies to one-dimensional arrays appears as follows:

```
var-name = new type [size];
```

For e.g.

```
myArray[]; //declaring array
 myArray = new int[20]; // allocating memory to array
```

Or

```
int[] myArray = new int[20]; // combining both statements in one
```

Array Literal

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

- int[] myArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 }; // Declaring array literal
- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1.

```
// accessing the elements of the specified array
for (int i = 0; i < arr.length; i++)
System.out.println("Element at index " + i + " : "+ arr[i]);
```

Multidimensional Arrays

- Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array. These are also known as Jagged_Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension. Examples:

- int[][] intArray = new int[10][20]; //a 2D array or

- matrix int[][][] intArray = new int[10][20][10]; //a 3D array

Example of Multidimensional Array

```
class Multi
{
   public static void main(String args[])
   {
      // declaring and initializing 2D array
      int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };

      // 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();
      }
   }
}
```