JAVA SYLLABUS

UNIT-I

Introduction to Java:

Java is one of the most popular programming languages out there. Released in 1995 and still widely used today, Java has many applications, including software development, mobile applications, and large systems development. Knowing Java opens a lot of possibilities for you as a developer.

Why to Learn java Programming?

OR

Features of Java Programming?

Java is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Software Development Domain. I will list down some of the key advantages of learning Java Programming:

- **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** − unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Explain OOP's Features in JAVA?

Note:

Refer PPT

Difference b/w C & C++:-

Sr. No.	Key	C	C++
1	Introduction	C was developed by Dennis Ritchie in around 1969 at AT&T Bell Labs.	C++ was developed by Bjarne Stroustrup in 1979.
2	Language Type	As mentioned before C is procedural programming.	On the other hand, C++ supports both procedural and object-oriented programming paradigms.
3	OOPs feature Support	As C does not support the OOPs concept so it has no support for polymorphism, encapsulation, and inheritance.	C++ has support for polymorphism, encapsulation, and inheritance as it is being an object-oriented programming language
4	Data Security	As C does not support encapsulation so data behave as a free entity and can be manipulated by outside code.	On another hand in the case of C++ encapsulation hides the data to ensure that data structures and operators are used as intended.
5	Driven type	C in general known as function-driven language.	On the other hand, C++ is known as object driven language.
6	Feature supported	C does not support function and operator overloading also do not have namespace feature and reference variable functionality.	On the other hand, C++ supports both function and operator overloading also have namespace feature and reference variable functionality.

Difference b/w C++ and JAVA

#1) Platform Independence

C++	Java
C++ is a platform dependent language. The source code written in C++ needs to be compiled on every platform.	Java is platform-independent. Once compiled into byte code, it can be executed on any platform.

#2) Compiler and Interpreter

C++	Java
C++ is a compiled language. The source program written in C++ is compiled into an object code which can then be executed to produce an output.	Java is a compiled as well as an interpreted language. The compiled output of a Java source code is a byte code which is platform-independent.

#3) Portability

C++	Java
C++ code is not portable. It must be compiled for each platform.	Java, however, translates the code into byte code. This byte code is portable and can be executed on any platform.

#4) Memory Management

C++	Java
Memory management in C++ is manual. We need to allocate/deallocate memory manually using the new/delete operators.	In Java the memory management is system-controlled.

#5) Multiple Inheritance

C++	Java
C++ supports various types of inheritances including single and multiple inheritances. Although there are problems arising from multiple inheritances, C++ uses the virtual keyword to resolve the problems.	Java, supports only single inheritance. Effects of multiple inheritance can be achieved using the interfaces in Java.

#6) Overloading

C++	Java
In C++, methods and operators can be overloaded. This is static polymorphism.	In Java, only method overloading is allowed. It does not allow operator overloading.

#7) Virtual Keyword

C++	Java
As a part of dynamic polymorphism, in C++, the virtual keyword is used with a function to indicate the function that can be overridden in the derived class. This way we can achieve polymorphism.	In Java, the virtual keyword is absent. However, in Java, all non-static methods by default can be overridden. Or in simple terms, all non-static methods in Java are virtual by default.

#8) Pointers

C++	Java
C++ is all about pointers. As seen in tutorials earlier, C++ has strong support for pointers and we can do a lot of useful programming using pointers.	Java has limited support for pointers. Initially, Java was completely without pointers but later versions started providing limited support for pointers. We cannot use pointers in Java as leisurely as we can use in C++.

#9) Documentation Comment

C++	Java
C++ has no support for documentation comments.	Java has a built-in support for documentation comments (/**…*/). This way Java source files can have their own documentation.

#10) Thread Support

C++	Java
C++ doesn't have in-built thread support. It mostly relies on third-party threading libraries.	Java is in-built thread support with a class "thread". We can inherit the thread class and then override the run method.

Hardware & Software Requirements:

Hardware Requirements:

Operating System: open source: windows, UNIX, Linux, Ubuntu, etc..

Software Requirements:

https://www.java.com/en/download/

JDK & JRE Installation

IDE: Eclipse

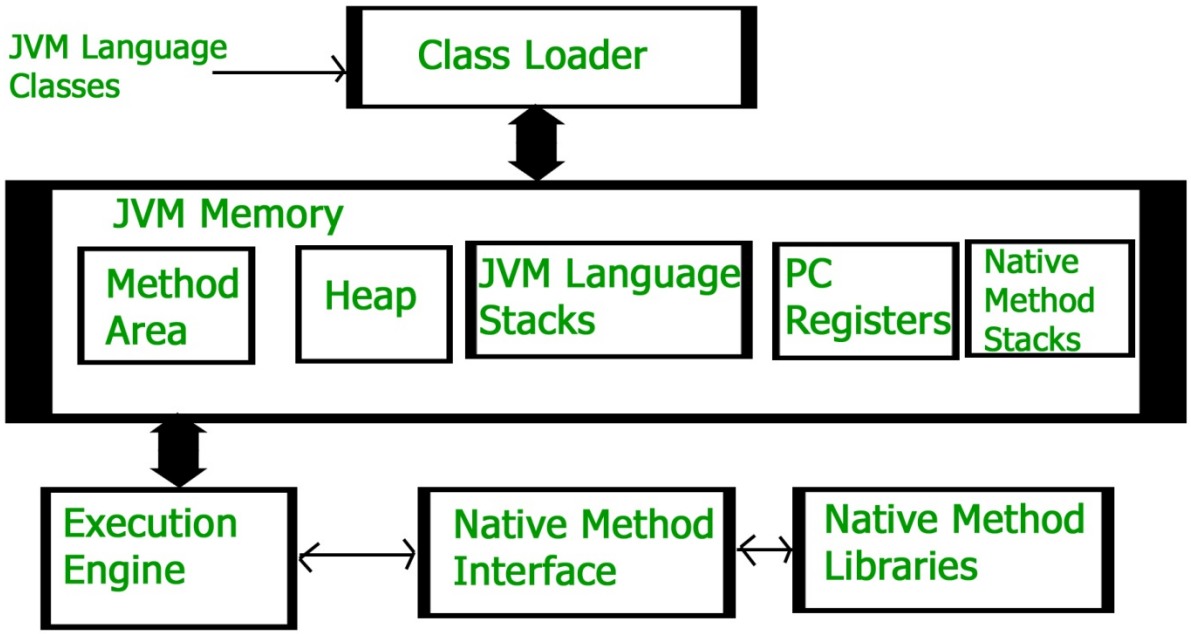
https://www.eclipse.org/

Sample Java program:

```
  Public class Sample
{
  Public static void main(String[] args)
{
 System.out.println("welcome to java");
}
}
```

Explain JVM Architecture?

Ans:



1) Class loader

Class loader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the class loader. There are three built-in class loaders in Java.

2) Class (Method) Area

Class (Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1. **A virtual processor**
2. **Interpreter:** Read byte code stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, and Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Explain Command Line Arguments:-

The java command-line argument is an argument i.e. passed at the time of running the java program.

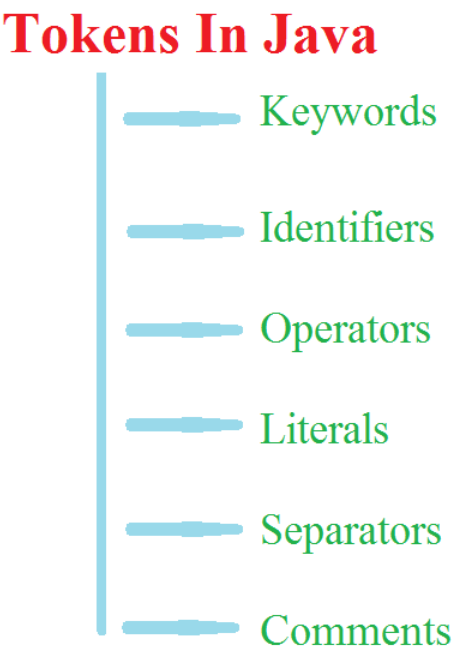
The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

Example Program:

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

JAVA TOKENS:-



1. Keyword:

Keywords are pre-defined or reserved words in a programming language.
Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. Java **language supports following keywords:**

```
abstract    assert     boolean
break       byte       case
catch       char       class
const       continue   default
do          double     else
enum        exports    extends
final       finally    float
for         goto       if
implements  import     instanceof
int         interface  long
module      native     new
open        opens      package
private     protected  provides
public      requires   return
short       static     strictfp
super       switch     synchronized
```

this throw throws
to transient transitive
try uses void
volatile while with

2. **<u>Identifiers:-</u>**
   A. **Identifiers are used to identify the given name for a class/object/method/variable.**
   B. **Identifiers** are the names of variables, methods, classes, packages and interfaces. Unlike literals they are not the things themselves, just ways of referring to them.
   C. **Identifiers are two types they are:**
      i. **<u>Valid Identifiers:</u>**
         - Each **identifier** must have at least one character.
         - The first character must be picked from: alpha, underscore, or dollar sign.
         - The first character cannot be a digit.
         - The rest of the characters (besides the first) can be from: alpha, digit, underscore, or dollar sign.

      ii. **Invalid Identifiers**

3. **Operators:**

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

S.NO	OPERATOR	MEANING	EXAMPLE
1	+	PLUS / ADDITION	C=A+B
2	-	SUBTRACTS	C=A-B

3	*	MULTIPLICATION	C=A*B
4	/	DIVISION	C=A/B
5	%	MODULO DIVISON	C=A%B

Example:

```
public class Test
{

  public static void main(String args[])
{
    int a = 10;
    int b = 20;

    System.out.println("a + b = " + (a + b) );
    System.out.println("a - b = " + (a - b) );
    System.out.println("a * b = " + (a * b) );
    System.out.println("b / a = " + (b / a) );
    System.out.println("b % a = " + (b % a) );
    System.out.println("c % a = " + (c % a) );

  }
}
```

Relational Operators

Relational operators are the operators perform the operations on the given operands.

S.NO	OPERATOR	MEANING	EXAMPLE
1	<	LESSTHAN	A<B
2	<=	LESSTHAN EQUALS	A<=B
3	>	GREATERTHAN	A>B
4	>=	GREATERTHAN EQUALS	A>=B
5	==	EQUALS	A==B
6	!=	NOT EQUALS	A!=B

EXAMPLE:

```
package app;
```

```java
import java.util.Scanner;

public class RelationalOperator
{

	public static void main(String[] args)
	{
		//relational Operator
		// <, <=, >,>=, ==, !=;

		int x,y;
		Scanner obj=new Scanner(System.in);
		System.out.println("Enter x, y values");
		x=obj.nextInt();
		y=obj.nextInt();
		if(x==y)
		{
			System.out.println("Both are equal");
		}
		else
		{
			System.out.println("both are not equal");
		}
	}

}
```

Logical Operators:

The logical operators || (conditional-OR) and && (conditional-AND) operate on Boolean expressions.

S.NO	OPERATOR	MEANING	EXAMPLE
1	&&	LOGICAL AND	C= A && B
2	||	LOGICAL OR	C=A||B
3	`	LOGICAL NOT	A=`A

LOGICAL AND:TRUTH TABLE

A	B	C=A*B
1	1	1
0	0	0
1	0	0

0 1 0

LOGICAL OR: TRUTH TABLE

A	B	C=A+B
0	0	0
1	0	1
0	1	1
1	1	1

LOGICAL NOT: TRUTH TABLE

A	`A
0	1
1	0

EXAMPLE:

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

      int a = 1, b = 2, c = 9;
      boolean result;

      // At least one expression needs to be true for the result to be true
      result = (a > b) || (c> a);

      // result will be true because (number1 > number2) is true
      System.out.println(result);

      // All expression must be true from result to be true
      result = (a > b) && (c> a);

      // result will be false because (number3 > number1) is false
      System.out.println(result);
   }
```

```
}
```

Ternary Operator:

Ternary operator is also known as conditional operator.

Syntax:

Expression= Exp1? Exp2:Exp3;

If Exp1 is true then Exp2 will execute.

If Exp1 is false then Exp3 will execute.

Example: Leap year OR not?

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

      int februaryDays = 29;
      String result;

       result =  (februaryDays == 28) ? "Not a leap year" : "Leap year";
       System.out.println(result);
   }
}
```

Bitwise Operator:

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift

>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR

Example:

```
public class Test {

  public static void main(String args[]) {
    int a = 60;         /* 60 = 0011 1100 */
    int b = 13;         /* 13 = 0000 1101 */
    int c = 0;

    c = a & b;       /* 12 = 0000 1100 */
    System.out.println("a & b = " + c );

    c = a | b;       /* 61 = 0011 1101 */
    System.out.println("a | b = " + c );

    c = a ^ b;       /* 49 = 0011 0001 */
    System.out.println("a ^ b = " + c );

    c = ~a;          /*-61 = 1100 0011 */
    System.out.println("~a = " + c );

    c = a << 2;      /* 240 = 1111 0000 */
    System.out.println("a << 2 = " + c );

    c = a >> 2;      /* 15 = 1111 */
    System.out.println("a >> 2  = " + c );

    c = a >>> 2;     /* 15 = 0000 1111 */
    System.out.println("a >>> 2 = " + c );
```

```
  }
}
```

Unary Operator:

Unary Operators are used to perform operations on the single operand.

They are:

Unary plus (+) pre increment, post increment

Unary minus (-) pre decrement, post decrement

```
package app;

public class UnaryOperator {

	public static void main(String[] args)
	{
		int i=10;
		//System.out.println(i);

		//unary operator
		//pre increment (++i) it incerement by 1

		 // 1st increment by 1 then assign to value1+(10) =11
		//post increment(i++)
		//pre decrement(--i)
		//post decrement(i--)

		//System.out.println(++i);
		System.out.println(i++);
		System.out.println(i);
		System.out.println(--i);
		System.out.println(i--);
		System.out.println(i);
	}

}
```

Assignment Operator:

Assignment Operator is used to assign a value to the declared variable.

Assignment operator is denoted by single equal (=).

Example

Int a;

Int a=60; // assign a value to the declared variable.

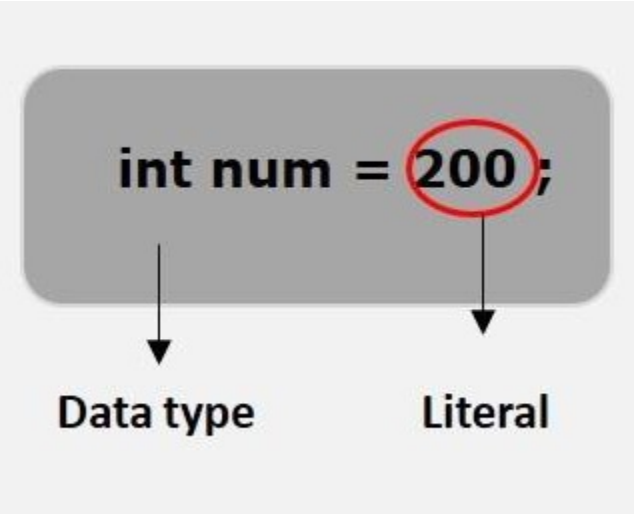
Literals:

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable.

Example

```
byte a = 68;

char a = 'A'
```



Separators:

Separators help define the structure of a program.

The separators used in HelloWorld are parentheses, ( ), braces, { }, the period, ., and the semicolon, ;.

Separator	Name	Use
.	Period	It is used to separate the package name from sub-package name & class name. It is also used to separate variable or method from its object or instance.
,	Comma	It is used to separate the consecutive parameters in the method definition. It is also used to separate the consecutive variables of same type while declaration.
;	Semicolon	It is used to terminate the statement in Java.
()	Parenthesis	This holds the list of parameters in method definition. Also used in control statements & type casting.
{}	Braces	This is used to define the block/scope of code, class, methods.
[]	Brackets	It is used in array declaration.
Separators in Java		

Comments:

In **Java** there are three **types of comments**:

- Single – line **comments**.
- Multi – line **comments**.
- Documentation **comments**.

- **Single – line comments:**

  A beginner level programmer uses mostly single-line comments for describing the code functionality. Its the most easiest typed comments.

Syntax:

```
//Comments here( Text in this line only is considered as comment )
```

Example:

```
//Java program to show single line comments

        class Scomment
        {
                public static void main(String args[])
                {
                        // Single line comment here
                        System.out.println("Single line comment above");
                }
        }
```

Multi-line Comments:

To describe a full method in a code or a complex snippet single line comments can be tedious to write, since we have to give '//' at every line. So to overcome this multi line comments can be used.

Syntax:

```
/*Comment starts
continues
continues
.
.
.
Commnent ends*/
```

Example:

```
//Java program to show multi line comments

class Scomment

{

        public static void main(String args[])

        {
```

```
            System.out.println("Multi line comments below");
            /*Comment line 1
            Comment line 2
            Comment line 3*/
        }
}
```

Documentation Comments

This type of comments are used generally when writing code for a project/software package, since it helps to generate a documentation page for reference, which can be used for getting information about methods present, its parameters, etc.

Syntax:

```
/**Comment start
*
*tags are used in order to specify a parameter
*or method or heading
*HTML tags can also be used
*such as <h1>
*
*comment ends*/
```

Data Types:

Data types specify the different sizes and values that can be stored in the variable.

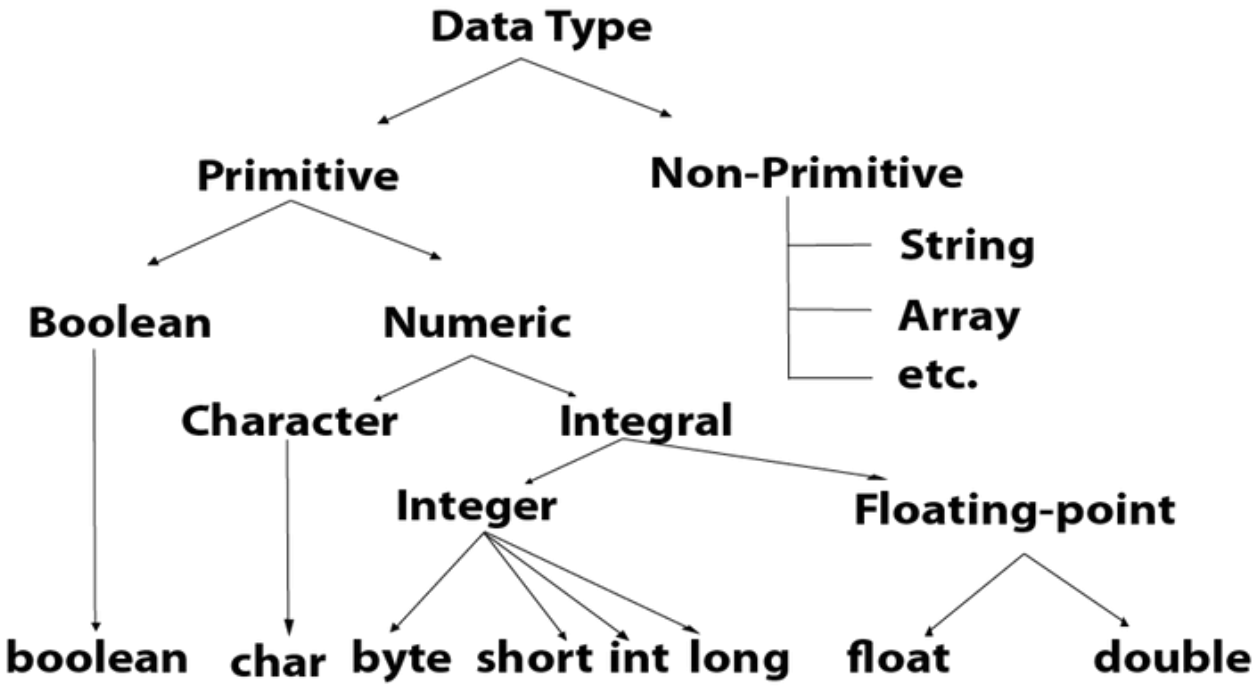
There are two types of data types in Java:

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.



Data Type	Default Value	Default size
boolean	False	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

Declaration of Variables:

Declaration of variable is nothing but defined a variable with respective data type.

Syntax:

<data type> variable name;

Example:

int i;

float f;

double d;

Initialization of Variables:

Initialization of Variables is nothing but assigning a value to the declared variable.

Syntax:

<data type> variable name=value;

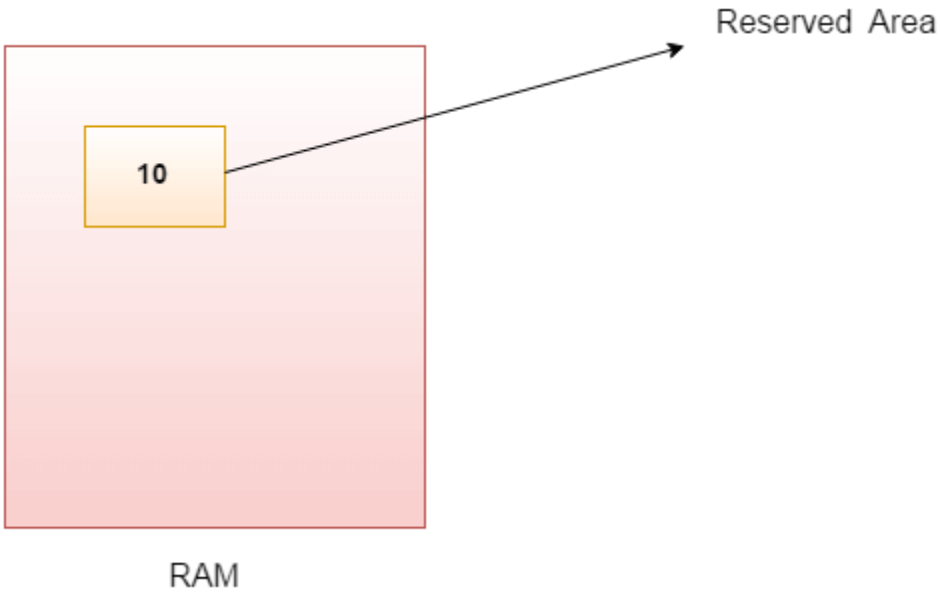
int i=90;

float f=20f;

char c='a';

Variable:

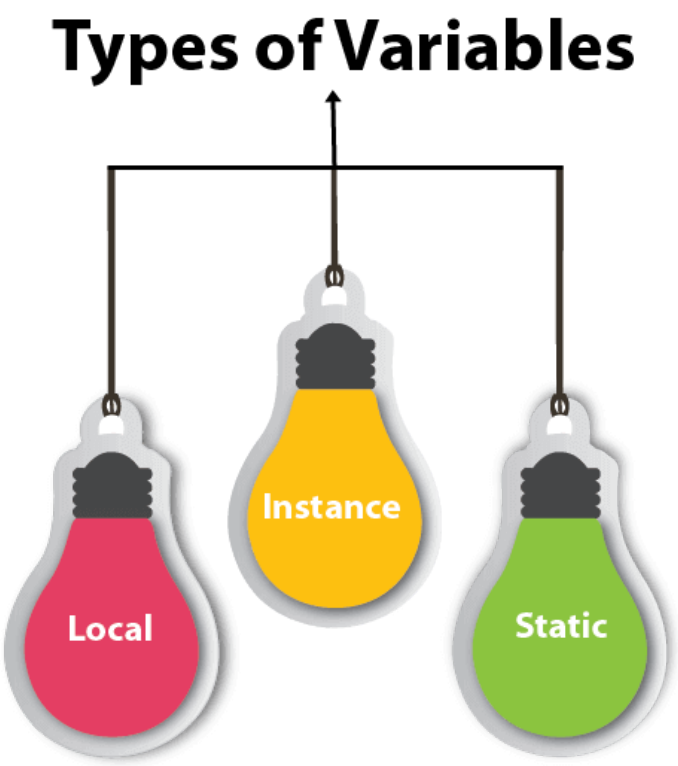
Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.



Example:

```
int data=50;//Here data is variable
```

Scope of Variables:



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.

Note:

A local variable cannot be defined with "static" keyword.

Example:

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

   //local variable declaration
```

```
        int i=60; //local variable
        System.out.println(i);
    }
}
```

Example 2:

```
package app;

public class Local
{
    void test1()
    {
        int m=60;
        System.out.println(m);
    }

    public static void main(String[] args)
    {
        Local l=new Local();
        l.test1();
        int i=90;
        float f=2f;
        boolean be=false;
        System.out.println(i);
        System.out.println(f);
        System.out.println(be);


    }

}
```

2) 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.

It is called instance variable because its value is instance specific and is not shared among instances.

Note:

1. Instance variables should access by using object invocation.
2. All non static global variables will have default values based on their data types

```
package app;

public class Manager5
{
  //global variable // instance variable // non-static global variables

        byte b;
        short s;
        int i;
        long l;
        float f;
        double d;
        char c;
        boolean be;

        public static void main(String[] args)
        {
                Manager5 m1=new Manager5();
          System.out.println("the default value of byte is"+m1.b);
          System.out.println("the default value of short is "+m1.s);
          System.out.println("default value of int is"+m1.i);
          System.out.println("default value of long is"+m1.l);
          System.out.println("default value of float is"+m1.f);
          System.out.println("default value of double is"+m1.d);
          System.out.println("default value of char is"+m1.c);
          System.out.println("default value of boolean"+m1.be);
        }


}
```

Default values:

1. Byte=0
2. Short=0
3. Int =0
4. Long =0
5. Flaot=0.0

6. Double=0.0
7. Char=null
8. Boolean=false

3) Static variable

A variable which is declared as static is called static variable. It cannot be local.

You can 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.

Note:

1. All static global variables will have default values with respect to their data types.

Default values:

1. Byte=0
2. Short=0
3. Int =0
4. Long =0
5. Flaot=0.0
6. Double=0.0
7. Char=null
8. Boolean=false

Example:

```
package app;

public class Manager5
{
  //global variable

        static byte b;
        static short s;
        static int i;
        static long l;
        static float f;
        static double d;
        static char c;
        static boolean be;

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

```
        System.out.println("the default value of byte is"+b);
        System.out.println("the default value of short is "+s);
        System.out.println("default value of int is"+i);
        System.out.println("default value of long is"+l);
        System.out.println("default value of float is"+f);
        System.out.println("default value of double is"+d);
        System.out.println("default value of char is"+c);
        System.out.println("default value of boolean"+be);
    }


}
```

Note:

1. **When variables are static object creation is optional.**

Constants:

A constant is a variable whose value cannot change once it has been assigned.

Java doesn't have built-in support for constants, but the variable modifiers *static* and *final* can be used to effectively create one.

```
package app;

public class const1 {

    public static void main(String[] args)
    {
        final float  pi=3.14f;
        System.out.println(pi);
    }

}
```

Type Casting/ conversion:

Type conversion is a process of converting from one data type to another data type.

We have two types of Type Casting/ Conversion:

They are:

1. **Auto up casting**
2. **Explicit down casting**

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

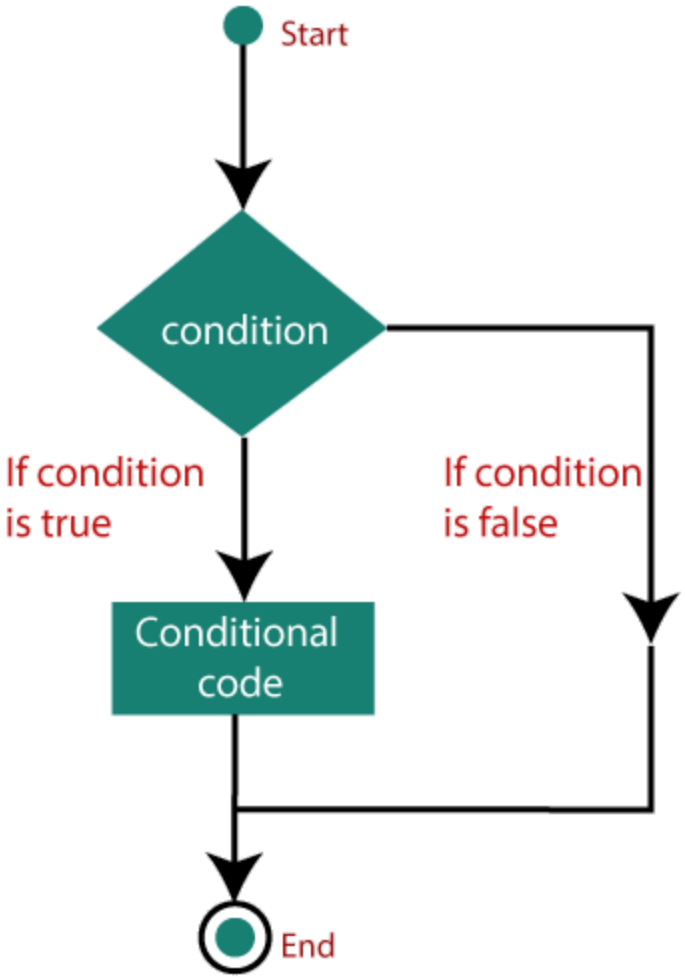
Example:

```
package app;

public class Typeconversion {

	public static void main(String[] args)
	{
		//type conversion
		/*
		 * it is a process of converting from one data type to
		 * another data type
		 * we have two types of type conversions they are:
		 * 1. Implicit (auto up casting)
		 * 2. Explicit (down casting)
		 *
		 *
		 * primitive data types:
		 * 1. byte ---->short--> int--->long---->float--->double
		 * 2.short
		 * 3.int
		 * 4.long
		 * 5.flaot
		 * 6.double
		 * 7.char
		 * 8.boolean
		 *
	  Explicit Down casting:

		 * double---->float(X)
		 * float---->long(x)
		 * long------>int(X)
		 * int-------->short(x)
		 *
		 */
```

```
            int i=10;
            int j=i;
            System.out.println(j);
        float f=j;          Auto up casting
        System.out.println(f);
        int k=(int) f;// explicit down casting.


        double d=90d;
        float f1=(float) d; //explicit type conversion


    }

}
```

Q . Explain different types of decision making statements in Java?

Ans:



Types of Conditional Statements

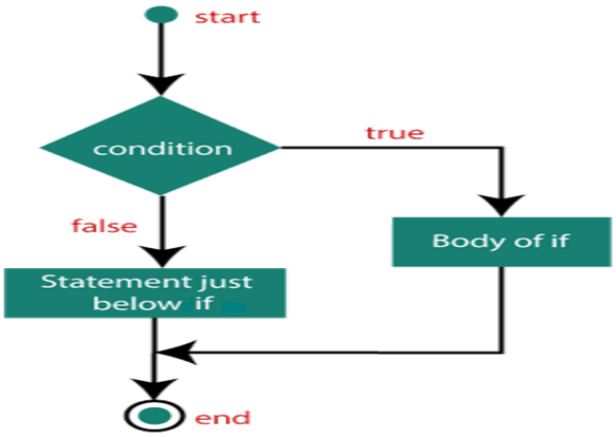
- **if statement**
- **if….else statement**
- **if….else if….statement**
- **nested if statement**

The if statement

It is one of the simplest decision-making statement which is used to decide whether a block of JavaScript code will execute if a certain condition is true.

Syntax

```
if (condition)
{
  // block of code will execute if the condition is true
}
```



Simple if:

Program 1:

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

  int age=18;
  if(age>=18)
   {
  System.out.println("Eligible to vote");
  }
 }
}
```

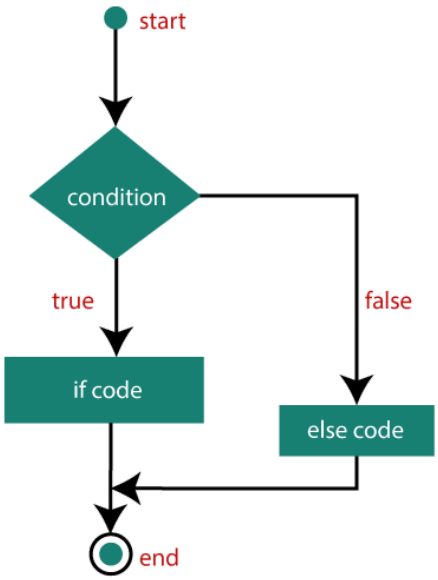
If-else:

1. An **if….else statement** includes two blocks that are **if block** and **else block.**
2. **if, else are keywords in java.**
3. **If the given condition is true then if-block will be executed.**
4. **If the given condition is false then else-block will be executed.**

Syntax:

```
if (condition)
 {
  // block of code will execute if the condition is true
}
 else
```

```
{
 // block of code will execute if the condition is false
}
```




```
package app;

public class Decision1 {

	public static void main(String[] args)
	{
		int age=17;
		if(age>=18)
		{
			System.out.println("eligible to vote");
		}
		else
		{
			System.out.println("not eligible to vote");
		}


	}

}
```

The if....else if.....else statement

It is used to test multiple conditions.

The **if statement** can have multiple or zero **else if statements** and they must be used before using the **else statement.** You should always be kept in mind that the **else statement** must come after the **else if statements.**

Syntax

```
if (condition1)
{
  //  block of code will execute if condition1 is true
}
 else if (condition2)
{
  //  block of code will execute if the condition1 is false and condition2 is true
}
else
{
  //  block of code will execute if the condition1 is false and condition2 is false
}
```

Example:

Write a java program to find biggest of 3 no's using if-else-if condition.

```
/** write a java program to find biggest of 3 no's using if else if condition*/

package app;


public class Biggest
{

        public static void main(String[] args)
        {
                int num1 = 10, num2 = 20, num3 = 7;

            if( num1 >= num2 && num1 >= num3)
               System.out.println(num1+" is the largest Number");

            else if (num2 >= num1 && num2 >= num3)
               System.out.println(num2+" is the largest Number");

            else
               System.out.println(num3+" is the largest Number");


                        }
```

```
}
```

The nested if statement or else-if ladder:

Syntax:

```
if (condition1)
{
Statement 1; //It will execute when condition1 is true
if (condition2)
 {
Statement 2; //It will execute when condition2 is true
}
else
{
 Statement 3; //It will execute when condition2 is false
}
}
```

```
package app;

import java.util.Scanner;

public class StudentGrade {

	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		String name;
		System.out.println("Enter studnet name:");
		name=sc.next();
		int java,CA,SE;
		System.out.println("Enter java,ca,se marks:");
		java=sc.nextInt();
		CA=sc.nextInt();
		SE=sc.nextInt();
		int total=java+CA+SE;
		System.out.println("the total is:"+total);
		float avg=total/3;
		System.out.println("the average is:"+avg);
		if(avg>=90&&avg<=100)
		{
```

```
                System.out.println("Distinction");
            }
            else if(avg>=75&&avg<=90)
            {
                System.out.println("First class");
            }
            else if(avg>=60&&avg<=75)
            {
                System.out.println("second class");
            }
            else
            {
                System.out.println("fail class");
            }

        }

    }
```

Example 2:

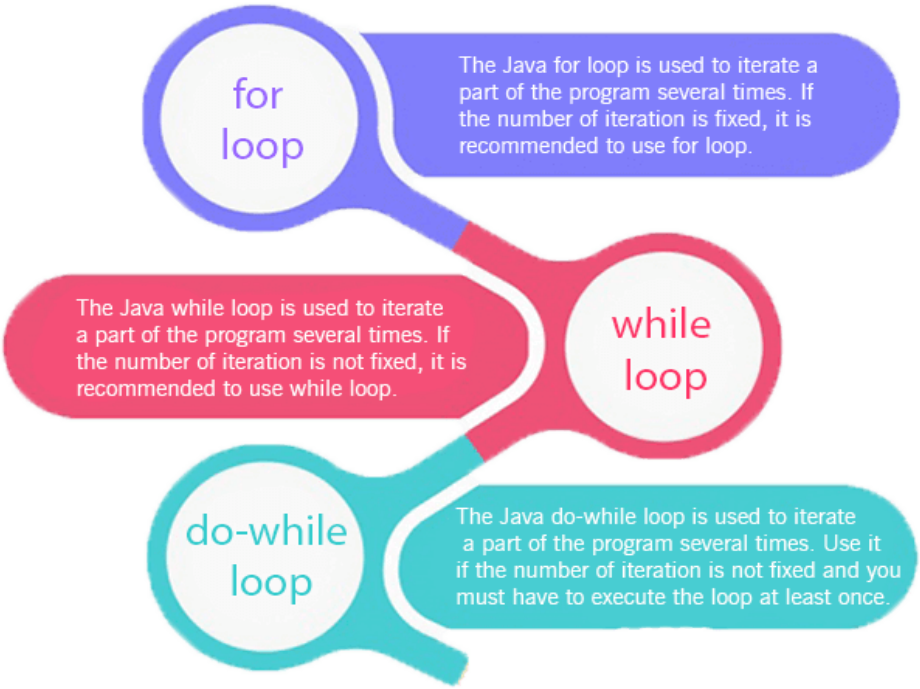
```
    package app;

    public class Ladder {

        public static void main(String[] args)
        {
            if-else-if ladder

            syntax:

             * if(condition1)
             * {
             *    s1,
             * }
             * else if(condition2)
             * {
             *    s2;
             *    }
             *    else if(condition3)
             *    {
             *      s3;
             *      }
             *      else if(condition4)
             *      {
             *        s4;
             *        }
             *
```

```
                *       else
                *       {
                *         s5;
                *         }
                *
                *
                */

                int i=20;
                if(i==10)
                {
                        System.out.println("i is 10");
                }
                else if(i==15)
                {
                        System.out.println("i  is 15");
                }
                else if(i==17)
                {
                        System.out.println("i is 17");
                }
                else if(i==200)
                {
                        System.out.println(" i is 20");
                }
                else
                {
                         System.out.println(" no match found!");
                }
        }

}
```

Q. Explain Loop control statements in java?

Loop:

Loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in Java.



We have two types of Loops they are:

1. **Entry Loop**
2. **Exit Loop**

1. **Entry Loop:**
    - first it will checks the given condition if the given condition is true then only the body of the loop will be executed.
    - it is also known as pre-test condition.
    - EXAMPLE: while,for

While loop:

while:

* it is a keyword in java
* it is one of entry loop condition
* it is also known as pre-test condition
*

Logic:

* First it will check the given condition if the given condition is true then only the body of the loop will be executed.

Syntax:

```
while(condition)
 {
   //body of the loop
```

```
    }
```

Example:

```
package app;

public class Loop1
{

        public static void main(String[] args)
        {
                int a=1;
                while(a<=100)
                {
                        System.out.println(a);   //1
                        a++;                //2
                }
        }
    }
```

Q. Explain for loop with an example?

Ans:

1. **for is a keyword in java.**
2. **It is one of the entry loop control statement, it is also known as pre-test condition.**
3. Logic:
   First it will check the given condition if the given condition is true then only the body of the loop will be executed.
4. **Syntax:**

```
for(initialization;condition;updation)
{
   //body of the loop;
}
```

Example:

```
package app;

public class Loop1
{

        public static void main(String[] args)
        {

      for(int i=0;i<=20;i++)
       {
         System.out.println(i);
```

```
        }

            }
    }
```

Exit Loop:

Definition:
first executes the body of the loop then checks the given condition if the condition is false though @ least oncethe body of the loop will execute.

Example: do while

1. do, while are keywords
2. it is one of the post-test condition
3. logic:first it will executes the body of the loop then checks the given condition if the given condition is false though @ least once the body of the loop will execute.

Syntax:

```
do
{
  //body of the loop;
}while(condition);
```

Example:

```
package app;

public class Loop1
{

        public static void main(String[] args)
        {
                int i=0;
                do
                   {
                       System.out.println(i);
                            i++;
                   }while(i<=10);

        }
    }
```

Output:
0,1,2,3,…….10 because the given condition is true

Nested for loop:

Loop with in a loop is known as nested loop

Syntax:

```
 for(initialization; condition; updation)
{
      // outer for loop
  for(initialization; condition; updation)
   {
          // inner for loop
    }
}
```

Logic:

1. **First outer for loop condition will be checked if the given condition is true then only the inner for loop will be executed.**

Example:

```
package app;

public class NestedForLoop
{

        public static void main(String[] args)
        {


                for(int i=1;i<=5;i++)
                {
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print("*");
                        }
                        System.out.println(" ");
                }
        }

}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Note

```
//nested for loop
                /*
                 * loop within a loop is known as nested loop
                 *
                 * syntax:
                 * for(initialization;condition;updation)
                 * {
                 *    //outer for loop
                 *   for(initialization;condition;updation)
                 *   {
                 *      //inner for loop
                 *    }
                 * }
                 *
                 *  step1: first outer for loop condition will be checked
                 *  if it is true then inner for loop will execute.
                 *
                 *  step2: if outer for loop condition is false then inner for loop
                 *  will not executed.
                 */
```

Jump control statements:

In Java, **Jump statements** are used to unconditionally transfer program control from one point to elsewhere in the program. *Jump statements* are primarily used to interrupt loop or switch-case instantly. Java supports three jump statements: **break, continue, and exit.**

THE break Statement

The break construct is used to break out of the middle of loops: for, do, or while loop. When a break statement is encountered, execution of the current loops immediately stops and resumes at the first statement following the current loop.

That is, we can force immediate termination of a loop, bypassing any remaining code in the body of the loop.

It is mostly used to exit early from the loop by skipping the remaining statements of loop or switch control structures. It is simply written as
break;

- We can have more than one break statement in a loop.
- The break command terminates only the current loop and not any enclosing loops.

Example:

```
class BreakStatement
{
public static void main(String args[])
{
      System.out.println("Show importance of break statement");
        for(int i =1; i<=10; i++)
{
System.out.println("i = "+i);
           if(i==5)
{
              System.out.println("\nBye");
break;
           }
        }
    }
}
```

THE continue STATEMENT

Like the break statement, the continue statement also skips the remaining statements of the body of the loop where it is defined but instead of terminating the loop, the control is transferred to the beginning of the loop for next iteration.

The loop continues until the test condition of the loop becomes false.
When used in the while and do-while loops, the continue statement causes the test condition to be evaluated immediately after it.

But in case of for loop, the increment/decrement expression evaluates immediately after the continue statement and then the test condition is evaluated.
It is simply written as,

continue;

```
/* Print Number from 1 to 10 Except 5 */
class NumberExcept
 {
    public static void main(String args[] )
 {
           int i;
        for(i=1;i<=10;i++)
```

```
        {
            if(i==5)
            continue;
              System.out.print(i +" ");
        }
    }
    }
```

The Exit Statement:

```
package app;

public class Jump_Control
{

        public static void main(String[] args)
        {
                System.out.println("welcome to java class:");
                System.exit(0);
                System.out.println("Good Morning:");
                System.out.println("I'm a BCA student");
                System.out.println("I'm a BSC Student");

        }

}
```

Output:

Welcome to java class

Q. Explain switch condition with an example?

Ans:

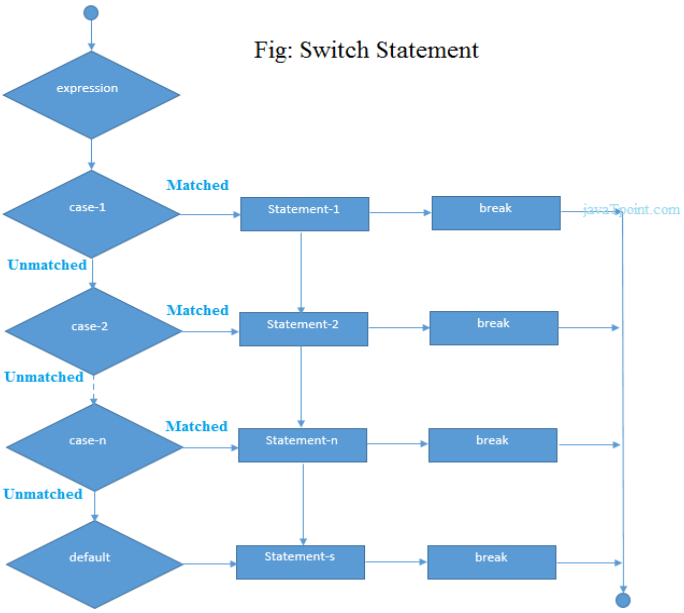


Fig: Switch Statement

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

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

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- 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.
- The case value can have a *default label* which is optional.

Syntax:

```
switch(expression)
{
case value1:
 //code to be executed;
```

```
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

default:
 code to be executed if all cases are not matched;
}
```

Example:

```
public class SwitchExample
{
public static void main(String[] args)
{
   //Declaring a variable for switch expression
   int number=20;
   //Switch expression
   switch(number)
{
   //Case statements
   case 10: System.out.println("10");
   break;
   case 20: System.out.println("20");
   break;
   case 30: System.out.println("30");
   break;
   //Default case statement
   default:System.out.println("Not in 10, 20 or 30");
   }
}
}
```