1. INTRODUCTION TO JAVA

❖ History of JAVA:-

➢ Around 1990 James Gosling, Bill Joy and others at Sun Microsystems began developing a language called Oak. The wanted it primarily to control microprocessors embedded in consumer items such as cable set-top boxes, VCR's, toasters, and also for personal data assistants (PDA).

➢ To serve these goals, Oak needed to be:
- Platform independent (since multiple manufacturers involved)
- Extremely reliable
- Compact.

➢ However, as of 1993, interactive TV and PDA markets had failed to take off. Then the Internet and Web explosion began, so Sun shifted the target market to Internet applications and changed the name of the project to Java.

➢ By 1994 Sun's HotJava browser appeared. Written in Java in only a few months, it illustrated the power of applets, programs that run within a browser, and also the capabilities of Java for speeding program development.

➢ Riding along with the explosion of interest and publicity in the Internet, Java quickly received widespread recognition and expectations grew for it to become the dominant software for browser and consumer applications.

➢ However, the early versions of Java did not possess the breadth and depth of capabilities needed for client (i.e. consumer) applications. For example, the graphics in Java 1.0 seemed crude and clumsy compared to mature software developed with C and other languages.

➢ Applets became popular and remain common but don't dominate interactive or multimedia displays on web pages. Many other "plug-in" types of programs also run within the browser environment.

➢ So Java has not succeeded at development of consumer applications. However, Java's capabilities grew with the release of new and expanded versions (see below) and it became a very popular language for development of enterprise, or middleware, applications such as on line web stores, transactions processing, database interfaces, and so forth.

➢ Java has also become quite common on small platforms such as cell phones and PDAs. Java is now used in several hundred cell phone models. Over 600 million JavaCards, smart cards with additional features provided by Java, have been sold as of the summer of 2004.

❖ What is JAVA?

- The term Java actual refers to more than just a particular language like C or Pascal. Java encompasses several parts, including :
- A high level language:- The Java language is a high level one that at a glance looks very similar to C and C++ but offers many unique features of its own.
- Java bytecode:- a compiler, such as Sun's javac, transforms the Java language source code to bytecode that runs in the JVM.
- Java Virtual Machine (JVM):- A program, such as Sun's java, that runs on a given platform and takes the bytecode programs as input and interprets them just as if it were a physical processor executing machine code.

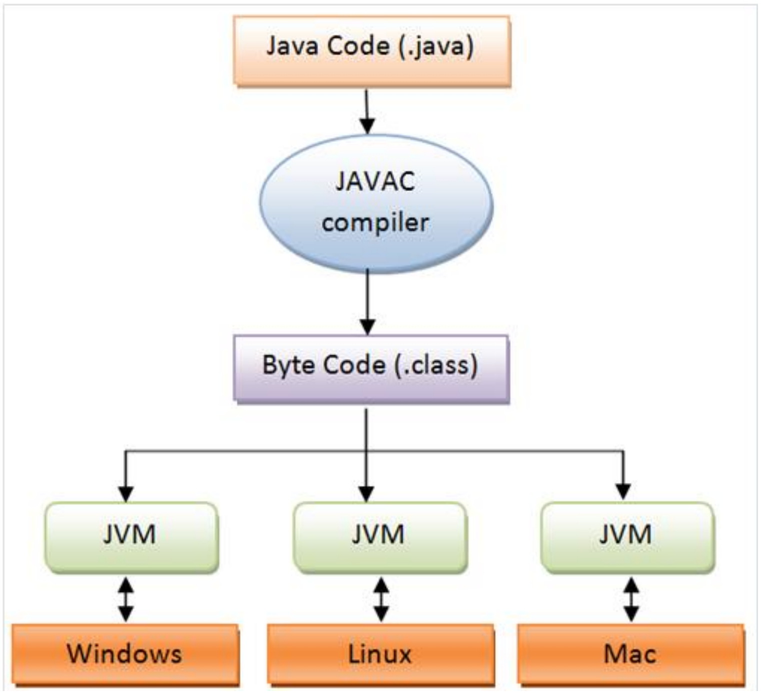
➢ Sun provides a set of programming tools such as javac, java and others in a bundle that it calls a Java Software Development Kit for each version of the language and for different platforms such as Windows, Linux, etc.

➢ Sun also provides a runtime bundle with just the JVM when the programming tools are not needed.

- Note that because of the open nature of Java (see below), any or all of these parts can be replaced by non-Sun components. For example, just as many different languages can create machine code for a given processor, compilers of other languages have been created that output bytecode to run in the JVM. Similarly, many JVMs have been written by groups outside of Sun.
- In this book and web course, when we use the term Java we are referring to the the high level language unless noted otherwise. Also, those packages that come with the SDK for a given version will be referred to as comprising the core language for that version, as distinguished from optional or third party packages.

❖ Java Virtual Machine(JVM):

- Java Virtual Machine or JVM as its name suggest it is a “virtual” machine that resides in the “real” computer as a software.
- JVM gives Java the flexibility of platform independence. Let us see first how exactly Java program is created, compiled and executed.
- Java code is written in **.java** file.
- This code contains one or more Java language attributes like Classes, Methods, Variable, Objects etc.
- Javac is used to compile this code and to generate .class file. Class file is also known as “byte code“.
- The name byte code is given may be because of the structure of the instruction set of Java program.
- Java byte code is an input to Java Virtual Machine. JVM read this code and interpret it and executes the program



- Java Virtual Machine like its real counterpart, executes the program and generate output. To execute any code, JVM utilizes different components.
- JVM is divided into several components like the stack, the garbage-collected heap, the registers and the method area. Let us see diagram representation of JVM.

❖ Features of JAVA:

- Simple
- Object oriented
- Compiled and interpreted

- Platform independent
- Robust and secure
- Distributed
- Multithreaded
- High performance
- Dynamic

- **Simple:-**
  - Java is a small and simple language. Java doesn't use pointers, preprocessor header files, GOTO statement and many others. It also eliminates operator overloading and multiple inheritance.

- **Object oriented:-**
  - Java is a true object oriented language. All program code and data reside within objects and clases.java becomes extensive set of class.
  - Advantage of Object oriented programming:
    - Simpler, easier to read programs
    - More efficient reuse of code
    - Faster time to market
    - More robust, error-free code

- **Compiled and interpreted:-**
  - Usually a computer language is either compiled or interpreated.Java combines both these features thus making java a Two-stage system.
  - java compiler translates source code into bytecode instructions.
  - Bytecodes are not machine instructions and therefore in the second stage Java interpreter generates machine code then execute the java program.
  - Thus we can say java is both compiled and interpreted language.

- **Platform independent and portable:-**
  - The most important features of java over other languages are its portability.
  - Java programs can be easily moved from one computer to another, anywhere and any timke.changes and upgrades in operating systems, and system resources will not force any changes in java programs.
  - Java ensures portability in two ways. First java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the sizes of the primitive data types are machine-independent.

- **Robust and secure:-**
  - Java is a robust language. it provides many safeguards to ensure reliable code.
  - It has strict compile time and run time checking for data types.
  - Security becomes an important issue for a language that is use for programming on internet.
  - Threat of viruses and abuse of resources are everywhere. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.

- **Distributed:-**
  - Java is designed as a distributed language for creating applications on networks. It has ability to share both data and programs.

- **Multithreaded:-**

- Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs.
- This means that we need not wait for the application to finish one task before beginning another.

➢ **High performance:-**

- Java performance is impressive for an interpreted language, due to the use of intermediate byte code.

➢ **Dynamic:-**

- Java is a dynamic language. Java is capable of dynamically linking new class libraries, methods, and objects.

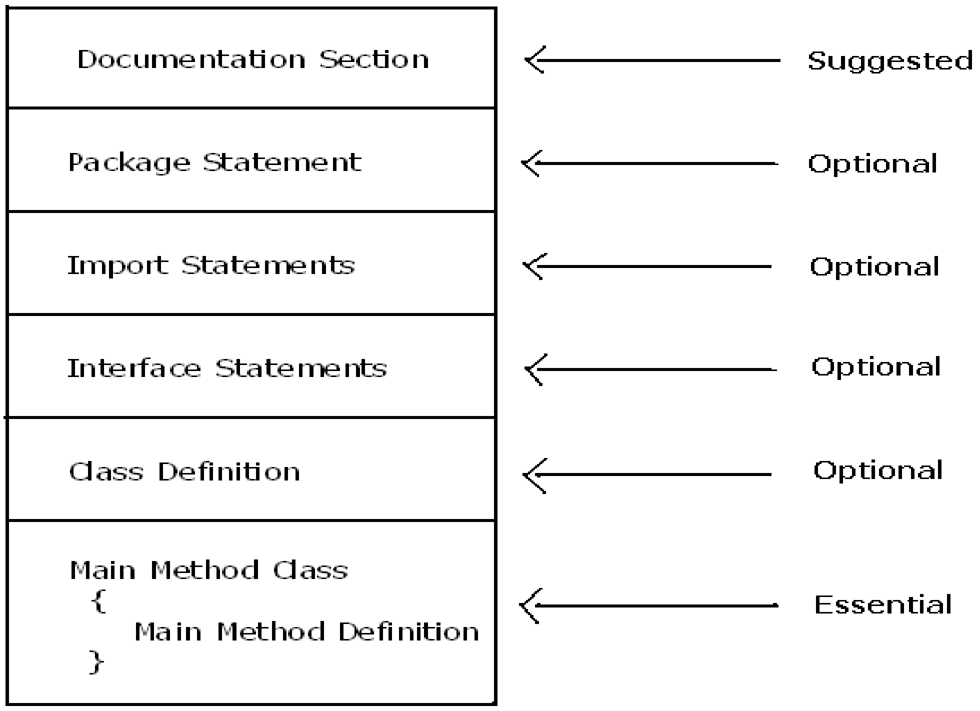
❖ Java Development Kit(JDK):-

➢ The java development kit comes with a collection of tools that are used for developing and running java programs. They include:

TOOL	**DESCRIPTION**
Appletviewe	Enables us to run java applets
Javac	The java compiler, which translates java source code to bytecode file.
Java	Java interpreter that interpreting bytecode files.
Javadoc	Creates HTML –format document from java source code files.
javah	Produces header files
javap	Java disassembler, which enables us to convert bytecode into program description.
Jdb	Java debugger, which helps us to find errors in our programs.

➢ The JDK. It consists of a framework of about 3700 classes. These classes must be implemented according to this specification in every JDK. Sun defines this specification.
➢ The Java Platform, or JDK, has evolved over time so there are several versions of it.
➢ The latest major version is Java SE 7 or JDK 7.
➢ Our applications are mostly written against either version 1.4 or 5, which are both still supported by Sun.
➢ Even though the Java Platform is often called the JDK, the correct definition is very different.
➢ As originally defined, the Java Development Kit (JDK) is actually a combination of Java compilation tools and API implementation for a particular version of the Java Platform.
➢ It also typically includes a Java runtime (JRE), so that you can run the programs you compile.

❖ Explain Java Program Structure?



- **Documentation Section**
  - The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. We use comment for documentation.

- **Package Statement**
  - The first statement allowed in a Java file is a package statement.
  - This statement declares a package name and informs the compiler that the classes defined here belong to this package.
  - The package statement is optional. That is, our classes do not have to be part of a package.

- **Import Statements**
  - The next thing after a package statement (but before any class definitions) may be a number of import statements. This is similar to the #include statement in C.

- **Interface Statement**
  - An interface is like a class but includes a group of method declarations.
  - It is new concept in java. This is also an optional section and is used only when we wish to implement the multiple inheritance features in the program.

- **Class Definitions**
  - A Java program may contain 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. The number of classes used depends on the complexity of the problem.

- **Main Method Class**
  - Since every Java stand –alone program requires a main method as its starting point ,this class is the essential part of a Java program.

❖ Explain the Basic Structure of JAVA Program.

➢ Let us see now how the simple java program will look like. You can use any editor like notepad or any Java IDE for writing java programs.

```
class Sample
{
        public static void main (String args[])
        {
                System.out.println ("JAVA IS BETTER");
        }
}
```

- Here **class Sample** declares a class
- **class** is a keyword and declares that a new class definition.
- Every class definition in java begins with an opening brace " { " and ends with a matching brace " } ",appearing in the last line in the example.
- **public static void main(String args[ ])**
- Defines a method name **main.** Conceptually this is similar to the main () function in C/C++.
- Every java application program must include the **main ()** method.
- This line contains a number of keywords, public, static and void.

public	The keyword public is an access specifier that declares the main method as unprotected & therefore making it accessible to all other classes.
static	The keyword Static, which declares this method as one that belongs to the entire class and not a part of any object
void	The type modifier void states that the main method does not return any value.

- The only executable statement in the program is **System.out.println("JAVA IS BETER");**
- The **println()** method is a member of the **out** object, which is a static data member of system class.

❖ DATA TYPES:

➢ Every variable in java has a data type. Data types specify the size and type of values that can be stored. The variety of data types available allows the programmer to select appropriate types.

➢ There are two data types available in Java:
- **Primitive Data Types**
- **Reference/Object Data Types**

➢ Primitive Data Types:
- There are eight primitive data types supported by Java.
- Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

➢ **PRIMITIVE DATA TYPES:**

- **Primitive data types is divided into four groups.**

Group Name	Data types	Size(in bits)	Range
Integers	byte	8	-128 to 127
	short	16	-32768 to 32767
	int	32	-2^{31} to $2^{31}-1$
	long	64	-2^{63} to $2^{63}-1$
Floating point numbers	float	32	1.4e-045 to 3.4e+038
	double	64	4.9e-324 to 1.8e+308
Characters (also use Unicode characters)	char	16	0 to 65536
Boolean	boolean	1	NA

❖ ESCAPE SEQUENCE IN JAVA

➢ Java language supports few special escape sequences for String and char literals as well. They are:

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

❖ KEYWORDS IN JAVA

➢ Keywords are reserved words that are predefined in the language; see the table below. All the keywords are in lowercase.

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

➢ An identifier is composed of a sequence of characters where each character can be a letter, a digit, an underscore, or a dollar sign.

- An identifier declared by the programmer cannot be a Java keyword and it cannot start with a digit. For instance:

❖ OPERATORS IN JAVA

- 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
  - Logical operators
  - Assignment operators
  - Conditional(ternery) operators
  - Bitwise operators

- **Arithmetic Operators:**
  - Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:
  - Assume integer variable A =10 and variable B =20 then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increase the value of operand by 1	B++ gives 21
--	Decrement - Decrease the value of operand by 1	B-- gives 19

- The following simple example program demonstrates the arithmetic operators. Copy and paste following Java program in Test.java file and compile and run this program.

```
class Test
{
public static void main(String args[])
 {
   int a = 10;
   int b = 20;
   int c = 25;
   int d = 25;
   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) );
   System.out.println("a++   = " +  (a++) );
   System.out.println("b--   = " +  (a--) );
   // Check the difference in d++ and ++d
   System.out.println("d++   = " +  (d++) );
   System.out.println("++d   = " +  (++d) );
```

```
 a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```

```
        }
    }
```

➢ Relational Operators:

- There are following relational operators supported by Java language
- Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

- The following simple example program demonstrates the relational operators. Copy and paste following Java program in Test.java file and compile and run this program.

```
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("a < b = " + (a < b) );
  System.out.println("b >= a = " + (b >= a) );
  System.out.println("b <= a = " + (b <= a) );
  }
}
```

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

➢ Logical Operators:

- The following table lists the logical operators:
- Assume Boolean variables A holds true and variable B holds false then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A || B) is true.

!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

```
class Test
{
  public static void main(String args[])
  {
   int a = true;
   int b = false;

   System.out.println("a && b = " + (a&&b) );

   System.out.println("a || b = " + (a||b) );

   System.out.println("!(a && b) = " + !(a && b) );
  }
}
```

Bitwise Operator:

Java defines several *bitwise operators* which can be applied to the integer types, **long**, **int**, **short**, **char**, and **byte**. These operators act upon the individual bits of their operands.
They are summarized in the following table:

Operator	Name
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
<<=	Shift left assignment

Left Shift

The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times. It has this general form:

value << num

Example:

```
class ByteShift
{
        public static void main(String args[])
        {
                byte a = 64, b;
                int i;
                i = a << 2;
                b = (byte) (a << 2);
                System.out.println("Original value of a: " + a);
                System.out.println("i and b: " + i + " " + b);
        }
}
```

OUTPUT

```
Original value of a: 64
i and b: 256 0
```

- **Right Shift**

  The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times. Its general form is shown here:

  value >> num

  Example:

  ```
  int a = 35;
  a = a >> 2; // a still contains 8
  ```

  Looking at the same operation in binary shows more clearly how this happens:

  ```
  00100011    35
    >> 2
  00001000     8
  ```

- **Assignment Operators:**

  - There are following assignment operators supported by Java language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C

- **Conditional Operator (? :):**
  - Conditional operator is also known as the ternary operator.
  - This operator consists of three operands and is used to evaluate boolean expressions.
  - The goal of the operator is to decide which value should be assigned to the variable.
  - Conditional operator is written as :

    variable x = (expression) ? value if true : value if false

  - Following is the example:

```
class Test {
  public static void main(String args[]){
    int a , b, c;
    a = 10;
    b=20;
    c = (a<b) ? 20: 30;
    System.out.println( "Value of b is : " +  c);
  }
}
```

❖ Precedence of Java Operators:

- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:
- For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Category	Operator	Associativity
Postfix(Highest)	**() [] . (dot operator)**	Left to right
Unary	**++ - - ! ~**	Right to left
Multiplicative	*** / %**	Left to right
Additive	**+ -**	Left to right
Shift	**>> >>> <<**	Left to right
Relational	**> >= < <=**	Left to right
Equality	**== !=**	Left to right
Bitwise AND	**&**	Left to right
Bitwise XOR	**^**	Left to right
Bitwise OR	**|**	Left to right
Logical AND	**&&**	Left to right
Logical OR	**||**	Left to right
Conditional	**?:**	Right to left
Assignment	**= += -= *= /= %= >>= <<=**	Right to left
Comma(Lowest)	**,**	Left to right

❖ Java Control Statements:

➢ Control statements control the order of execution in a java program, based on data values and conditional logic. There are three main categories of control flow statements;
- **Selection statements:** if, if-else, nested if-else, else-if ladder and switch.
- **Iteration statements:** while, do-while and for.

➢ We use control statements when we want to change the default sequential order of execution

❖ Selection Statement

➢ If Statement

- if statement executes a block of code only if the specified expression is true.
- If the value is false, then if block is skipped and execution continues with the rest of the program.
- You can either have a single statement or a block of code within if statement.
- Note that the conditional expression must be a Boolean expression.
- **The simple if statement has the following syntax:**
  - ***if (<condition>) <statement>***
- Below is an example that demonstrates conditional execution based on if statement condition.
- **Example:**

```
class IfDemo
{
        public static void main(String[] args)
        {
                int a = 67;
                if(a <100)
                {
                        System.out.println("a is less than 100");
                }
        }
}
```

➢ If-else Statement:-

- if-else statement is an extension of the if statement.
- Here condition in if statement is false then it will goes into the else block.
- Note that the conditional expression must be a Boolean expression.
- **Syntax:**

```
if(condition)
    statement1;
else
    statement2;
```

- Below is an example that demonstrates conditional execution based on if else statement condition.
- **Example:-**

```
class IfElseDemo
{
    public static void main(String[] args)
    {
            int a = 10, b = 20;
            if(a > b)
            {
                    System.out.println("a is maximum");
            }
            else
            {
                    System.out.println("b is maximum");
            }
        }
    }
```

➢ Nested If-else:-

Syntex:-

- The syntax for a nested if...else is as follows:

```
if(condition1)
{
    if(condition2)
    {
        Statement-1;
    else
        Statement-2;
    }
else
    Statement-3;
}
```

- In nested if else logic if condition1 is true then check condition2 if it is true then statement1 will be execute else statement2 will be execute.
- In nested if else logic if condition -1 is false then statement -3 is execute.

- **Example:**

```
class NestedIfElse
{
    public static void main(String args[])
    {
        int a=10;
        int b=20;
        int c=30;

        if (a>b)
        {
            if (a>c)
            {
                System.out.println("a is max ");
            }
            else
            {
                System.out.println("c is a max");
            }
        }
        else
        {
            if (b>c)
            {
                System.out.println("b is max");
            }
            else
            {
                System.out.println("c is a max");
            }
        }
    }
}
```

➢ <u>If-else-if ladder:</u>

- In this if-else-if ladder, conditions are evaluated from top. First condition-1 will be evaluated and if this is true, the code inside the if block will be executed. If condition-1 is false, condition-2 will be evaluated.
- If condition-2 is true, the code inside that else if block will be executed. If condition-2 is false, condition-3 will be evaluated. This will go on like this.
- If none of the conditions are true, the code inside the else block will be executed.
- **Syntex:**

```
if(condition-1)
{
        Statement1;
}
else if (condition-2)
{
        Statement2;
}
.
.
.
else if (condition-n)
{
      Statement n-1;
}
else
{
      Statement n;
}
```

Example:-

```
public class IfElseIfLadderDemo1
      {
      public static void main(String args[])
                  {
         int test=2;

            if(test==1)
             {
                        System.out.println("Hello");
             }
            else if(test==2)
         {

           System.out.println("Hi");
         }
             else if(test==3)
          {
           System.out.println("Good");
          }
             else
             {
           System.out.println("No Match Found");
              }
         }
     }
```

➢ Iteration statement(Looping structure):-

- Loop is the control statement of any language in which whenever you want to perform the repetitious work then you use the **Loop control statement**.
- There are mainly three types of loops.
- Loop repeats a statement or a process multiple times according to the specified conditions.
- It allows the multiple statements or process to be run for the specified time or it also follows the certain conditions.
- Loop makes your program readable, flexible and reliable.

- **While Loop:**
  - While loop checks the certain condition first, if the condition is true then all the statements or processes written under the while loop are executed otherwise ignored all.
  - **Syntax**:

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

  - In this program you will see how to use the while loop statement.
  - This program takes the 1 to 10 number.

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

- **do-while:**
  - In do-while loop, compiler executes some statements and processes once at least.
  - do-while loop is same as the while loop statement but while loop checks the condition is first and in do-while loop executes all the statements first at once and then check the condition if the condition is true then all the statements are also executed second times otherwise loop terminates.
  - **Syntax** :

```
Initialization;
do
 {
    statements;


 }while(condition);
```

  - In this example you will see how to use the do-while loop statement.

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

}

- **for loop:**
  - for loop is often used when you want to repeat a section of a program for a fixed number of times.
  - **Syntax:**

```
for( initialization; condition; increment/decrement)
{
    statements;
}
```

  - **Initialization:** It allows the variable to be initialize. Such as:
    int i = 1;
    int j = 1;
  - **condition:** It allows to check the certain condition. If condition is true then all statements and processes written in the for block will be executed otherwise ignored
  - Condition such as:
    i <= 5;
    j <= i;
  - **Increment:** It allows the how much increase the given variable. Such as:
    i++;
    j++;
  - **Example:-**

```
class test
{
        public static void main(String[] args)
        {
                int i;
                for(i=1;i<=10,i++)
                {
                        System.out.println("Number : " + i);
                }
                System.out.println(“Loop Complete”);
        }

}
```

❖ switch statement:

- An alternative to a series of else if is the switch statement.
- The switch statement allows you to choose a block of statements to run from a selection of code, based on the return value of an expression.
- The expression used in the switch statement must return an int or an enumerated value.
- The syntax of the switch statement is as follows.

```
switch (expression)
{
    case value_1 :
        statement (s);
        break;
    case value_2 :
        statement (s);
        break;
      .
```

```
                .
                .
                case value_n :
                    statement (s);
                    break;
                default:
                    statement (s);
            }
```

- **Example:**

```
            class test
            {
                    public static void main(String[] args)
                    {
                            int i = 1;
                            switch (i)
                            {
                                    case 1 :
                                            System.out.println("One.");
                                            break;
                                    case 2 :
                                            System.out.println("Two.");
                                            break;
                                    case 3 :
                                            System.out.println("Three.");
                                            break;
                                    default:
                                            System.out.println("You did not enter a valid value.");
                            }
                    }
            }
```

Break statement:

- The break statement is used to break from an enclosing do, while, for, or switch statement.
- It is compile errors to use break anywhere else.
- 'break' breaks the loop without executing the rest of the statements in the block.
- For example, consider the following code

```
    class test
    {
        public static void main(String[] args)
        {
            int i = 0;
            while (i<100)
            {
                if (i > 3)break;
                System.out.println(i);
            }
        }
    }
```

❖ continue statement:

- The continue statement stops the execution of the current iteration and causes control to begin with the next iteration.
- For example, the following code prints the number 0 to 9, except 5.

```
class test
{
    public static void main(String[] args)
    {
        for (int i = 0; i < 10; i++)
        {
                if (i == 5)
                {
                        continue;
                }
                System.out.println(i);
        }
    }
}
```

❖ Array:

- An array is a very common type of data structure where in all elements must be of the same data type.
- Once defined, the size of an array is fixed and cannot increase to accommodate more elements.
- The first element of an array starts with zero
- Using and array in your program is a 3 step process -
  - Declaring your Array
  - Constructing your Array
  - Initializing your Array
- **Syntax for Declaring Array Variables is**

  <elementType>[] <arrayName>;
- **Example:**

  int a[];
  int []a;

- **Constructing an Array**

  = new [];
- **Example:**
  - a = new int[10]; // Defines that intArray will store 10 integer values

- **Declaration and Construction combined**
  - int a[] = new int[10];

- **Example:First Array Program(One dimensional)**

```
class test
{
    public static void main(String args[])
    {
        int a[] = new int[3];
        int i;
        for (i=0;i<3;i++)
        {
            a[i]=i+1;
        }
        for(i=0;i<3;i++)
        {
            System.out.println("Value of array"+a[i]);
        }
    }
}
```

Multidimensional arrays:

- Multidimensional arrays, are arrays of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.
- int twoD[ ][ ] = new int[4][5] ;
- When you allocate memory for a multidimensional array, you need only specify the memory for the first (leftmost) dimension.
- You can allocate the remaining dimensions separately.

Example:-1

```
class test
{
    public static void main(String args[])
    {
        int a[ ][ ];
        a=new int[3][3];
        int i,j,k=0;
        for (i=0; i<3; i++)
        {
            for(j=0;j<i;j++)
            {
                a[i][j] = k;
                k++;
            }
        }
        for (i=0; i<3; i++)
        {
            for(j=0;j<i;j++)
            {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

Example:-2

```
class test
{
    public static void main(String args[])
    {
        int a[ ][ ] = { { 2, 1, 3 }, { 1, 3, 2 }, { 3, 2, 1 } };
        int I,j;
```

```
            for (i=0; i<3; i++)
            {
                for(j=0;j<i;j++)
                {
                    System.out.println(a[ I ][ j ]);
                }
            }
        }
    }
```

❖ **<u>Difference between JRE, JDK and JVM:</u>**

➢ In short here are few differences between JRE, JDK and JVM:

- JRE and JDK come as installer while JVM are bundled with them.
- JRE only contain environment to execute java program but doesn't contain other tool for compiling java program.
- JVM comes along with both JDK and JRE and created when you execute Java program by giving "java" command.

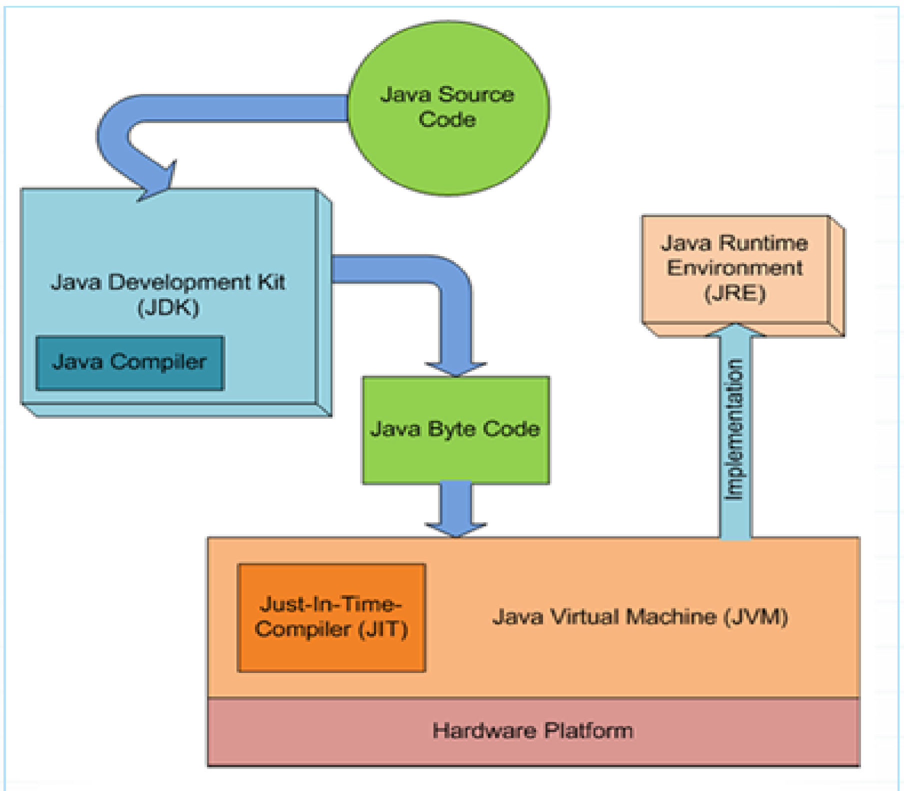

Java Source Code
Java Development Kit (JDK)
Java Compiler
Java Runtime Environment (JRE)
Java Byte Code
Implementation
Just-In-Time-Compiler (JIT)
Java Virtual Machine (JVM)
Hardware Platform

Diagram to show the relations between JVM JRE JDK

➢ **<u>Difference between Java and C</u>**

JAVA	C
Java is object-oriented language.	C is procedure oriented language.
Java does not support structure, union. enum.	C supports structure,union,enum.
Java does not support Pointer.	C supports pointer.

Java does not support Global variable	C supports Global variable
Java does not support go to, sizeof, typedef.	C supports goto, sizeof, typedef.
There are no header files in java	There is no. of header files in C.
Java does not support Pre-processor.	C support Pre-processor.
Java does not define Type modifiers keyword Auto, extern, register, signed, unsigned.	C defines Type modifiers keyword Auto, extern, register, signed, unsigned.

- **<u>Difference between Java and C++.</u>**

JAVA	**C++**
Java does not support Operator overloading.	C++ supports Operator overloading.
Java does not support Template class.	C++ support Template class.
Java does not support Multiple inheritance but it supports interface.	C++ supports Multiple inheritance.
Java does not support Global variable	C++ support Global variable
Java does not use Pointer.	C++ use Pointer.
There are no header files in java	There is no. of header files In C++.
Java has replace destructor Function with finalize ( ).	C++ has a constructor & also Destructor Function.
Java does not support Pre-processor.	C++ support Pre-processor.