Introduction & History of JAVA:

- The Java programming language is a general purpose, concurrent, class based, robust, secure(no pointer concept as in C/C++), safe and object oriented programming language.
- It is designed to be simple enough that many programmers can achieve fluency in the language.
- Java was created by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan combine a team of programmers at Sun Microsystems, Inc. in 1991.
- This language was initially called 'Oak' but it was renamed to Java in 1995.
- The original goal was to develop a language that could be used to write software for different platform support.
- Java was designed for the development of consumer electronic devices like TVs, VCRs, Toasters and such other electronic machines. So the basic aspect of Java was to create a computer language that could be used to build programs that can run on all environments.
- Because these machines provide a variety of hardware and software environments, it was necessary for Java to be platform independent (Architecture Neutral).
- Java was the first programming language that is not tide to any particular hardware of any computer system.
- The World Wide Web which was emerging at that time had played a crucial role in the future of Java.
- Java satisfied need of demanded portable programs.
- Internet allows many different types of computers to be connected together including all computers having different CPUs and Operating Systems. Therefore, a ability to write a program is as beneficial to be transferred over the internet. Java's "Write Once Run Any Where" philosophy provides this advantage.
- The inventor of Java examined many different programming languages and adopted their best features.
- The Java programming language is related to C and C++ but it is organized rather differently with a number of aspects of C and C++ are discarded and few ideas from other powerful languages are added.
- The Java programming language is strongly typed language that means it is clearly specifies the difference between the compile time and run time errors.
- Compile time normally consist of translating programs into machine independent byte code representation.
- Run time activities include loading and linking of the classes needed to execute a program, optional machine code generation and dynamic optimization of the program with the actual program execution.
- Java has two lives one as standalone computer language for general purpose programming (known as applications) and other as a supporting language for internet programming (known as applets).

Features of JAVA:

SIMPLE

- The Java programming language is a high level language in which details of the machine representations are not available to the language.
- Java is a small and simple language. Many features of C and C++ are either redundant or sources of unreliable code are not a part of Java.

- It includes automatic storage management, typically using a garbage collector to avoid the safety problems of explicit de-allocation of memory reference (free () in C and delete in C++).
- Java doesn't use pointers, pro processors, and header files, go to statements and many more. It also eliminates operator overloading and multiple inheritance.
- Java uses many constructs of C and C++ therefore, Java code looks like a C++ code.

OBJECT ORIENTED

- Almost everything in Java is an object. All the programs and data reside within the objects and classes.
- Java comes with extensive set of classes arranged in a packages that we can use in our program by inheritance.
- The major difference between Java and CPP is in multiple inheritance which Java has replaced with the simple concept of interfaces.

DISTRIBUTED

- Java is designed as distributed language for creating application on networks. It has an ability to share both data and programs
- Java applications can open and access remote objects on internet as easily as they can do with local systems.
- This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

ROBUST AND SECURE

- Java provides many safe guards to ensure reliable code. It has strict compile time and run time checking of data.
- Java includes the concept of exception handling which captures various errors and eliminates any risk of crashing the system.
- For internet programming, security is a major issue. Java systems not only verify all memory access but also ensure that no viruses are communicated with an Applet.
- The absence of pointer in Java ensures that programs cannot gain an access to memory location without prior authorization.
- Java enables the construction of virus free and tempered free systems. Java was designed to make certain kinds of attacks impossible like.
  - Corrupting memory outside its own process space
  - Reading or writing files without permission.

ARCHITECTURE NEUTRAL [PLATFORM INDEPENDENT]

- Instead of creating machine code file, Java compiler creates a byte code file and byte code are not for any specific CPU but they are designed to be interpreted by a Java Virtual Machine.
- The same byte code can be interpreted by any Java Virtual Machine at any platform

PORTABILITY

- The most significant and important difference between Java and other languages is its portability.
- Java programs can be easily moved from one computer system to another. Changes and up gradation in Operating System, processor and system resources will not force any change in Java program.

- We can download a Java Applet from a remote computer in to our local system via internet and execute it locally.
- Java ensure portability in two ways…
    1. Java compiler generates byte code instructions that can be implemented on any machine
    2. The size of primitive data type in Java is machine independent

COMPILED AND INTERPRETED

- Generally, a computer language is either compiled or interpreted. Java combines both these approaches which make Java a two stage system.
- First Java compiler translate source code in to byte code and in second stage, Java interpreter Generates machine code that can be directly executed by the machine that is running the Java program.

HIGH PERFORMANCE

- Java's performance is impressive for an interpreted language mainly due to the use of intermediate byte code.
- Java's architecture is also designed to reduce overheads during run time. Furthermore, the concept of Multithreading enhances the overall execution speed of Java program.

MULTITHREADING

- Multithreading means handling simple tasks simultaneously.
- Java supports multithreaded programs, it means that we don't have to wait for an application to finish one task before beginning the other. For e.g. we can be listening an audio clip while scrolling a page at the same time downloading an applet from a distant computer.
- This feature improves the interactive performance for Graphical applications.

DYNAMIC

- Java is capable of dynamically linking new class libraries, methods and objects.
- Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program depending on the response.
- Java programs support functions written in other languages such as C and C++. These functions are called native methods and it can be linked dynamically at runtime.

JAVA Editions:

Since Java's invention, Sun has released a new version of the Java Language every two years or so. These new versions brought enhancement, new capabilities and fixes to bugs.
Until recently, the versions of Java were numbered 1.x where x reached up till 4 (Intermediate revisions were labelled with a third number i.e. 1.x.y). The newest version however is called Java 5.0 rather than Java 1.5
Below given are the different versions of the basic, or Standard Edition of java along with some of the new features that are added.

1995 : Version 1.0 of Java Development Kit (JDK)

- Code named Oak
- 8 packages with 212 classes
- Netscape 2.0 – 4.0 include Java 1.0
- Microsoft and other companies licensed Java

1997 : Version 1.1

- 23 packages – 504 classes
- Improvements include better event handling, inner classes, improved JVM
- Microsoft developed its own 1.1 compatible JVM for the Internet Explorer
- Many browsers in use are still compatible only with 1.1
- Availability of improved Swing packages
- Includes Inner classes, JDBC, Java Beans and RMI

1999 : Version 1.2 also called Java 2 Platform

- Code named Playground
- 59 packages with 1520 classes
- Code and tools distributed as the Software Development Kit (SDK)
- Collections API included support for various lists, sets and hash maps

2000 : Version 1.3

- Code named Kestrel
- 76 packages with 1842 classes
- Performance enhancements including the Hotspot virtual machine
- Java sound and JAR indexing

2002 : Version 1.4

- Code named Merlin
- 135 packages with 2991 classes
- Improved IO, XML support etc
- IPv6 support, Chained Exception etc

2004 : Version 5.0

- Code Named Tiger
- 165 packages, over 3000 classes
- Faster start up and smaller memory footprint
- Metadata
- Formatted output
- Generics, varargs
- Improved multithreading features
- Autoboxing / unboxing

2006 : Version 6.0

- Code Named Mustang
- Scripting Language support
- Integrated Web Services etc

2008 : Version 7.0

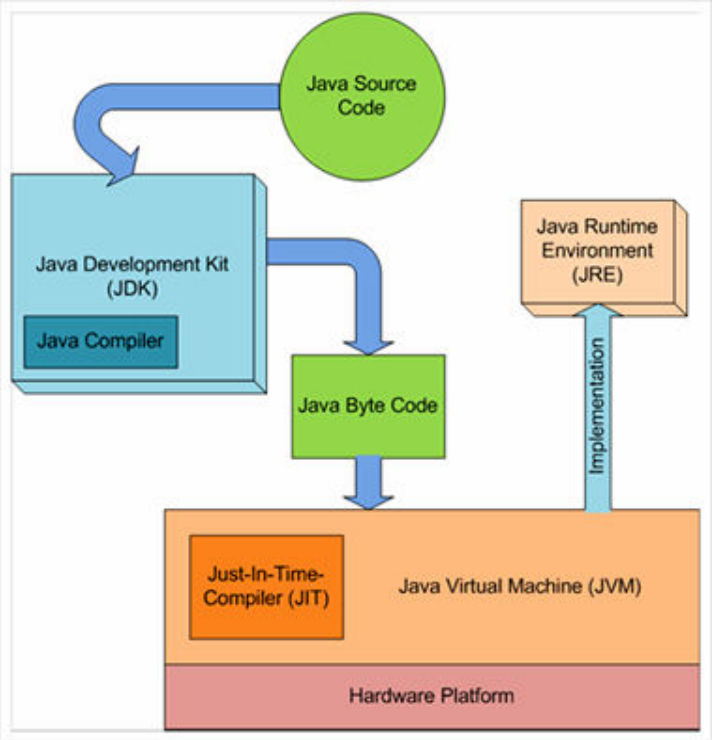
- Code Named Dolphin
- Sting in switch
- Multiple Exception Handling

JVM, JRE , JDK and its Tools:

- Program exist in two forms that is Source code and object code. Source code is the textual version of the program that we write using a text editor. The executable form of a program

is an object code. Object code can be executed by the computer. Typically object code is specific to a particular CPU therefore; it cannot be executed on different platform.

- But in Java program, compiler produces a .class file (Byte code file) instead of creating executable code (object code).
- Byte code are the instructions that are not for any CPU but they are designed to be interpreted by JVM.
- The Java Virtual Machine (JVM) is the virtual machine that runs the Java byte code. The JVM doesn't understand Java typo, that's why we compile our .java files to obtain .class files that contain the byte code understandable by the JVM.



- It is a simulator of computer within the computer and does all the major functions of a real computer.
- It is also the entity that allows Java to be a 'portable language' (*write once, run anywhere*).
- Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine and other components to run applets and applications written in Java. In addition two key development technologies are part of JRE i.e. is Java Plug-In which enables applets to run in popular browser and Java Web Start which deploys standalone application over a network.
- JRE doesn't contain tools and utilities such as compilers or debuggers for developing applications and applets.
- Java environment includes a large numbers of developing tools and hundreds of classes and methods. The development tools are the part of the system known as Java Development Kit (JDK) while the classes and methods are the part of Java Standard Library (JSL) which is also known as Application Programming Interface.
- Java Development Kit comes with a collection of tools that are used for developing and running java programs.
  They are as follows:

javac (Java Compiler)

- The javac compiles java source code and produces byte code class file.

- The source code file must have .java suffix while the resulting class file have a .class suffix
- The file must contain one public class with the class name same as the file name.

java (Java Interpreter)

- The Java interpreter runs java applications. It loads the application's class file and invokes the main
- Method of specific class which must be public, void and static.
- There are numbers of option available for java. The format of java interpreter is as follow:
- java [option] classname [program parameters]
- Note that we don't have to specify .class suffix in the class name.

jdb (Java debugger)

- The Java debugger is used to monitor the execution of Java programs in support of debugging and test activities.

appletviewer

- The appletviewer is used to run Java Applets and test the applets that we develop
- The appletviewer creates a window in which an applet can be viewed
- Syntax is appletviewer [option] URL
- The debug is the only option supported by an appletviewer

javah

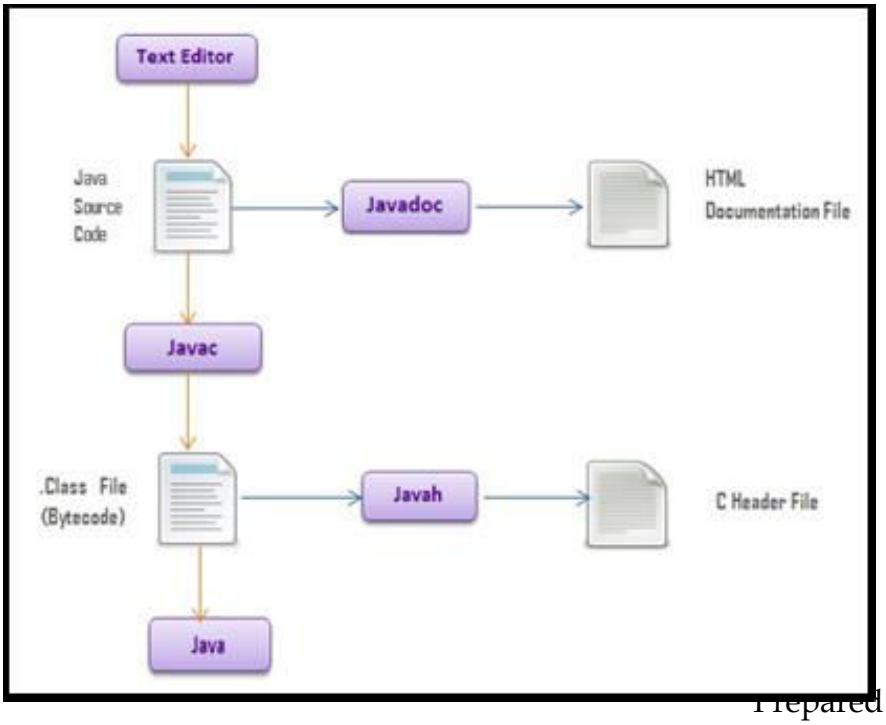
- It produces header files for use with native methods

javap

- Java disassembler, which enables us to convert byte code file into a program descriptions.

javadoc

- It creates HTML format documentation from Java Source code file. It extracts documentation comment and generate HTML file.
- The way these tools are applied to build and run Java application program is illustrated in below diagram.



JAVA program Structure:

A Java program may contains more than one class from which only one class has a main() method.

Classes may contains data and methods. The structure of Java Program is as follow

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, we use comments. 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
```

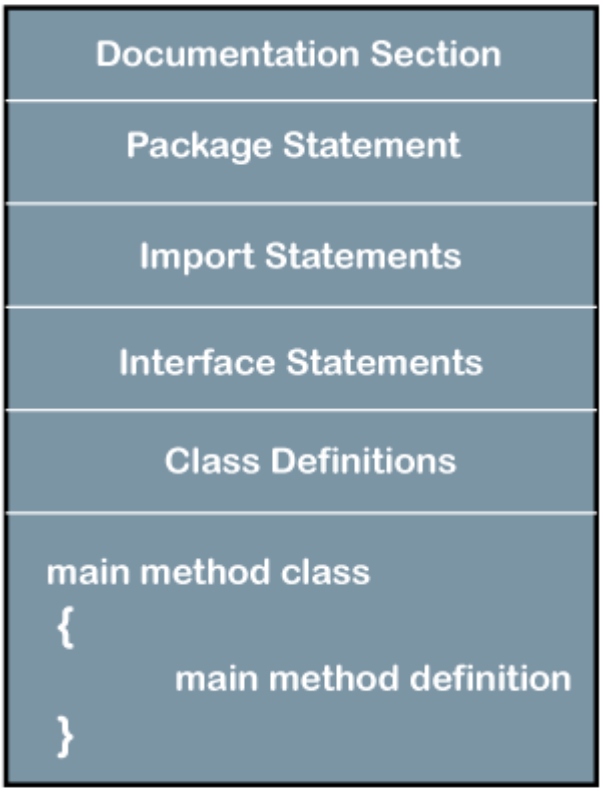
- Multi-line Comment: It starts with a /* and ends with */. We write between these two symbols. For example:

```
/*It is an example of
multiline comment*/
```

Package Declaration

The package declaration is optional. It is placed just after the documentation section. In this section, we declare the package name in which the class is placed. Note that there can be only one package statement in a Java program. It must be defined before any class and interface declaration. It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes package belongs to a single parent directory. We use the keyword package to declare the package name. For example:

```
package student;
```



Structure of Java Program

Import Statements

The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class. The import statement represents the class stored in the other package. We use the **import** keyword to import the class. It is written before the class declaration and after the package statement. We use the import statement in two ways, either import a specific class or import all classes of a particular package. In a Java program, we can use multiple import statements.

For example:

1. import java.util.Scanner; //it imports the Scanner class only
2. import java.util.*; //it imports all the class of the java.util package

Interface Section

It is an optional section. We can create an interface in this section if required. We use the interface keyword to create an interface. An interface is a slightly different from the class. It contains only constants and method declarations. Another difference is that it cannot be instantiated. We can use interface in classes by using the implements keyword.

For example:

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

Class Definition

In this section, we define the class. It is vital part of a Java program. Without the class, we cannot create any Java program. A Java program may conation more than one class definition. We use the class keyword to define the class. The class is a blueprint of a Java program. It contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method.

For example:

```
class Student //class definition
{
        //coding
}
```

Class Variables and Constants

In this section, we define variables and constants that are to be used later in the program. In a Java program, the variables and constants are defined just after the class definition. The variables and constants store values of the parameters. It is used during the execution of the program. We can also decide and define the scope of variables by using the modifiers. It defines the life of the variables. For example:

```
class Student //class definition
{
        String sname;  //variable
        int id;
        double percentage;
}
```

Main Method Class definition:

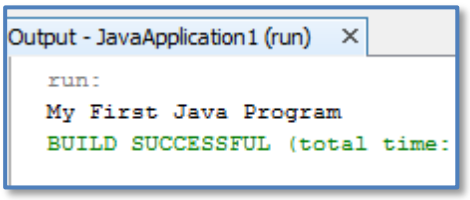
In this section, we define the main() method. It is essential for all Java programs. Because the execution of all Java programs starts from the main() method. In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

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

Compile and Execute:

Let's start Java programming by compiling and executing a simple program.

```
// This is my first java sample Program...
public class Example
{
    public static void main(String args[])
    {
        System.out.println("My First Java Program");
    }
}
```



- The first thing you learn about Java is that the name you give to the file is very important. The name of file should be the class name in which you declared main() method. In the above program the file name should be Example.java
- To compile the above program, execute the compiler, javac specifying the name of the source file on the command line. For e.g.

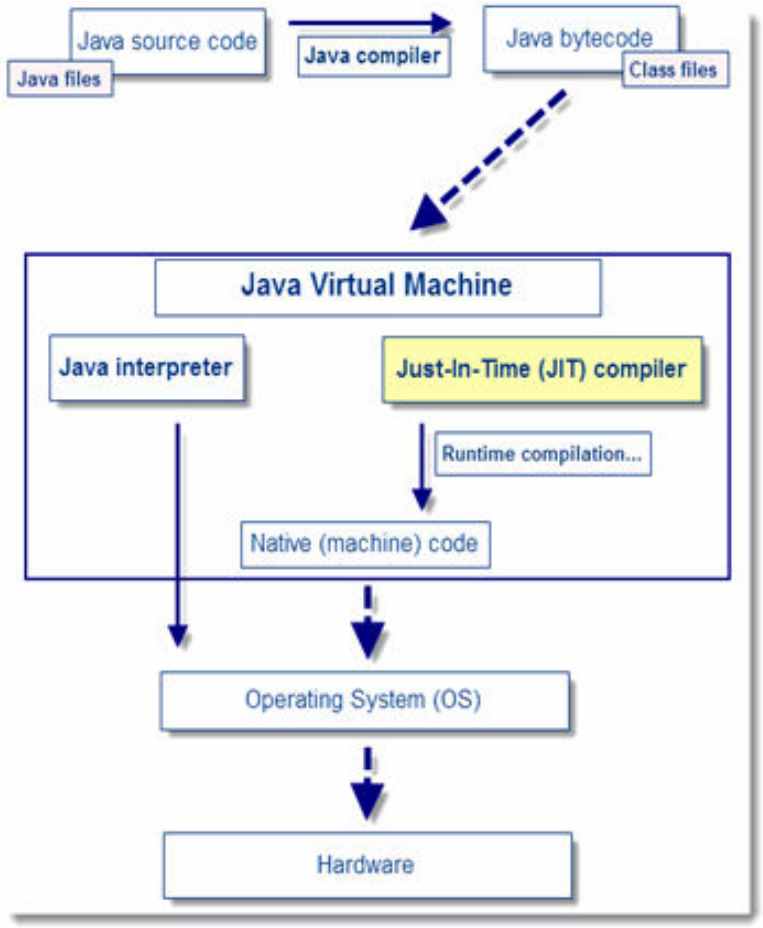
  javac Example.java

- The javac compiler creates a file called Example.class that contains the byte code version of the program. Therefore the output of javac is not code that can be directly executed.
- When Java source code is compiled, file with .class extension will be created for each individual class.
- To actually run the program, we have to use the Java application launcher, called java. Simple we have to pass the class name Example as command line argument to interpreter. For e.g.

  java Example

- The line uses the keyword **class** to declare that a new class is being defined. **Example** is an identifier that is the name of the class to be defined. The entire class definition will be between the braces {}.
- The next line in the program is **public static void main(String args[])**
- This line begins with main() method. All Java applications begin execution by calling main().
- The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members. When specified then, that member may be assessed by code outside the class in which it is declared.
- The keyword **static** allows main() to be called without having to instantiate a particular instance of the class. This is necessary since main() is called by the Java Virtual Machine before any objects are created.
- The keyword **void** simply tells the compiler that main() doesn't returns any value.

- The main() takes array of String as argument, which receives any command line argument during run time.
- The statement inside the main() method is **System.out.println("My First Java Program");**
- Where **System** is a predefined class that provides an access to the system and **out** is the output stream that is connected to the console.
- The **println()** is a built-in method which displays the string on the console that is passed to it.
- The below diagram represents an entire process of a program execution.



Differences between OOP and POP:

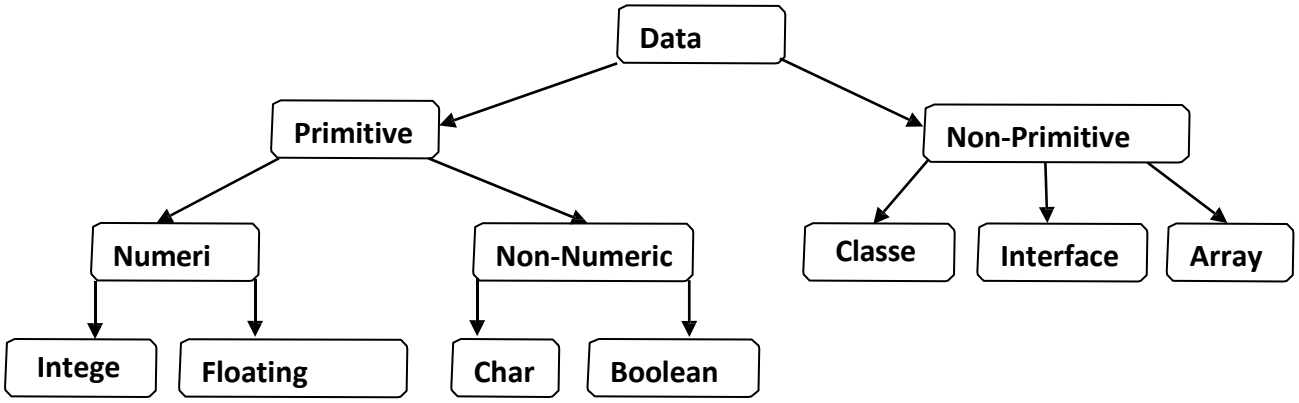
Basic Definition	OOP is object-oriented.	POP is structure or procedure-oriented.
Program Division	The program is divided into objects.	The program is divided into functions.
Approach	Bottom-Up approach	Top-down approach
Data Control	Data in each object is controlled on its own.	Every function has different data, so there's no control over it.
Entity Linkage	Object functions are linked through message passing.	Parts of a program are linked through parameter passing.

Expansion	Adding new data and functions is easy.	Expanding data and function is not easy.
Inheritance	Inheritance is supported in three modes: public, private & protected.	Inheritance is not supported.
Access control	Access control is done with access modifiers.	No access modifiers supported.
Data Hiding	Data can be hidden using Encapsulation.	No data hiding. Data is accessible globally.
Overloading or Polymorphism	Overloading functions, constructors, and operators are done.	Overloading is not possible.
Friend function	Classes or functions can be linked using the keyword "friend, only in C++.	No friend function.
Virtual classes or functions	The virtual function appears during inheritance.	No virtual classes or functions.
Code Reusability	The existing code can be reused.	No code reusability.
Problem Solving	Used for solving big problems.	Not suitable for solving big problems.
Example	C++, JAVA, VB.NET, C#.NET.	C, VB, FORTRAN, Pascal

Data types in JAVA:

- Data types specify the size and type of values that can be stored. Java language is rich in its data type

.



Java defines eight simple types of data which can be grouped as follows:

Integer : This group includes byte, short , int and long , which are for whole valued signed numbers

Name	Width (in bits)	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Floating-point Numbers: This group includes float and double, which represent numbers with fractional precision. Float is a single precision type while double is double precision type. Floating-points numbers are also known as real numbers. They are used when evaluating expressions that require fractional precision(For e.g root, sine, cosine etc)

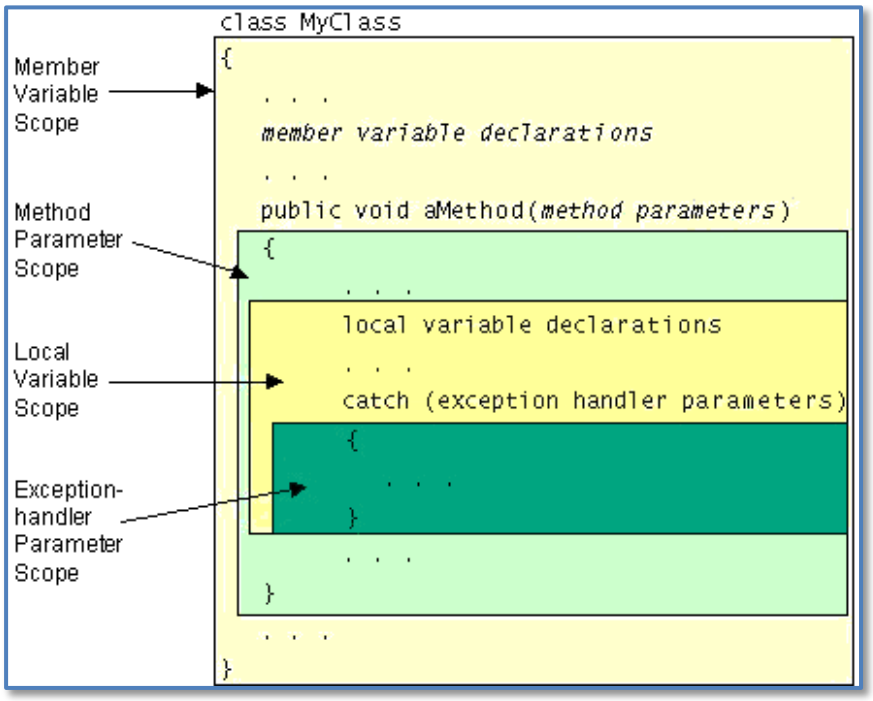
Name	Width (in bits)	Range
double	64	4.9e-324 to 1.8e+308
float	32	1.4e-045 to 3.4e+038

Character : This group includes char, which represents symbols in a character set, like letters and numbers. Java uses char data type to store character value. Java uses Unicode to represent characters (in C/C++ its ASCII). Unicode defines a fully International character set that can represent all the characters found in all human languages. It requires 16 bits and its range is 0 to 65,536.

Boolean : This group includes boolean**,** which is a special type for representing logical values. It can have two possible values true or false. This is the type returned by all relational operators and also required by the conditional expression that governs the control statements.

SCOPE OF VARIABLE...

- A variable's scope is the region of a program within which the variable can be referred to by its simple name.
- Scope also determines when the system creates and destroys memory for the variable. The location of variable declaration within your program establishes its scope and places it into one of these categories:
    - Member Variable
    - Local Variable
    - Method Parameter
    - Exception Handler Parameter
- A member variable is a member of a class or an object. It is declared within the class but outside any method or constructor. Its scope is an entire declaration of the class.
- A local variable can be declared within a block of code. Its scope extends from its declaration to the end of code block in which it is declared.
- Parameters are formal arguments to method or constructor and are used to pass values into method or constructor. Its scope is an entire method or constructor for which it is a parameter.
- Exception handler parameters are similar to parameters but they are arguments to an exception handler rather than method or constructor. Its scope is the code block between curly braces follow a catch statement.



- Each class variable, instance variable or array component is initialized with default value when it is created. Default value for primitive data types are as follows:

Data type	Default value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
boolean	false
char	null

JAVA Tokens:

A Java program is a collection of tokens, comments and white spaces. Java language includes the following tokens.

- **Reserve Keywords**
- **Identifiers**
- **Literals**
- **Operators**
- **Seperators**

Keywords:

- Keywords are an essential part of a language definition. They implement specific feature of the language. Java has 60 reserved words as keywords.

- These keywords, combined with the syntax of the operators and separators, form the definition of the Java language. Since keywords have a specific meaning, we cannot use them as a name of variable, class, method etc. Java's keywords are shown in the following table.

Abstract	**threadsafe***	public	long	**generic***	double
Case	Try	super	**outer***	**inner***	finally
const*	Boolean	throw	**rest***	native	**goto***
Else	**cast***	**var***	switch	package	instanceof
Float	Continue	break	throws	return	new
If	Extends	catch	void	synchronized	private
Int	For	default	byte	transient	short
*null***	Implements	*false***	char	volatile	this
protected	Interface	**future***	do	**byvalue***	while
Static	**operator***	import	final	class	*true***
* represents Keywords reserved for future use ** represents keywords are values					

IDENTIFIERS

- Identifiers are used for naming classes, methods, variables, objects, packages and interfaces in a program. An identifier may be any descriptive sequence of uppercase and lowercase letters, number, or underscore (_) and dollar-sign character ($).
- They must not begin with a number.
- They can be of any length but they are case-sensitive.
- An identifier should be meaningful, short enough to be quickly and easily typed and long enough to be descriptive and easily read.

LITERALS

- A constant value in Java is created by using a literal representation of it. For eg. **100 51.92 'A' "abc"**
- The first literal (left to right) specifies an integer, the next is a floating-point value, the third is character constant and the last is a string.
- A literal can be used anywhere a value of its typed is allowed.

SEPARATORS

Separators are symbols used to indicate where group of code are divided and arranged. They basically define the shape and function of our code. The separators are shown in the following table:

Symbo	Purpose	Name
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array
;	Semicolon	Terminates statements.

,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from sub packages and classes. Also used to separate a variable or method from a reference variable.

COMMENTS

- A comment is a note to yourself that you put in your source code. All comments are ignored by compiler that means they are not translated.
- Comments provide a better readability of program. They indicate purpose of your source code so that you can remember later.
- There are three forms of comment in Java.

Single Line Comment

This begins with the two character sequence // and include the rest of characters on the same line. For example

```
int count=0; // Count is used as counter
```

Multi Line Comment

It begins with the two character sequence /* and ends with */. This form of comment may also extend over several lines.

```
/* This is multi line comment extend To multiple lines */
```

Multi Line Comment

It begins with a three character sequence /** and ends with two character sequene */.

```
/** This is Java Documentation Comment */
```

The advantage of documentation comment is that javadoc tool can extracts this comment from source file and automatically generates HTML documentation. In Java, comment goes anywhere except in the middle of an Identifier.

WHITESPACES

- Java is a free form language that means we don't have to follow any special indentation rules. For example, we can write our program in one line or in other strange way that we felt typing it.
- There must be at least one whiter space character between each token.
- In Java, white space can be a space, tab or new line.

OPERATORS

- Operators are the symbols that used to perform certain mathematical and logical manipulation on data.
- Java operators can be classified as follows:

1. **Arithmetic Operators**

- Arithmetic operators are used in mathematical expressions. Operands of the arithmetic operators must be of a numeric type. We can use char type since it is subset of int.

Operator	Use	Description
+	Op1 + op2	Adds Op1 and Op2

-	Op1 - op2	Subtracts Op2 from Op1
*	Op1 * op2	Multiplies Op1 by Op2
/	Op1 / op2	Divides Op1 by Op2
%	Op1 % op2	Computes the remainder of dividing Op1 by Op2

2. **Relational Operators**

- It is used to compare two quantities depending on their values. It returns true if condition is true else false.
- The relational operators are most frequently used in the expressions that control the if statement and the various loop statements.

Operator	Description	Example
= =	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the 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 the 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 the 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 the 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 the condition becomes true.	(A <= B) is true.

3. **Boolean Logical Operators**

- The Boolean logical operators operate only on boolean operands. All of the binary logical operators combine two boolean values to form a resulting boolean value.

O	Result
&	Logical AND
|	Logical OR
^	Logical XOR
||	Short-circuit OR
&&	Short-circuit
!	Logical unary
&=	AND
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to

- The logical boolean operators, &,| and ^ operate on boolean values in the same way that they operate on the bits of an integer. The logical ! operator inverts the boolean state: !true==false and !false ==true. The following table shows the effect of each logical operation:

A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

4. **Assignment Operators**

- The assignment operator is the single equal sign (=). The general form of assignment operator is :

  var = expression;

- Here, the type of variable must be compatible with the type of expression. Java language also provides several shorthand assignments that allow performing arithmetic, shift or bitwise operations.

Ope	Result	Equivalent to
+=	op1 += op2	op1=op1 + op2
-=	op1-= op2	op1=op1 - op2
***=**	op1 *= op2	op1=op1 * op2
/=	op1 /= op2	op1=op1 / op2
%=	op1 %= op2	op1=op1 % op2
&=	op1 &= op2	op1=op1 & op2
|=	op1 |= op2	op1=op1 | op2
^=	op1 ^= op2	op1=op1 ^ op2
<<=	op1 <<= op2	op1=op1 << op2
>>=	op1 >>= op2	op1=op1 >> op2
>>>	op1 >>>= op2	op1=op1 >>> op2

5. **Increment/Decrement Operators**

- Increment and decrement operators are summarized in the following table

O	Use	Descriptions
++	op++	Increments op by 1, evaluates to the value of op before it was incremented.
++	++op	Increments op by 1, evaluates to the value of op after it was incremented.
--	op--	Decrements op by 1, evaluates to the value of op before it was decremented.
--	--op	Decrements op by 1, evaluates to the value of op after it was decremented.

- Postfix version (op++ /op--) evaluates the value of operand before increment/decrement operation while prefix version after.

6. **Bitwise Operators**

- Java defines several bitwise operators that can be applied to integer types, long, short, char and byte. These operators operate on individual bits of their operands. Bitwise operators are :

Operato	Name	Example	Result /Process
~	Bitwise Unary NOT	int a=42; int c; c=~a;	42=00101010 will convert to 11010101
&	Bitwise AND	**int** a=42; **int** b=15; System.*out*.println(a&b);	42=00101010 15=00001111 10=00001010
|	Bitwise OR	**int** a=42; **int** b=15; System.*out*.println(a|b);	42=00101010 15=00001111 47=00101111
^	Bitwise XOR	**int** a=42; **int** b=15; System.*out*.println(a^b);	42=00101010 15=00001111 37=00100101
>>	Shift Right	**int** a=19; System.*out*.println(a>>2);	19=0001 1 4 0 000100
>>>	Shift Right zero fill	**int** a=-1; System.*out*.println(a>>>24);	Result =255 Here 24 bits (>>>3)
<<	Left Shift	**int** a=25; System.*out*.println(a<<2);	2 0 011001 100=1100100

CONDITIONAL OPERATOR / TERNARY OPERATOR

- Java includes a special ternary operator that can replace certain types of if-then-else statements.

Syntax is as follows:

expression1 ? expression2 : expression3;

- If expression1 is true then expression 2 is evaluated else expression 3. Here expression1 is any expression that evaluates to a Boolean value.
- Both expression2 and expression3 are required to return same type, which can't be void.

7. **Special Operators**

- Java supports some special operators like instanceof and dot(.) operator (known as Member selection operator).
- The dot (.) operator is used to call methods and to access data member using class object.
- It is also used to extract class and sub packages from package. For eg. Import java.util.Date
- The instanceof operator is an object reference operator and returns true if an object on left side is an instance of the class given on the right hand side.
- Syntax is : **objref instanceof type**

class A

{

```
}
class B
{
}
public class InstanceOf
{
        public static void main(String a[])
        {
                A obja=new A();
                B objb=new B();
                if(obja instanceOf A)
                        System.out.println("obja is an object of class A");
                else
                        System.out.println("obja is an object of class B");
        }
}
```

OPERATOR PRECEDENCE AND ITS ASSOCIATIVITY

- Each operator has a precedence associated with it, which is used to determine how an expression with multiple operators is evaluated.
- The operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from left to right or right to left depending on the level.

Op	Name	Associa	R
.	Member Selection	Left-to-Right	1
()	Function Call		
[]	Array element reference		
-	Unary Minus	Right-to-Left	2
++	Increment		
--	Decrement		
!	Logical Negation		
~	One's Complement		
(ty	Casting		
*	Multiplication	Left-to-Right	3
/	Division		
%	Modulus		
+	Addition	Left-to-Right	4
-	Subtraction		
<<	Left Shift	Left-to-Right	5
>>	Right Shift		
>>	Right Shift with zero fill		
<	Less than	Left-to-Right	6
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
inst	Type comparison		
==	Equality	Left-to-	7

!=	Inequality	Right	
&	Bitwise AND	Left-to-	8
^	Bitwise XOR	Left-to-	9
|	Bitwise OR	Left-to-	10
&&	Logical AND	Left-to-	11
||	Logical OR	Left-to-	12
?:	Conditional Operator	Right-	13
=	Assignment Operator	Right-to-Left	14
op	Shorthand Assignment		

- For example if(x==10+15 && y<10) then, precedence rules say that addition operator has higher priority than logical AND and relational < and ==. Therefore the addition of 10 and 15 is evaluated first. i.e. if(x==25 && y<10)

- The next step is to determine whether x is equal to 25 and y is less than 10. Suppose value of x =20 and y=5 then,

  x==25 is false y<10 is true

- Note that since operator < has higher priority compared to ==. Therefore y<10 is tested first and then x== 25. Finally we get if(False && True)

IF STATEMENT

The if statement enables your program to selectively execute other statement based on some condition result.

Syntax:

```
if(condition)
{
    statement(s);
}
```

Example:

```
class ifstmt
{
    public static void main ( String a[])
    {
        int no=12;
        if (no%2==0)
        {
            System.out.println ("no is even ");
        }
        if (no%2! =0)
        {
            System.out.println ("no is odd ");
        }
    }
}
```

IF-ELSE STATEMENT

If we want to perform a different set of statements if the expression is false then we can use else statement in if statement.

Syntax:

```
if(condition)
{
        true block statement(s);
}
else
{
        false block statement(s);
}
```

- If the test expression is true then the true block statement(s) execute otherwise false –block statement(s). In no case both statements will executed.
- The expression involves the relational operators. It is also possible to control the if by using boolean variable. Remember only one statement appears directly after the if statement . If more statements exist they must be in block code.

Example:

```
class ifelsestmt
{
        public static void main ( String a[])
        {
                int no=12;
                if (no%2==0)
                {
                        System.out.println ("no is even ");
                }
                else
                {
                        System.out.println ("no is odd ");
                }
        }
}
```

NETSTED IF ELSE STATEMENT:

A nested if else is the if statement that is target of another if or else. One thing to remember is that an else statement always refers to the nearest if statement.

```
if(test condition1)
{
        if(test condition2)
        {
                //stmt 1
        }
        else
        {
                //stmt 2
        }
}
else
{
        //stmt 3
}
```

If the test condition1 is true then test condition 2 is evaluated and if it is true then statment1 will be executed otherwise statement2. But if test condition1 is false then only statement 3 will be executed.

```
class nested
{
        public static void main( String a[])
        {
                        int num=12;
                        if (num = = 0 )
                        {
                                System.out.println ("Number is Zero");
                        }
                        else
                        {
                                if (num < 0)
                                {
                                        System.out.println ("Number is Negative");
                                }
                                else
                                {
                                        System.out.println ("Number is Positive");
                                }
                        }
        }
}
```

ELSE-IF LADDER

- In this control logic conditions are evaluated from top to down. As soon as true condition found statements associates with it are executed and control is transferred to the next statement.
- When all conditions become false then the final else containing default statement will be executed.

Syntax:

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

Example:

```
public class elseif
{
        public static void main ( String a[])
```

```
{
        int a=12,b=13 ,c=14,max ;

        if (a>b && a>c)
        {
                max=a;
        }
        else if(b>a && b>c)
        {
                max=b;
        }
        else
        {
                max=c;
        }
        System.out.println ("Maximum number is:" + max);
}

}
```

SWITCH CASE STATEMENT:

- Switch statement is Java's multi way branching statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.
- It is a better alternative than a large series of if-else-if statement.

```
switch(expression)
{
        case value1:
                statement sequence;
                break;
        case value2:
                statement sequence;
                 break;
        .
        .
        case valuen:
                statement sequence;
                 break;
        default:
                default statement
}
```

- The expression must be of type byte, short, int or char. Each of the value specified in the case statement must be of a type compatible with an expression.
- Each case value must be unique literal. Duplicate case values are not allowed.
- The value of an expression is compared with each of the literal values in case statement. If a match is found then the code sequence following that case is executed. If no match found, then default statement will execute. Default is an optional.
- Break statement is used inside the switch to terminate a statement sequence. It is used to jumping out of the switch.
- We can also use a switch as a part of the statement sequence of an outer switch. This is called nesting of switch.

- Java iteration statements are for, while and do-while which we called loops.
- A loop repeatedly executes the same set of instructions until a termination condition is met.

The while statement

It is used to repeatedly execute a block of statements while a condition remains true.

```
while(condtional expression)
{
        statements block;
}
```

First the while statement evaluates conditional expression, which must returns a boolean value. If expression returns true, then statements associated with while are executed. While loop continues testing expression and execute its block until expression returns false.

Example:

```
class while
{
        public static void main  ( String a [])
        {
                int digit = 0;
                while(digit<=9)
                {
                        System.out.println (digit);
                        digit++ ;
                }
        }
}
```

The do-while statement

- While loop will test the condition before looping, making it possible that the body of the loop will never execute if condition is false for first time when tested.
- While do..while loop runs the body of the loop at least once before testing the condition

```
do
{
        statements block;
} while(condtional expression);
```

The for statement

The for loop is often used where you want to repeat a section of a code for a fixed amount of times.

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

- Initialization: is a statement that initializes the loop. It is executed once at the beginning of the loop
- Condition: is an expression that determines when to terminate loop. It is evaluated at top in each iteration.
- Increment/decrement: it get invoked for each iteration. It can be empty also.

Example:

```
for (I=1; I<=10; I++)
```

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

Continue Statement:

- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
- The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
- We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```
continue;
```

Example:

```
for(I=1; I<=10; I++)
{
   if(I==5)
         continue;
   System.out.println (I);
}
```

Break Statement

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```
break;
```

```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample
 {
      public static void main(String[] args)
      {
         for(int i=1;i<=10;i++)
        {
            if(i==5)
           {
              //breaking the loop
              break;
           }
            System.out.println(i);
       }
    }
}
```

return Statement:

- The return statement is used for returning a value when the execution of the block is completed.
- The return statement inside a loop will cause the loop to break and further statements will be ignored by the compiler.
- In Java, every method is declared with a return type such as int, float, double, string, etc.
- These return types required a return statement at the end of the method. A return keyword is used for returning the resulted value.
- The return type of the method and type of data returned at the end of the method should be of the same type. For example, if a method is declared with the float return type, the value returned should be of float type only.

Syntax:

```
return ret_val;
```

Eg.

```
public class first
{
   /* Method with an integer return type and no arguments */
   public int max()
   {
      int x = 3;
      int y = 8;
      System.out.println("x = " + x + "\ny = " + y);
      if(x>y)
         return x;
      else
         return y;
   }
public static void main(String ar[])
   {
      first obj = new first();
      int result = obj.max();
      System.out.println("The greater number among x and y is: " + result);
   }
}
```

Arrays

An array is a group of like-typed variables that are referred to by a common name. Array of any type can be created and may have one or more dimensions.

A specific element in array is accessed by its index. There are three steps to create an Array

- Declaration
- Construction
- Initialization

Declaration includes the name of an array and type of element of an array.

One Dimensional Array

A one dimensional array is essentially a list of like typed variables.

To create an array first we must create an array variable of the desired type.

Syntax

```
type array_variable_name[];
or type [] array_variable_name;
```

e.g.

```
int array1[];   long [] arr;
int month_days[];
```

To allocate array to memory physically we can use syntax :

```
array_var=new type[size];
e.g. month_days=new int[30];
```

We can initialize one dimensional array in either of following 3 ways:

int arr[];
arr=new int[5];
for(int j=0;j<5;j++)
arr[j]=j+5;
	int arr=new int[5];
	arr[0]=5;
	arr[1]=10;
	arr[2]=15;
	arr[3]=20;
	arr[4]=25;
		int arr[]={5,10,15,20,25};

Multi Dimensional Array

In Java, a multidimensional array is an array of array. To declare a multidimensional array variable, specify each additional index using another set of square bracket.

type array_variable_name[][];
or // two dim array type [][] array_variable_name;

e.g. int arr[][]={{1,2},{3,4},{5,6}};

Left index specifies row and right index specifies columns.

Non Regular Array

In Java, it is compulsory to allocate memory for the first dimension in multi dimension arrays. We can allocate memory to remaining dimension afterwards.
In non rectangular arrays, number of rows is fixed, but number of columns for each row is different.

```
e.g. int arr[][]=new int[5][]; //valid statement
int arr[][]=new int[][5];// invalid statement
```

We can also resize arrays after construction of an array.

```
e.g. int arr[][]=new int[5][]; // 5 rows
arr[][]=new int[15][];// 15 rows..
```

In Java, all arrays have one instance variable length, which holds the number of elements or size of an array. For eg.

```
public class ArrLength
{
        public static void main(String[] args)
        {
                int arr1[]={1,2,3,4,5,6,7,8,9,10};
                int arr2[]={1,2,3,4,5};
```

```
                int arr3[]={1,2,3};
                System.out.println(arr1.length);
                System.out.println(arr2.length);
                 System.out.println(arr3.length);
        }
}
```

Command Line Argument Array:

- On many systems it is possible to pass arguments from the command line (these are known as command-line arguments) to an application by including a parameter of type String[] (i.e., an array of Strings) in the parameter list of main.
- By convention, this parameter is named args. When an application is executed using the java command, Java passes the command-line arguments that appear after the class name in the java command to the application's main method as Strings in the array args.
- The number of arguments passed in from the command line is obtained by accessing the array's length attribute.
- For example, the command "java MyClass a b" passes two command-line arguments to application MyClass. Note that command-line arguments are separated by white space, not commas. When this command executes, MyClass's main method receives the two-element array args (i.e., args.length is 2) in which args[0] contains the String "a" and args[1] contains the String "b".
- Common uses of command-line arguments include passing options and file names to applications.

```
class A
{
        public static void main(String args[])
        {
                for(int i=0;i<args.length;i++)
                        System.out.println(args[i]);
        }
}
```

compile by > javac A.java
run by > java A hello how are you
Output:

```
hello
how
are
you
```

Type Casting in Java

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.

Type casting
Convert a value from one data type to another data type is known as **type casting**.

Types of Type Casting
There are two types of type casting:

- Widening Type Casting

- Narrowing Type Casting

Widening Type Casting
Converting a lower data type into a higher one is called widening type casting. It is also known as implicit conversion or casting down. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

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

        //automatically converts the integer type into long type
        long y = x;
        //automatically converts the long type into float type
        float z = y;
        System.out.println("Before conversion, int value "+x);
        System.out.println("After conversion, long value "+y);
        System.out.println("After conversion, float value "+z);
}
}
```

Narrowing Type Casting
Converting a higher data type into a lower one is called narrowing type casting. It is also known as explicit conversion or casting up. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

```
public class ntypecast
{
public static void main(String args[ ])
{
        double d = 166.66;
        //converting double data type into long data type
        long l = (long)d;
        //converting long data type into int data type
        int i = (int)l;
        System.out.println("Before conversion: "+d);
        //fractional part lost
        System.out.println("After conversion into long type: "+l);
        //fractional part lost
        System.out.println("After conversion into int type: "+i);
}
}
```

OOPs (Object-Oriented Programming System)
Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism

- Abstraction
- Encapsulation
- Overloading
- Message Communication

Object

Any entity that has state and behaviour is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

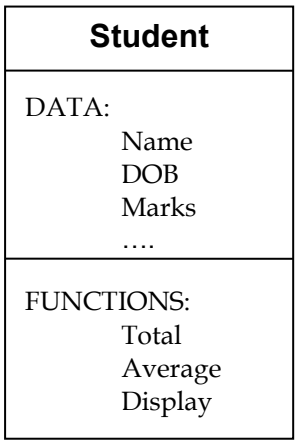
An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

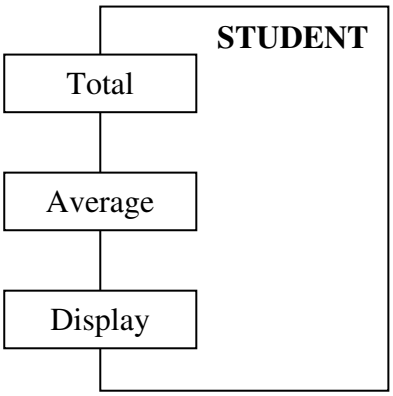
Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

- Objects are the basic run-time entities of object-oriented programming.
- Objects are identified by its unique name.
- An object represents a particular instance of a class.
- There can be more than one instance of a class.
- Each instance of a class can hold its own relevant data.
- An Object is a collection of data members and associated member functions also known as methods.
- When a program is executed, the objects interact by sending messages to one another.

Eg: If 'customer' and 'account' are two objects in a program, then the customer object may send message to account object requesting for a bank balance.

- Each object contains data and code to manipulate data.
- Objects can interact without having to know details of each other's data or code.
- It is sufficient to know the type of massage accepted and the type of response returned by the objects.
- Objects take up space in the memory and have an associated address.
- The objects can be represented by objects as follows:





Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

- We have just mentioned that objects contain data and function or code to manipulate that data.
- The entire set of data and code of an object can be made a user-defined data type with the help of a class.
- In fact objects are variables of type class.
- Once a class has been defined, we can create any number of objects associated with that class.
- A class is thus a collection of objects of similar type

Eg: For example, mango, apple and orange are members of class fruit.

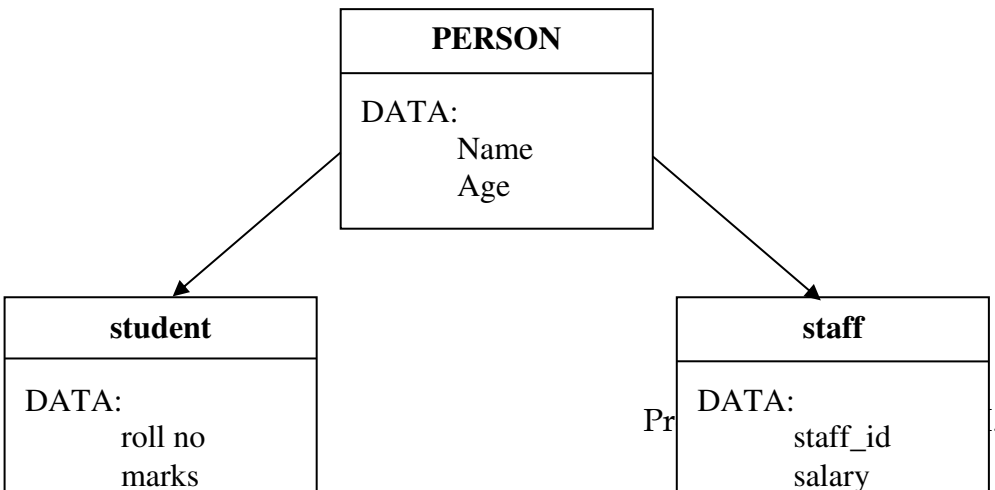
- Classes are user-defined data types and behave like the built in data types of a programming language.
- If fruit has been defined as a class, then the statement
    - fruit mango;
    - It will create an object mango belonging to the class fruit.
- So a class gives a structure of what an object of its type will have.
- No memory is allocated when a class is created. Memory is allocated only when an object is created, i.e., when an instance of a class is created.

Inheritance

When one object acquires all the properties and behaviours of a parent object, it is known as inheritance. It provides code reusability. With inheritance, reusability is a major advantage. You can reuse the fields and methods of the existing class. In Java, there are various types of inheritances: single, multiple, multilevel, hierarchical, and hybrid. It is used to achieve runtime polymorphism.

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- Inheritance is the process of forming a new class from an existing class or base class.
- The concept of inheritance provides the idea of reusability.
- This means that we can add additional features to an existing class without modifying it.
- This is possible by deriving a new class from the existing old one.
- The new class will have the combined features of both the classes.
- The old class is known as base class or parent class or super class.
- The new class that is formed is called derived class or child class or sub class.
- The sub class defines only those features that are unique to it.
- Inheritance helps in reducing the overall code size of the program because it can reuse the code which is once declared.

Eg: A common class person is derived into Student and Staff class. So these two classes use the data of Person class



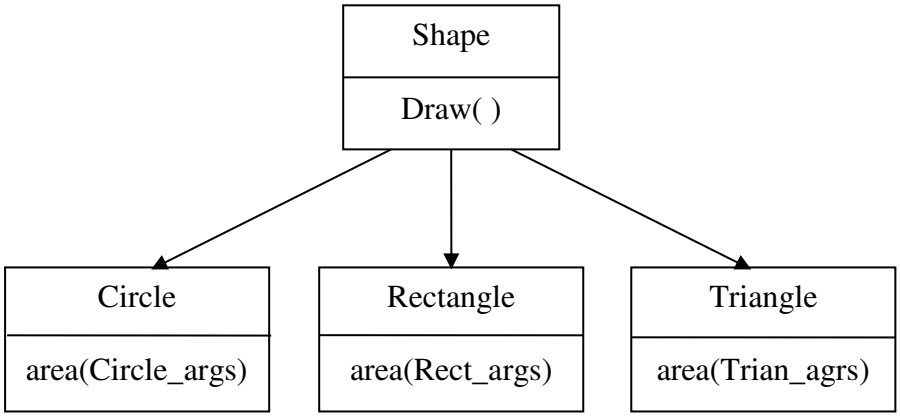
Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism.

- Polymorphism means the ability to take more than one form.
- An operation may exhibit different behaviors in different instances.
- The behavior depends upon the types of data used in the operation.

Eg: Consider the operation of addition. For 2 numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

- The process of making an operator to exhibit different behaviors in different instances is known as **operator overloading.**
- A single function name can be used to handle different number and different types of arguments.
- This is something similar to a particular word having several different meanings depending on the context.
- Using a single function name to perform different types is known as **function overloading.**
- So polymorphism plays an important role in allowing objects having different internal structures to share the same external interface.
- This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ.
- The following figure shows the concept of polymorphism.



- **Here the Circle, Rectangle and Triangle object contains the area ( ) function but the arguments passed are different for all the objects.**

Encapsulation

- Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.
- Encapsulation is accomplished when each object maintains a private state, inside a class. Other objects cannot access this state directly, instead, they can only invoke a list of public functions. The object manages its own state via these functions and no other class can alter it unless explicitly allowed. In order to communicate with the object, you will need to utilize the methods provided.

  - The wrapping up of data and functions into a single unit is called as encapsulation.
  - The data is not accessible to the outside world and those functions which are wrapped in the class can access it.
  - If you want to reach the data item in an object, you call a member function in the object. It will read the data item and return the value to you.
  - These functions provide the interface between the object's data and the program.
  - The prevention of the data from direct access by the program is called data hiding or information hiding.

Abstraction

- Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.
- In Java, we use abstract class and interface to achieve abstraction.
- Abstraction is an extension of encapsulation. It is the process of selecting data from a larger pool to show only the relevant details to the object.
- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Classes use the concept of abstraction and are defined as a list of abstract attributes and functions to operate on these attributes.
- They encapsulate all the essential properties of the objects that are to be created.
- Since the classes use the concept of data abstraction, they are known as Abstract Data Types.

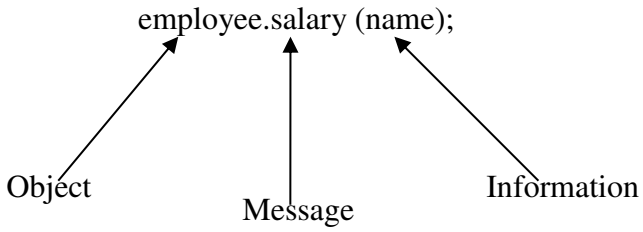
Eg: The capsule of medicine is the Eg. of Data Encapsulation and Abstraction. The powdered medicine is hidden inside capsule and it is covered with a wrapper, it is Encapsulation. Capsule hides the essential powdered medicine inside it and we don't know what is hidden inside it, we can only see the covered capsule, it is abstraction.

Message Communication:

Object-oriented programming as a programming paradigm is based on objects. Objects are a representation of real-world and objects communicate with each other via messages. When two or more objects communicate with each other that means that those objects are sending and receiving messages.

- An object-oriented program consists of a set of objects that communicate with each other.
- The process of programming in an object-oriented language, therefore, involves the following basic steps:
    - Creating classes that define objects and their behavior.
    - Creating objects from class definitions
    - Establishing communication among objects.
- Object communicate with one another by sending and receiving information much the same way as people pass messages to one another.
- A message for an object is a request for execution of a procedure, and therefore will invoke function in the receiving object that generates the desired result.
- Message passing involves specifying the name of the object, the name of the function (message) and the information to be sent.

Eg: The object is employee, message is passed to the salary ( ) and sent the information name to it. So the salary ( ) will sent the salary of the employee as the name specified.



CLASS:

In object-oriented programming, a class is a basic building block. It can be defined as template that describes the data and behavior associated with the class instantiation. Instantiating is a class is to create an object (variable) of that class that can be used to access the member variables and methods of the class.

A class can also be called a logical template to create the objects that share common properties and methods.

For example, an Employee class may contain all the employee details in the form of variables and methods. If the class is instantiated i.e. if an object of the class is created (say e1), we can access all the methods or properties of the class.

Java provides a reserved keyword **class** to define a class. The keyword must be followed by the class name. Inside the class, we declare methods and variables.

```
class classname
{
    Datatype  variablel;
    Datatype variable2;

return_type methodname(parameter-l ist)
{
```

```
    //body of method
}
return_type     methodname2 (parameter-list)
{
    //body of method
}

return_type     methodnameN (parameter-l ist)
{
    //body of method
}
}
```

- A class is declared by use of the class keyword.
- The data, or variables, defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data of one object is separate and unique from the data of another.
- The actual code contained within methods**.**
- The methods and variables defined within a class are called members of the class.
- The name of the method is specif ied by method_name. This can be any legal identifier other than those already used by other member within the current scope. The parameter-list is a sequence of type and identifier pairs separated by commas.
- Parameters are essentially variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

Eg.

```
class Box
{
        double width;
        double height;
        double depth;
}
```

As stated, a class defines a new type of data. In this case, the new data type is called Box. You will use this name to declare objects of type Box.
To actually create a Box object,

```
Box mybox = new Box(); // create a Box object called mybox
```

After this statement executes, mybox will be an instance of Box. Thus, it will have "physical" reality. Thus, every Box object will contain its own copies of the instance variables width, height, and depth. To access these variables, you will use the dot (.) operator. The dot operator links the name of the object with the name of an instance variable. For example, to assign the width variable of mybox the value 100, you would use the following statement:

```
mybox.width = 100;
```

This statement tells the compiler to assign the copy of width that is contained within the mybox object the value of 100. In general, you use the dot operator to access both the instance variables and the methods within an object.

class Box

```
{
        double width;
        double height;
        double depth;
}
// This class declares an object of type Box.
class BoxDemo
{
        public static void main(String args[])
        {
                Box mybox = new Box();
                double vol;
                // assign values to mybox's instance variables
                mybox.width = 10;
                mybox.height = 20;
                mybox.depth = 15;
                // compute volume of box
                vol = mybox.width * mybox.height * mybox.depth;
                System.out.println("Volume is " + vol);
        }
}
```

Declaring Objects As just explained, when you create a class, you are creating a new data type. You can use this type to declare objects of that type. However, obtaining objects of a class is a two-step process. First, you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object. Second, you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the new operator. The new operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.

```
Box mybox = new Box();
```

This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

The first line declares mybox as a reference to an object of type Box. After this line executes, mybox contains the value null, which indicates that it does not yet point to an actual object. Any attempt to use mybox at this point will result in a compile-time error. The next line allocates an actual object and assigns a reference to it to mybox. After the second line executes, you can use mybox as if it were a Box object. But in reality, mybox simply holds the memory address of the actual Box object.

Assigning Object Reference Variables Object reference variables act differently than you might expect when an assignment takes place. For example, what do you think the following fragment does?

```
Box b1 = new Box();
Box b2 = b1;
```

You might think that b2 is being assigned a reference to a copy of the object referred to by b1. That is, you might think that b1 and b2 refer to separate and distinct objects. However, this would be wrong. Instead, after this fragment executes, b1 and b2 will both refer to the same object. The assignment of b1 to b2 did not allocate any memory or copy any part of the original object. It simply makes b2 refer to the same object as does b1. Thus, any changes made to the object through b2 will affect the object to which b1 is referring, since they are the same object.

Introducing Methods

This is the general form of a method:

```
ret_type name(parameter-list)
{
        // body of method
}
```

Here, type specifies the type of data returned by the method. This can be any valid type, including class types that you create. If the method does not return a value, its return type must be void. The name of the method is specified by name. This can be any legal identifier other than those already used by other items within the current scope. The parameter-list is a sequence of type and identifier pairs separated by commas. Parameters are essentially variables that receive the value of the arguments passed to the method when it is called. If the method has no parameters, then the parameter list will be empty. Methods that have a return type other than void return a value to the calling routine using the following form of the return statement:

```
return value;
```

Here, value is the value returned. In the next few sections, you will see how to create various types of methods, including those that take parameters and those that return values.

```
class Box
{
        double width;
        double height;
        double depth;
        // display volume of a box
        void volume()
        {
                int ans = width * height * depth;
                System.out.print("Volume is ");
                System.out.println(ans);
        }
}
class BoxDemo
{
        public static void main(String args[])
        {
                 Box mybox1 = new Box();
                // assign values to mybox1's instance variables
                mybox1.width = 10; mybox1.height = 20; mybox1.depth = 15;
                // display volume of  box
                mybox1.volume();
        }
}
```

Returning a Value

While the implementation of volume( ) does move the computation of a box's volume inside the Box class where it belongs, it is not the best way to do it. For example, what if another part of your program wanted to know the volume of a box, but not display its value? A better way to implement volume( ) is to have it compute the volume of the box and return the result to the caller. The following example, an improved version of the preceding program, does just that:

```
class Box
{
        double width;
        double height;
        double depth;
        // display volume of a box
        double volume()
        {
                double ans = width * height * depth;
                return ans;
        }
}
class BoxDemo
{
        public static void main(String args[])
        {
                 Box mybox1 = new Box();
                // assign values to mybox1's instance variables
                mybox1.width = 10; mybox1.height = 20; mybox1.depth = 15;
                // display volume of  box
                double ans = mybox1.volume();
                System.out.print("Volume is ");
                System.out.println(ans);
        }
}
```

There are two important things to understand about returning values:
■ the type of data returned by a method must be compatible with the return type specified by the method. For example, if the return type of some method is boolean, you could not return an integer. ■ The variable receiving the value returned by a method (such as vol, in this case) must also be compatible with the return type specified for the method.

Adding a Method That Takes Parameters

While some methods don't need parameters, most do. Parameters allow a method to be generalized. That is, a parameterized method can operate on a variety of data and/or be used in a number of slightly different situations. To illustrate this point, let's use a very simple example. Here is a method that returns the square of the number 10:

```
        int square()
        {
                return 10 * 10;
        }
```

While this method does, indeed, return the value of 10 squared, its use is very limited. However, if you modify the method so that it takes a parameter, as shown next, then you can make square( ) much more useful.

```
int square(int i)
{
        return i * i;
}
```

Now, square( ) will return the square of whatever value it is called with. That is, square( ) is now a general-purpose method that can compute the square of any integer value, rather than just 10. Here is an example:

```
int x, y;
x = square(5); // x equals 25
x = square(9); // x equals 81
y = 2;
x = square(y); // x equals 4
```

```
class Box
{
        double width;
        double height;
        double depth;

        void setdata(double w, double h, double d)
        {
                width = w; height = h; depth = d;
        }
        double volume()
        {
                return width * height * depth;
        }
}
class BoxDemo
{
        public static void main(String args[])
        {
                 Box mybox1 = new Box();
                // assign values to mybox1's instance variables
                mybox1.setDim(10, 20, 15);
                // display volume of  box
                double ans = mybox1.volume();
                System.out.print("Volume is ");
                System.out.println(ans);
        }
}
```

The this Keyword

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the "this" keyword. "this" can be used inside any method to refer to the current object. That is, this is always a reference to the object on which the method was invoked. You can use this anywhere a reference to an object of the current class' type is permitted.

Here is given the 6 usage of java this keyword.

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.

- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

To better understand what this refers to, consider the following version of Box( ):

```
Box(double w, double h, double d)
{
        this.width = w;
        this.height = h;
        this.depth = d;
}
```

This version of Box() operates exactly like the earlier version. The use of this is redundant, but perfectly correct. Inside Box( ), this will always refer to the invoking object.

Eg.

```
        class Student
        {
                int rollno;
                String name;
                float fee;
                void setdata (int rollno,String name,float fee)
                {
                        this.rollno=rollno;
                        this.name=name;
                        this.fee=fee;
                }
        void display()
        {
                System.out.println(rollno+" "+name+" "+fee);
        }
}
class stud
{
        public static void main(String args[])
        {
                Student s1=new Student();
                s1.setdata(111,"pratham",5000f);
                Student s2=new Student();
                s2.setdata(112,"sumit",6000f);
                s1.display();
                s2.display();
        }
}
```

Static Keyword:

The **static keyword** in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

The *static* keyword is a non-access modifier in Java that is applicable for the following:

1. Blocks
2. Variables
3. Methods
4. Classes

When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.

```
class Test
{
   // static method
   static void m1()
   {
      System.out.println("from m1");
   }

   public static void main(String[] args)
   {
       // calling m1 without creating
       // any object of class Test
        m1();
   }
}
```

Static variables

When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

Important points for static variables:

- We can create static variables at the class level only.
- static block and static variables are executed in the order they are present in a program.

```
class Test
{
   // static variable
       static int a = m1();
    // static block
   static
   {
       System.out.println("Inside static block");
   }
    // static method
   static int m1()
   {
      System.out.println("from m1");
      return 20;
   }
    // static method(main !!)
   public static void main(String[] args)
   {
     System.out.println("Value of a : "+a);
     System.out.println("from main");
```

```
    }
  }
```

Static methods

When a method is declared with the *static* keyword, it is known as the static method. The most common example of a static method is the *main( )* method. As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object. Methods declared as static have several restrictions:

- They can only directly call other static methods.
- They can only directly access static data.
- They cannot refer to this or super in any way.

```
class Test
{
  // static variable
  static int a = 10;

  // instance variable
  int b = 20;

  // static method
  static void m1()
  {
    a = 20;
    System.out.println("from m1");

     // Cannot make a static reference to the non-static field b
     b = 10; // compilation error

     // Cannot make a static reference to the
          // non-static method m2() from the type Test
     m2();  // compilation error

     //  Cannot use super in a static context
     System.out.println(super.a); // compiler error
  }
  // instance method
  void m2()
  {
    System.out.println("from m2");
  }
   public static void main(String[] args)
  {
    // main method
  }
}
```

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable (It will be constant).

When a variable is declared with the final keyword, its value can't be modified, essentially, a constant. This also means that you must initialize a final variable.

If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but the internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from the final array or final collection. It is good practice to represent final variables in all uppercase, using underscore to separate words.

Eg.

```
class Bike
{
        final int speedlimit=90;//final variable
        void run(){
                speedlimit=400;   //gives error
        }
 public static void main(String args[])
{
        Bike obj=new  Bike();
        obj.run();
 }
```

Final classes

When a class is declared with final keyword, it is called a final class. A final class cannot be extended (inherited).

There are two uses of a final class:

Usage 1: One is definitely to prevent inheritance, as final classes cannot be extended. For example, all Wrapper Classes like Integer, Float, etc. are final classes. We can not extend them.

```
final class A
{
    // methods and fields
}
// The following class is illegal
class B extends A
{
   // COMPILE-ERROR! Can't subclass A
}
final class Bike
{

}
 class Honda1 extends Bike
```

```
{
          void run()
        {
                System.out.println("running safely with 100kmph");
        }
        public static void main(String args[])
        {
                Honda1 honda= new Honda1();
                honda.run();
        }
}
```

Final Methods

When a method is declared with final keyword, it is called a final method. A final method cannot be overridden. The Object class does this—a number of its methods are final. We must declare methods with the final keyword for which we are required to follow the same implementation throughout all the derived classes.

```
class A
{
   final void m1()
   {
      System.out.println("This is a final method.");
   }
}
class B extends A
{
   void m1()
   {
      // Compile-error! We can not override
      System.out.println("Illegal!");
   }
}
```

Eg.

```
class Bike
{
          final void run()
        {
                System.out.println("running");
        }
}
class Honda extends Bike
{
  void run()
  {
        System.out.println("running safely with 100kmph");
  }
 public static void main(String args[])
 {
          Honda honda= new Honda();
           honda.run();
  }
```

}

Overloading Methods

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a (int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading
Method overloading increases the readability of the program.
Different ways to overload the method
There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.
Eg.

```
class Adder
{
        static int add(int a,int b)
        {
                return a+b;
        }
        static int add(int a,int b,int c)
        {
                return a+b+c;
        }
}
class overload
{
        public static void main(String[] args)
        {
                System.out.println(Adder.add(11,11));  //static method called thru class nm
                System.out.println(Adder.add(11,11,11));
        }
}
```

When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call. Thus, overloaded methods must differ in the type and/or number of their parameters.

Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder
{
	static int add(int a, int b)
	{
		return a+b;
	}
	static double add(double a, double b)
	{
		return a+b;
	}
}
class overload2
{
	public static void main(String[] args)
	{
		System.out.println(Adder.add(11,11));
		System.out.println(Adder.add(12.3,12.6));
	}
}
```

Can we overload java main() method?
Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Constructors
In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

How Constructors are Different From Methods in Java?

- Constructors must have the same name as the class within which it is defined while it is not necessary for the method in Java.
- Constructors do not return any type while method(s) have the return type or **void** if does not return any value.
- Constructors are called only once at the time of Object creation while method(s) can be called any number of times.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.
Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors
There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

1. Default constructor

A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler creates a default constructor(with no arguments) for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor. Default constructor provides the default values to the object like 0, null, etc. depending on the type.

```
class test
{
        //creating a default constructor
        test()
        {
                System.out.println("object is created");
        }
        //main method
        public static void main(String args[])
        {
                //calling a default constructor
                test t=new test();
        }
}
```

2. Parameterized Constructor

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.
The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```
class Student
{
        int id;
        String name;
        //creating a parameterized constructor
        Student(int i,String n)
        {
                id = i;
                name = n;
        }
        //method to display the values
        void display()
        {
                System.out.println(id+" "+name);
```

```
    }
    public static void main(String args[])
    {
            //creating objects and passing values
            Student s1 = new Student (111,"Karan");
            Student s2 = new Student (222,"Aryan");
            //calling method to display the values of object
            s1.display();
            s2.display();
    }
}
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods. Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

```
//Java program to overload constructors
class Student
{
   int id;
   String name;
   int age;
   //creating two argt constructor
   Student(int i,String n)
   {
            id = i;
            name = n;
   }
   //creating three argt constructor
   Student (int i,String n,int a)
   {
            id = i;
            name = n;
            age=a;
   }
   void display()
   {
        System.out.println(id+" "+name+" "+age);
   }

   public static void main(String args[])
   {
            Student s1 = new Student (111,"Karan");
            Student s2 = new Student (222,"Aryan",25);
            s1.display();
            s2.display();
   }
}
```

Using Objects as Parameters

So far we have only been using simple types as parameters to methods. However, it is both correct and common to pass objects to methods. Whereas Objects in java are reference variables, so for objects a value which is the reference to the object is passed. Hence the whole object is not passed but its referenced gets passed. A constructor creates a new object initially the same as passed object. It is also used to initialize private members. For example, consider the following short program:

```
class Add
{
        private int a,b;

        Add(Add A)
        {
                a=A.a;
                b=A.b;
        }
        Add(int x,int y)
        {
                a=x;
                b=y;
        }
        void sum()
        {
                int sum1=a+b;
                System.out.println("Sum of a and b :"+sum1);
        }
}
class methodobj
{
        public static void main(String arg[])
        {
                Add A=new Add(15,8);
                Add A1=new Add(A);
                A1.sum();
        }
}
```

Variable Argument (Varargs):

The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

```
        return_type method_name(data_type... variableName)
        {
                //code
        }
class test
{
```

```
    static void display(String... values)
    {
        System.out.println("display method invoked ");
    }
    public static void main(String args[])
    {
        display();//zero argument
        display("my","name","is","varargs");//four arguments
    }
}
```

Finalize() method:

Finalize() is the method of Object class. This method is called just before an object is garbage collected. The Java finalize() method of Object class is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection, or we can say resource de-allocation. Remember, it is not a reserved keyword. Once the finalize() method completes immediately, Garbage Collector destroys that object.

Finalization: Just before destroying any object, the garbage collector always calls finalize() method to perform clean-up activities on that object. This process is known as Finalization in Java.

```
public class finalz
{
   public static void main(String[] args)
   {
       finalz obj = new finalz();
       System.out.println(obj.hashCode());
        obj = null;
           // calling garbage collector
           System.gc();
           System.out.println("end of garbage collection");
   }
   protected void finalize()
   {
       System.out.println("finalize method called");
   }
}
```

Instance Initializer block (IIB)

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created. The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

```
class Bike
{
    int speed;
    Bike()
    {
        System.out.println("constructor is invoked");
```

```
        }
        {
            System.out.println("instance initializer block invoked");
        }
        public static void main(String args[])
        {
            Bike8 b1=new Bike8();
            Bike8 b2=new Bike8();
        }
}
```