UNIT-1

Object Oriented Programming(OOPS): Object-oriented

programming (**OOP**) is a **programming** paradigm based on the concept of "**objects**", which can contain data and code: data in the form of fields (known as attributes or properties), and code, in the form of behaviour (known as methods).

Characteristic or Features of Object Oriented Programming:

Objects:

- An object is *a real-world entity*. An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.
- Example: Student, table, teacher, car, fan, chair etc..

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.
- For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

  Ex- In student object

  Roll no is identity

  Name is attribute

  getResult() is behaviour

Creating Objects:

Example:

Student S1= new Student();

Class:-

- Class is the blue-print of class. Or it is the template of object.
- The entire set of data and code of an object can be made a user-defined data type with the help of a class.
- The objects are variables of the type class.
- Once a class has been defined, we can create any number of objects belonging to that class.

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and functions from the class.

Declaring Classes:

We declare classes using the following syntax:-

```
class < Class Name >
{
variables…
methods…
}
```

Here class is a keyword.

Ex:

```
class Student{

int rollno;

String name;

void show(){

}

}
```

Data Hiding:

Data hiding is hiding the details of internal data members of an object.

- Data hiding is also known as Information hiding.
- Sometimes Data Hiding includes Encapsulation. Thus Data Hiding is heavily

related to Abstraction and Encapsulation.

- Data Hiding is the one most important OOP mechanism. Which is hide the details of the class from outside of the class.
- The Class used by only a limited set of variables and functions, others are hidden by the class.
- We use private keyword to hide data.
- Data hiding provide security to the data.

Ex:

```
class Student{
private int rollno;//data hiding using private access specifier
String name;
}
```

Abstraction:

Abstraction is the concept of object-oriented programming that "shows" only essential attributes and "hides" unnecessary information. The main purpose of abstraction is hiding the unnecessary details from the users. It helps in reducing programming complexity and efforts. It is one of the most important concepts of OOPs.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

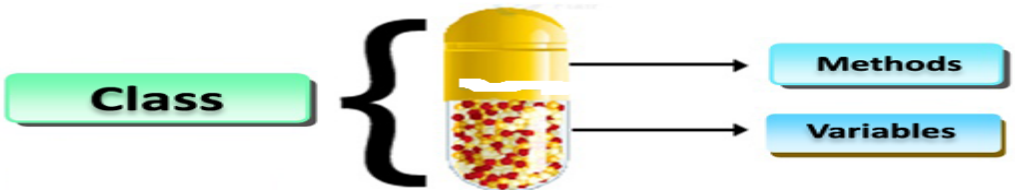
Encapsulation :

Encapsulation represents the information of variables (attributes) in terms of data and, methods (functions) and its operations in terms of purpose.

- Encapsulation is the process of combining data and function into a single unit called class.
- Encapsulation is a powerful feature that leads to information hiding and abstract data type.
- They encapsulate all the essential properties of the object that are to be

created.

- Using the method of encapsulation the programmer cannot access the class directly.



Ex: class Student{
int rollno;
String name;
void getName(){
}
}

Advantages of Encapsulation:

- The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.
- The major benefit of data encapsulation is the security of the data. It protects the data from unauthorized users that we inferred from the above stated real-real problem.
- We can apply the concept of data encapsulation in the marketing and finance sector where there is a high demand for security and restricted access of data to various departments.
- Encapsulation helps us in binding the data(instance variables) and the member functions(that work on the instance variables) of a class.
- Encapsulation is also useful in hiding the data(instance variables) of a class from an illegal direct access.
- Encapsulation also helps us to make a flexible code which is easy to change and maintain.

Dis-Advantage of Encapsulation

You can't access private date outside the class.

Inheritance :

- When the properties and the methods of the parent class are accessed by the child class, Then it called inheritance. The child class can inherit the parent variables and methods and can also give own method implementation.

In the inheritance the class which is give data members and methods is known as base or super or parent class.

The class which is taking the data members and methods is known as sub or derived or child class.

For code Re-usability is provided by inhritance

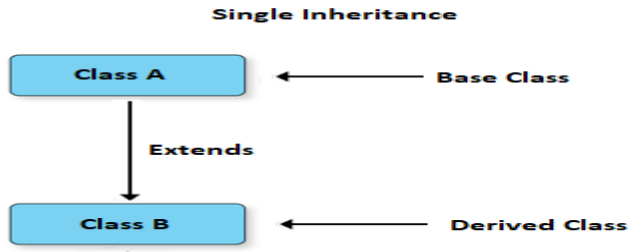
Types of Inheritance

Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance; they are:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

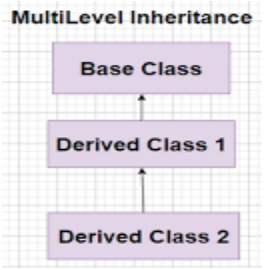
Single inheritance

In single inheritance there exists single base class and single derived class.



Multilevel inheritances

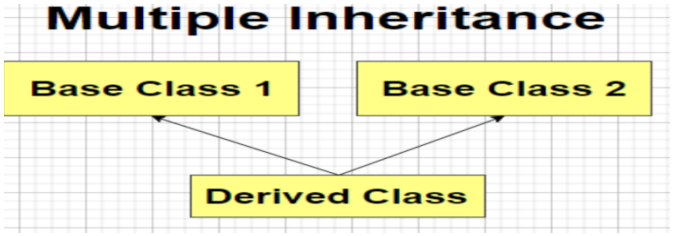
In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.



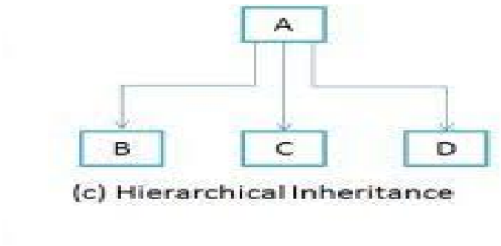
Multiple inheritance

In multiple inheritance there exist multiple classes and singel derived class.

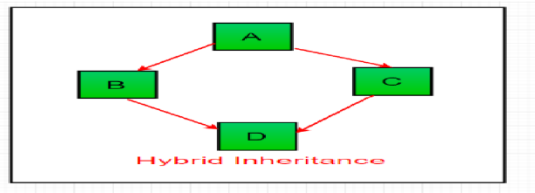
The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.



Hierarchical Inheritance: When more than one classes inherit a same class then this is called *hierarchical inheritance*. For example class B, C and D inherit a same class A.
L



Hybrid inheritance: In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.



Advantage of inheritance

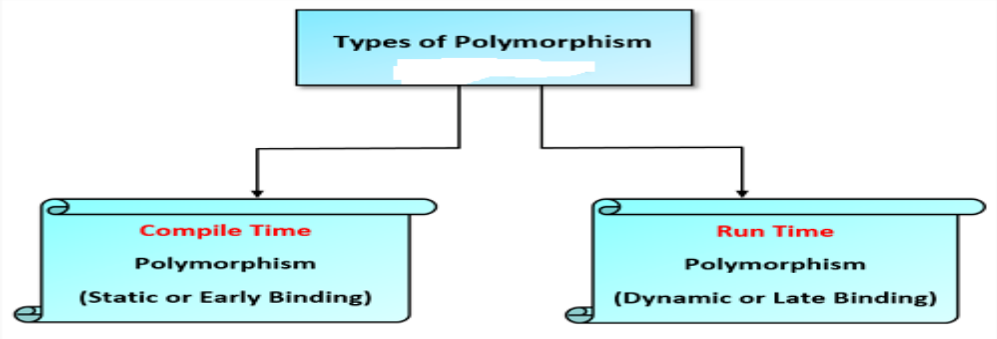
If we develop any application using concept of Inheritance than that application have following advantages,

- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

Polymorphism

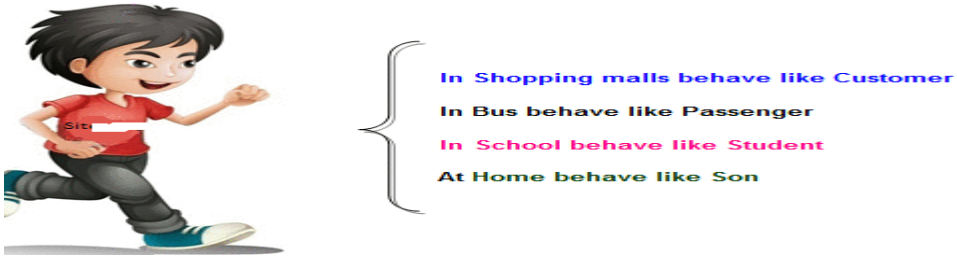
The process of representing one form in multiple forms is known as **Polymorphism**.

Polymorphism is derived from 2 greek words: **poly** and **morphs**. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.



Real life example of polymorphism

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.



How to achieve Polymorphism in Java ?

In java programming the Polymorphism principal is implemented with method overriding concept of java.

Polymorphism principal is divided into two sub principal they are:

- Static Binding or Compile time polymorphism
- Dynamic Binding or Runtime polymorphism

Static polymorphism

- The process of binding the overloaded method within object at compile time is known as **Static polymorphism** due to static polymorphism utilization of resources (main memory space) is poor because for each and every overloaded method a memory space is created at compile time when it binds with an object.
- Compile time polymorphism or static polymorphism relates to method overloading.

Dynamic polymorphism

- In dynamic polymorphism method of the program binds with an object at runtime the advantage of dynamic polymorphism is allocating the memory space for the method (either for overloaded method or for override method) at run time.
- Run time polymorphism or dynamic polymorphism or dynamic binding relates to method

What is Java

Java is a high level, robust, secured and object-oriented programming language.

Characteristics or Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.
Simple
Object-Oriented
Platform independent
Secured
Robust
Portable
Dynamic
Compiled and Interpreted
High Performance
Multithreaded
Distributed

Simple

Java language is simple because:
syntax is based on C++ (so easier for programmers to learn it after C++).
removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
No need to remove objects because there is Automatic Garbage Collection in java.

Object-oriented

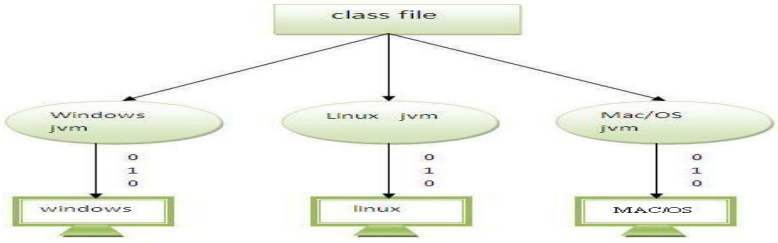
Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Basic concepts of OOPs are:
Object
Class
Data Hiding
Abstraction
Encapsulation
Inheritance
Polymorphism

Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the

sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:
1. Runtime Environment
2. API(Application Programming Interface)



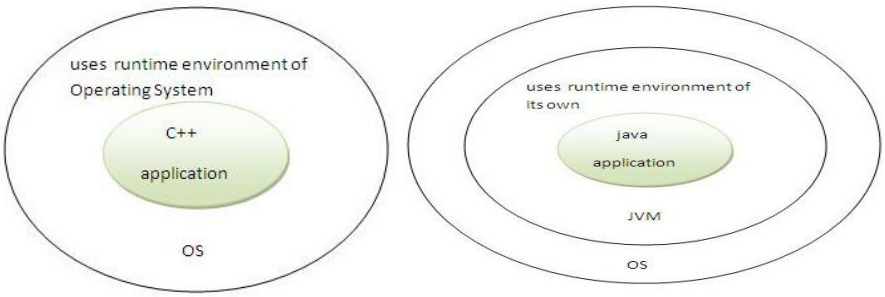
Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured because:

No explicit pointer

Programs run inside virtual machine sandbox.



Classloader- adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier- checks the code fragments for illegal code that can violate access right to objects.

Security Manager- determines what resources a class can access such as reading and writing to the local disk.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Portable

We may carry the java bytecode to any platform. i.e it can be run in windows,linux,unix ,macOS etc. operating system.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code

still somewhat slower than a compiled language (e.g., C++)

Distributed
We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.
Multi-threaded
A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

Example of Creating hello java program:

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

save this file as Simple.java
To compile: javac Simple.java
To execute: java Simple
Output: Hello Java

Understanding first java program:
class :keyword is used to declare a class in java.
public :keyword is an access modifier which represents visibility, it means it is visible to all.
static :is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
void :is the return type of the method, it means it doesn't return any value.
main :represents startup of the program.
String[] args :is used for command line argument.
System.out.println() :is used print statement. Where System is class and out is a static variable defined in System class and print() is a method defined inside the PrintStream class.

JVM (Java Virtual Machine) Architecture

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

The JVM performs following operation:

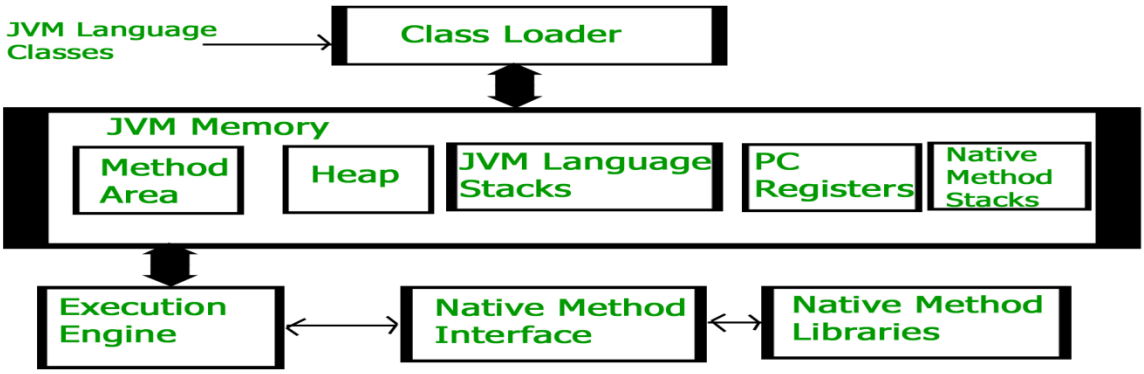
- Loads code
- Verifies code
- Executes code

- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

JVM Architecture



1) Classloader

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

1. **Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area

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

3) Heap

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

4) Stack

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

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

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

5) Program Counter Register

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

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

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

8) Java Native Interface

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

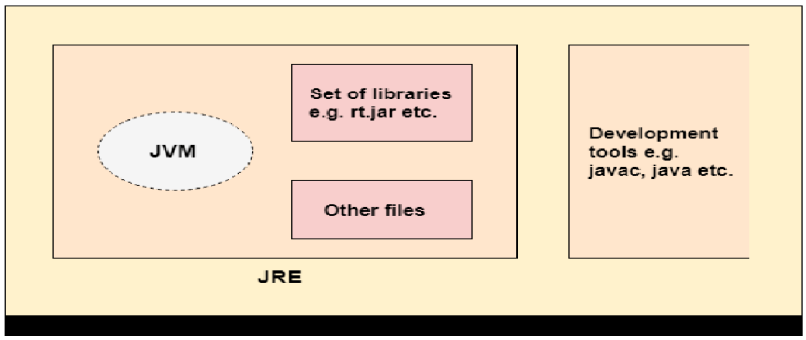
JRE:

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.

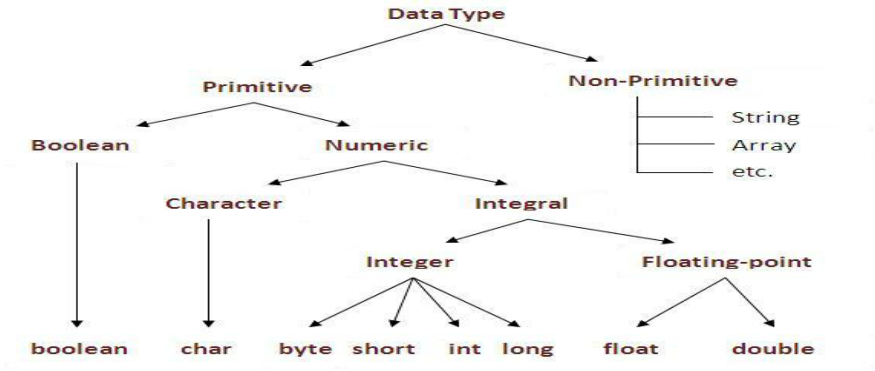
JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications . It physically exists. It contains JRE + development tools.



Data Types in Java:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:



Data Type	Default Value	Size
boolean	False	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

Variables in java:

Variable is an identifier which holds data or another one variable is an identifier whose value can be changed at the execution time of program. Variable is an identifier which can be used to identify input data in a program.

Rules to declare a Variable

- Every variable name should start with either alphabets or underscore (_) or dollar ($) symbol.
- No space are allowed in the variable declarations.
- Except underscore (_) no special symbol are allowed in the middle of variable declaration
- Variable name always should exist in the left hand side of assignment operators.
- Maximum length of variable is 64 characters.
- No keywords should access variable name

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

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

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

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

3) Static variable

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

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

Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.
But, it is not forced to follow. So, it is known as convention not rule.
All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.

variable name should start with lowercase letter e.g. firstName, orderNumber etc.

package name should be in lowercase letter e.g. java, lang, sql, util etc.

constants name should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

CamelCase in java naming conventions:

Java follows camelcase syntax for naming the class, interface, method and variable. If name is combined with two words, second word will start with uppercase letter always e.g. method names : actionPerformed(), firstName

Class names: ActionEvent, ActionListener

Java Tokens:

The smallest units of Java language are called Java tokens. There are five tokens available in java

1. Keywords 2.Identifiers 3.Literals 4. Separators 5.Operators

1.Keywords:

Java Keywords also called a reserved word. Keywords are Java reserves words for its own use. These have built-in meanings that cannot change. There are 49 reserved keywords currently defined in the Java language and they are shown in the below table.

abstract	Double	Int	Switch
assert	Else	interface	Synchronized
boolean	Extends	long	This
break	False	native	Throw
byte	Final	new	Transient
case	Finally	package	True
catch	Float	private	Try
char	For	protected	Void
class	Goto	public	Volatile
const	If	return	While
continue	implements	short	
default	Import	static	
Do	instanceof	super	

The keywords const and goto are reserved but not used.

Identifiers:

Identifiers are defined by programmer.

Java developers follow naming conventions for writing identifiers.

Ex:-Names of all methods and variables, classes, interfaces, packages, constant

Example: average
sum

Literals:--

=>Literals in Java are a sequence of characters (digits, letters, and other characters) that represent values to be stored in variables.

=>Java language specifies five major types of literals. Literals can be any number, text, or other information that represents a value. They are:
Integer literals
Ex:10,20
Floating literals
Ex:12.33
Character literals
Ex: 'c'
String literals
Ex: "Hello"
Boolean literals
Ex: true and false
Separators:---
Separators helps to define the structure of a program. The separators are parentheses, ( ), braces, { }, the period, ., and the semicolon, ;.

Operators:---
Arithmetic , relational, bitwise, logic , Assignment and conditional operator.

JAVA TYPE CASTING or TYPE CONVERSION:
Conversion of one data type to another data type is called type casting.
There are two types of type casting:
1.implicit type casting or widening or upcasting or automatic
2. Explicit type casting or narrowing or downcasting.

1. Implicit type casting:-- Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.

```
class Test
{
   public static void main(String[] args)
   {
      int i = 100;

      //automatic type conversion
      long l = i;

      //automatic type conversion
      float f = l;
      System.out.println("Int value "+i);
      System.out.println("Long value
"+l);
      System.out.println("Float value
"+f);
   }
}
```
Output:

Int value 100
Long value 100
Float value 100.0

2.Narrowing or Explicit Type Castinng / Conversion:

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.

```
class Test
{
   public static void main(String[] args)
   {
      double d = 100.04;
      //explicit type casting
      long l = (long)d;
     //explicit type casting
      int i = (int)l;
      System.out.println("Double value "+d);
     //fractional part lost
      System.out.println("Long value "+l);
      //fractional part lost
      System.out.println("Int value "+i);
   }
}
```

Output:

Double value 100.04
Long value 100
Int value 100

Operators in Java:

Operator in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Unary Operator Example: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10
12
12
10
```

Unary Operator Example 2: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21

}}
```

Output:

```
22
21
```

Unary Operator Example: ~ and !

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
```

}}

Output:

```
-11
9
false
true
```

Arithmetic Operators

Java arithmatic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Arithmetic Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15
5
50
2
0
```

Arithmetic Operator Example: Expression

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

Output:

```
21
```

Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Left Shift Operator Example

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
```

```
System.out.println(15<<4);//15*2^4=15*16=240
}}
```
Output:
```
40
80
80
240
```

Right Shift Operator

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Right Shift Operator Example

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```
Output:
```
2
5
2
```

Shift Operator Example: >> vs >>>

```
class OperatorExample{
public static void main(String args[]){
   //For positive number, >> and >>> works same
   System.out.println(20>>2);
   System.out.println(20>>>2);
   //For negative number, >>> changes parity bit (MSB) to 0
   System.out.println(-20>>2);
   System.out.println(-20>>>2);
}}
```
Output:
```
5
5
-5
1073741819
```

AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

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

```
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```
Output:
```
false
false
```

AND Operator Example: Logical && vs Bitwise &

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);//false && true = false
System.out.println(a);//10 because second condition is not checked
System.out.println(a<b&a++<c);//false && true = false
System.out.println(a);//11 because second condition is checked
}}
```
Output:
```
false
10
false
11
```

OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```
Output:
```
true
true
true
```

10
true
11

Ternary Operator

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in Java programming. it is the only conditional operator which takes three operands.

Ternary Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:
2

Assignment Operator

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

Assignment Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:
14
16

Assignment Operator Example

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
```

}}
Output:
13
9
18
9

Control Flow:

Conditional Statement:

1.if statement
2.if-else statement
3.if-else-if ladder
4.nested if statement
5.switch statement

if Statement:

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if(condition){
//code to be executed
}
```

Example:

```
public class IfExample {
public static void main(String[] args) {
//defining an 'age' variable
int age=20;
//checking the age
if(age>18){
System.out.print("Age is greater than 18");
}
}
}
```

Output:

Age is greater than 18

if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition){
//code if condition is true
}
else{
```

```
//code if condition is false
}
```

Example:

```
public class IfElseExample {
public static void main(String[] args) {
//defining a variable
int number=13;
//Check if the number is divisible by 2 or not
if(number%2==0){
System.out.println("even number");
}else{
System.out.println("odd number");
}
}
}
```

Output:

```
odd number
```

if-else-if ladder Statement:

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Example:

```
public class IfElseIfExample {
public static void main(String[] args) {
    int marks=65;

    if(marks<50){
        System.out.println("fail");
    }
```

```
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
        System.out.println("Invalid!");
    }
}
}
```

Output:

C grade

Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
    }
}
```

Example:

```
    public class NestedIfExample {
    public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=20;
    int weight=80;
    //applying condition on age and weight
    if(age>=18){
    if(weight>50){
    System.out.println("You are eligible to donate blood");
    }
    }
    }}
```

Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else -if ladder statement.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

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

Example:

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

Output:

```
20
```

Looping Statement:

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

- for loop
- while loop
- do-while loop

For Loop

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

types of for loops in java.

Simple For Loop

For-each or Enhanced For Loop

Simple For Loop

In simple for loop we can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

Syntax:

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

Example:

```
public class ForExample {
public static void main(String[] args) {
//Code of Java for loop
for(int i=1;i<=10;i++){
System.out.println(i);
} }}
```

for-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript

notation.

It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```
for(Type var:array){
//code to be executed
}
```

Example:

```
public class ForEachExample {
public static void main(String[] args) {
//Declaring an array
int arr[]={12,23,44,56,78};
//Printing array using for-each loop
for(int i:arr){
System.out.println(i);
}
}
}
```

Output:

```
12
23
44
56
78
```

While Loop

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

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

Example:

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

do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

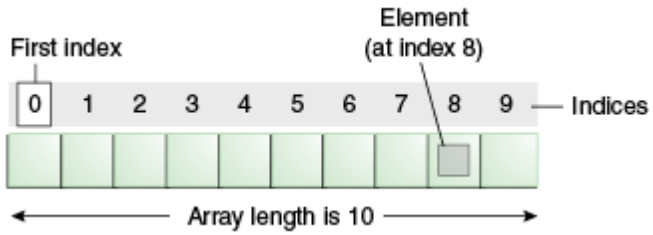
```
do{
//code to be executed
}while(condition);
```

Example:

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

Arrays

an array is a collection of similar type of elements which has contiguous memory location. Array is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't

grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=new datatype[size];

Example of Array:

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Example:

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
33
3
4
5
```

Multidimensional Array in Java

data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

example to declare, instantiate, initialize and print the 2Dimensional array.

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

```
//printing 2D array
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
System.out.print(arr[i][j]+" ");
}
System.out.println();
}
}}
```

Output:

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

Objects in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

  Creating object in a student class :

Ex: Student st=new Student();

Class in java:

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields or variables**
- **Methods or functions**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Syntax to declare a class:

```
class <class_name>{
   variables;
```

```
    method;
}
```

Object and Class Example: main within the class:

```
//Defining a Student class.
class Student{
//defining fields
int id;//field or data member or instance variable
String name;
//creating main method inside the Student class
public static void main(String args[]){
//Creating an object or instance
Student s1=new Student();//creating an object of Student
//Printing values of the object
System.out.println(s1.id);//accessing member through reference variable
System.out.println(s1.name);
}
}
```

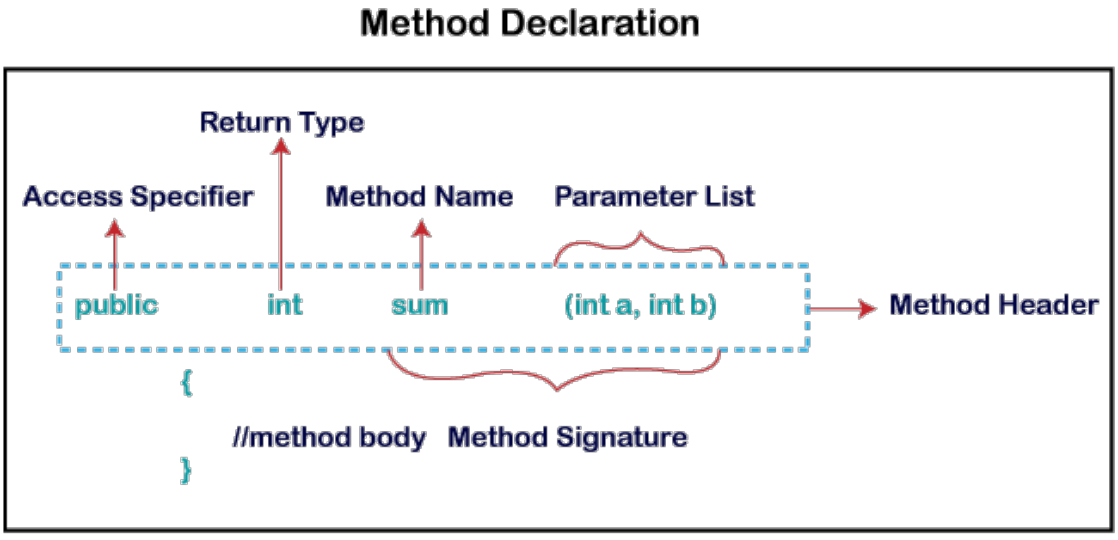
Output:

```
0     // The output is 0 because the default value of int
Null   //The output is Null because the default value of object or class
```

Method in Java:

a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods**.

Method Declaration



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Example:

```
import java.util.Scanner;
public class EvenOdd
{
//user defined method
 void findEvenOdd(int num)
{
//method body
if(num%2==0)
System.out.println(num+" is even");
else
System.out.println(num+" is odd");
}


public static void main (String args[])
{
//creating Scanner class object  for taking input from user.
Scanner scan=new Scanner(System.in);
System.out.print("Enter the number: ");
//reading value from user
```

```
int num=scan.nextInt();
EvenOdd    ob=new EvenOdd();
//method calling
Ob.findEvenOdd(num);
}
}
```

Output 1:

```
Enter the number: 12
12 is even
```

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.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

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

Default Constructor:

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>(){
}
```

Example of default constructor:

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

Output:
Employee is created

Note: If there is no constructor in a class, compiler automatically creates a default constructor. As example below

Example of default constructor that displays the default values

```
class Student{
int id;
String name;
//method to display the value of id and name
void display(){
System.out.println(id+" "+name);
}
  public static void main(String args[]){
//creating objects
Student  s1=new Student3();
Student  s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Output:

```
0 null
0 null
```

Parameterized Constructor:

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

```
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(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
    Student4 s1 = new Student4(111,"Kiran");
    Student4 s2 = new Student4(222,"Aru");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Output:
111 Kiran
222 Arun

Constructor Overloading :

Constructor overloading 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

Example of Constructor Overloading

```
. //Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n){
    id = i;
    name = n;
    }
    //creating three arg constructor
    Student5(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[]){
    Student5 s1 = new Student5(111,"Kiran");
    Student5 s2 = new Student5(222,"Arun",25);
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Kiran 0
222 Arun 25
```

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.