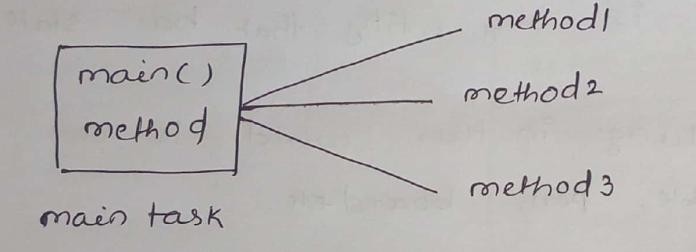
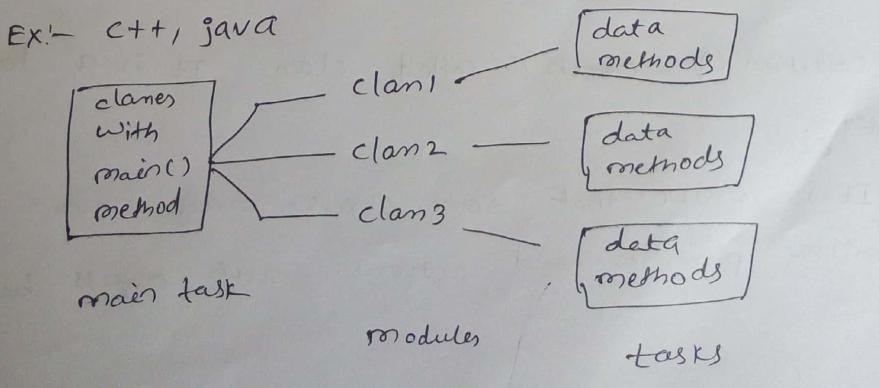
# Class and Object Basics Procedure Oriented Approach: -

* Procedure oriented programming languages, a programmer uses procedures or functions to perform a task.
* When a programmer wants to write a program, he will first divide the task into separate sub tasks, each of which is expressed as a function.
* Every program generally contains several functions which are called and controlled from a main() function.
* This approach is called procedure oriented approach.
* Ex: C, Pascal, Fortran … etc.

# OOP approach:-

* In OOP uses classes and objects in their programs. A class is a module which itself contains data and methods to achieve tasks.
* The main task is divided into several methods and they are represented as classes.

Each class can perform some tasks for which several methods are written in a class. This approach is called OOP approach.

* Ex: C++, Java, Python

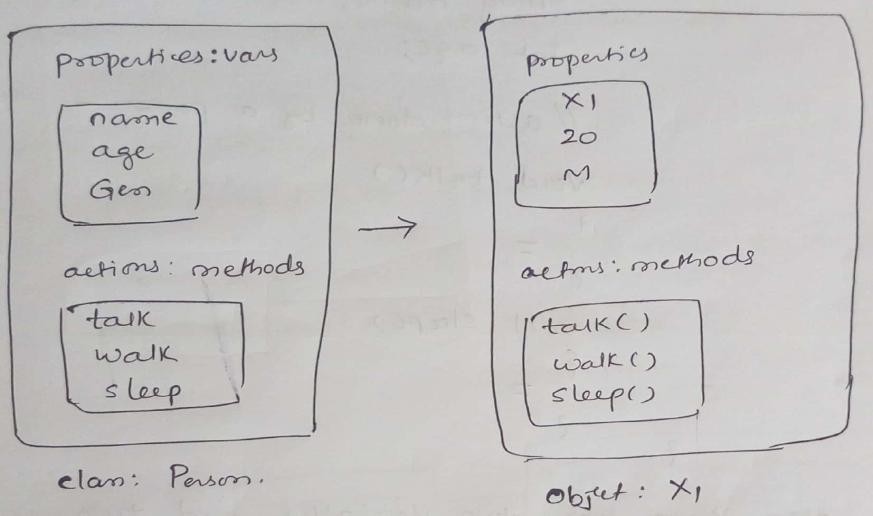
# Features of OOP systems:

* There are many features for OOP approach:
  + class / Object
  + Encapsulation
  + Inheritance
  + Abstraction
  + Polymorphism

# Class / Object:-

* OOP methodology has been derived from a single root concept called „Object‟.
* An object is any entity that has state and behavior.
* So everything in this world is an object.
  + Ex: table, pen, board … etc.
* Every object has properties and can perform certain actions.
  + Let us take a person X. X is a object because he exists physically. He has properties can be represented by variables in programming.
  + Ex: String name;

int age;

* Collection of objects is called class. It is a logical entity.
* It is possible that some objects may have similar properties and actions. Such objects belongs to same category called a „class‟.
  + Ex: X1,X2,X3 … persons have same properties and actions. So they are all objects of same class Person.
* Person class is not exist physically, but objects exist physically.

# Differences b/w class and Object:

* A class is a model for creating objects and doesnot exist physically.
* A object is anything that exists physically
* Both the class and object contains Variables and methods.

# Example:

class Employee

{

int eid; // data member (or instance variable)

String ename; // data member (or instance variable) eid=101;

ename="Hitesh";

public static void main(String args[])

{

Employee e=new Employee(); // Creating an object of class Employee System.out.println("Employee ID: "+e.eid);

System.out.println("Name: "+e.ename);

}

}

# Creating classes and Objects in Java:

Let us create a class with the name Person for which X and X1 are objects. A class is created by using the keyword „class‟.

A class describes the properties and actions performed by its objects. So the properties (variables) and actions (methods) in the class as:

class Person{

String name; //properties of a person-variables int age;

//actions done by a person – methods void talk()

{

//code

}

void sleep()

{

//code

}

}

class Person has two variables and two methods. This class method is stored in JVM‟s method area. When we want to use this class, we should create an object to the class as:

class Demo{

public static void main(String args[])

{

//create Person class object: X Person X =new Person();

//call the methods System.out.println (X.name);

//output is : null

System.out.println (X.age);

// output is: 0

}

}

# Object creation:

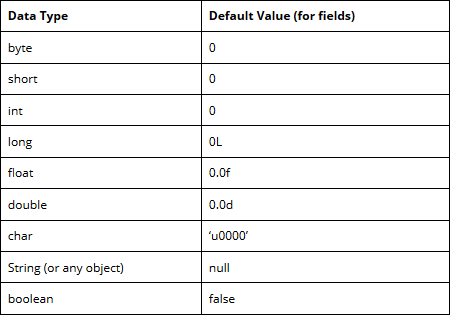
The class code along with method code is stored in „Method area‟ of the JVM When an object is created, the memory is allocated on heap, after creation of the object. JVM produced a unique reference number for the object from the memory address of the object. This reference number is called hash code num.

To know the Object reference number, we can use hashCode() method of Object class.

Ex: Person x = new Person(); // x ref. to Person obj.

System.out.println(x.hashCode()); //displays the hash code ref. num stored in x.

# Default values of instance variables by java compiler:



Example:

class Person{

String name; int age;

void talk()

{

System.out.println (“My name is :”+name);

System.out.println(“My age is: “+age);

}}

Class Demo{

public static void main(String args[]){ Person x = new Person();

x.talk();

}

}

O/P: My name is null My age is 0

# Initializing the Objects in Java:

There are 3 ways to initialize object in java.

1. By reference variable
2. By method
3. By constructor

# Object and Class Example: Initialization through reference

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

# Example:

class Student{ int id;

String name;

}

class TestStudent2{

public static void main(String args[]){ Student s1=new Student();

s1.id=10; s1.name="RGUKT";

System.out.println(s1.id+" "+s1.name);//printing members with a white space

}

}

Output: 10 RGUKT

# Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

Example:

class Student{ int rollno; String name;

void insertRecord(int r, String n){ rollno=r;

name=n;

}

void displayInformation(){System.out.println(rollno+" "+name);}

}

class TestStudent4{

public static void main(String args[]){ Student s1=new Student();

Student s2=new Student(); s1.insertRecord(111,"Karan"); s2.insertRecord(222,"Aryan"); s1.displayInformation(); s2.displayInformation();

}

}

Output:

111 Karan

222 Aryan

# Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables: int a=10, b=20;

Initialization of refernce variables:

Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects

Example:

class Rectangle{ int length;

int width;

void insert(int l,int w){ length=l;

width=w;

}

void calculateArea(){System.out.println(length\*width);}

}

class TestRectangle2{

public static void main(String args[]){

Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects r1.insert(11,5);

r2.insert(3,15); r1.calculateArea(); r2.calculateArea();

}

}

Output:

55

45

# Real World Example: Account

class Account{ int acc\_no; String name; float amount;

void insert(int a,String n,float amt){ acc\_no=a;

name=n; amount=amt;

}

void deposit(float amt){ amount=amount+amt; System.out.println(amt+" deposited");

}

void withdraw(float amt){ if(amount<amt){ System.out.println("Insufficient Balance");

}else{ amount=amount-amt;

System.out.println(amt+" withdrawn");

}

}

void checkBalance(){System.out.println("Balance is: "+amount);} void display(){System.out.println(acc\_no+" "+name+" "+amount);}

}

class TestAccount{

public static void main(String[] args){ Account a1=new Account(); a1.insert(832345,"Ankit",1000); a1.display();

a1.checkBalance(); a1.deposit(40000); a1.checkBalance(); a1.withdraw(15000); a1.checkBalance();

}}

Output:

832345 Ankit 1000.0

Balance is: 1000.0

40000.0 deposited

Balance is: 41000.0

15000.0 withdrawn

Balance is: 26000.0

# Initializing through constructor:

class Person{

//instance variables String name;

int age;

Person(){ //default constructor

{

name=”RGUKT”; age=10;

}

void talk(){

System.out.println(“My name is: “+name); SOP(“My age is: “+age);

}

}

class Demo{

public static void main(String args[]){ Person x = new Person();

x.talk();

}

}

O/P: My name is RGUKT My age is 10

# Access Modifiers in Java

**Access modifiers** are those which are applied before data members or methods of a class. These are used to where to access and where not to access the data members or methods.

In Java programming these are classified into four types:

* + Private
  + Default (not a keyword)
  + Protected
  + Public

Default is not a keyword (like public, private, protected are keyword). If we are not using private, protected and public keywords, then JVM is by default taking as default access modifiers.

Access modifiers are always used for, how to reuse the features within the package and access the package between class to class, interface to interface and interface to a class. Access modifiers provide features accessing and controlling mechanism among the classes and interfaces.

Protected members of the class are accessible within the same class and another class of same package and also accessible in inherited class of another package.

# Rules for access modifiers:

The following diagram gives rules for Access modifiers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifiers** | **Within same Class** | **Within other class of same**  **package** | **Within derived class of other**  **package** | **Within external class of other**  **package** |
| Private | Yes | No | No | No |
| Default | Yes | Yes | No | No |
| Protected | Yes | Yes | Yes | No |
| Public | Yes | Yes | Yes | Yes |

**private:** Private members of class in not accessible anywhere in program these are only accessible within the class. Private are also called class level access modifiers.

# Example

class Hello

{

private int a=20; private void show()

{

System.out.println("Hello java");

}

}

public class Demo

{

public static void main(String args[])

{

Hello obj=new Hello();

System.out.println(obj.a); //Compile Time Error, you can't access private data obj.show(); //Compile Time Error, you can't access private methods

}

}

**public:** Public members of any class are accessible anywhere in the program in the same class and outside of class, within the same package and outside of the package. Public are also called universal access modifiers.

# Example

class Hello

# Output

20

{

public int a=20; public void show()

{

System.out.println("Hello java");

}

}

public class Demo

{

public static void main(String args[])

{

Hello obj=new Hello(); System.out.println(obj.a); obj.show();

}

}

Hello Java

**protected:** Protected members of the class are accessible within the same class and another class of the same package and also accessible in inherited class of another package. Protected are also called derived level access modifiers.

In below the example we have created two packages pack1 and pack2. In pack1, class A is public so we can access this class outside of pack1 but method show is declared as a protected so it is only accessible outside of package pack1 only through inheritance.

# Example

// save A.java package pack1; public class A

{

protected void show()

{

System.out.println("Hello Java");

}

}

//save B.java

package pack2; import pack1.\*;

class B extends A

{

public static void main(String args[]){ B obj = new B();

obj.show();

}

}

# Output

Hello Java

**default:** Default members of the class are accessible only within the same class and another class of the same package. The default are also called package level access modifiers.

# Example

//save by A.java package pack; class A

{

void show()

{

System.out.println("Hello Java");

}

}

//save by B.java package pack2; import pack1.\*; class B

{

public static void main(String args[])

{

A obj = new A(); //Compile Time Error, can't access outside the package obj.show(); //Compile Time Error, can't access outside the package

}

}

# Output

Hello Java

# Constructors:-

A constructor is similar to a method that is used to initialize the instance variables. The sole purpose of a constructor is a to initialize the instance variables. Characteristics of constructors:-

- The constructor‟s name and class name should be same. The constructor‟s name should end with a pair of braces.

Ex: Person()

{

}

- A constructor may have or may not have parameters. parameters are variables to receive data from outside into the constructor If a constructor doesnot have any parameters, it is called as „default constructor‟. If a constructor have 1 or more parameters, it is called parameterized constructor.

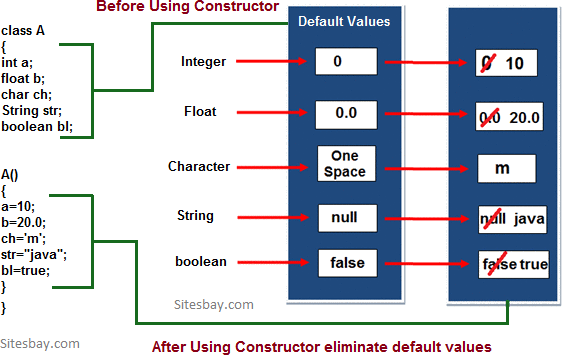
Ex:

parameterized constructor:

Person(int i, int s){

}

* A constructor does not return any value, not even void type.
* A constructor is automatically called and executed at the time of creating an object. While creating an object, if nothing is passed to the object the default constructor is called and executed.
* If some values are places to the object, then the parameterized constructor is called.
* A constructor is called and executed only once per object.



Example: - example for default constructor

class Sum

{

int a,b; Sum()

{ a=10; b=20;

}

public static void main(String s[])

{

Sum s=new Sum(); c=a+b;

System.out.println("Sum: "+c);

}

}

Example: parameterized constructor

class Test

{

int a, b;

Test(int n1, int n2)

{

System.out.println("I am from Parameterized Constructor..."); a=n1;

b=n2;

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

}

}

class TestDemo1

{

public static void main(String k [])

{

Test t1=new Test(10, 20);

}

}

# Methods in Java:-

A method represents a group of statements that performs a task. Here the task represents a calculation or processing of data or generating a report etc..

ex: sqrt() method

It calculates square root value and returns that value. Method has two parts:

1. method header or method prototype
2. method body
3. Method header or method prototype:

It contains method name, method parameters and method return type.

Syntax:

return datatype methodname(parameter1, parameter2….)

1. Method body:

Method body consists a group of statements which contains logic to perform the

task. Syntax:

{

}

stmts;

Method types:

* 1. No parameters and no return type
  2. with parameters and with return type
  3. without parameters and with return type
  4. with parameters and without return type

# Static Methods:

* A static method is a method that doesn‟t act upon instance variables of a class.
* A static method is declared by using the keyword static.
* A static methods are called using **classname.methodname()**
* The reason why static methods can‟t act on instance variables in that the JVM first executes the static methods and they only it creates the objects. Since the objects are not available at the time of calling the static methods, the instance variables are also not available.

The static methods can be:

1. static variables
2. static methods
3. static block

# static variables:-

* + static variables in java is available which belongs to the class and initialized only once at the start of the execution.
  + It is a variable which belongs to the class and not to the object (instance).
  + static variables initialized first, before the initialization of any instance variables.
  + a single copy to be shared by all instance of a class.
  + A static variable can be accessed directly by the class name and doesn‟t need any object.
  + Syntax: classname.varablename; Example:

class Person{ int rollno; String name;

static String branch=”ECE”; Person(int n,String s){ rollno=n;

name=s;

}

void display(){

SOP(rollno+” “+name+” “+branch);

}

}

class Person\_Demo{

public static void main(String args[]){ Person p=new Person(1001,”RGUKT”); p.display();

}

}

O/P: 1001 RGUKT ECE

# static methods:

* + static method in java is a method which belongs to the class and not the object. A static method can access only static data.
  + It can‟t access non static data.
  + A static method can be accessed directly by the class name and doesn‟t need any object.
  + A static method can‟t refer „this‟ and „super‟ keywords.
  + Syntax: **classname.methodname();**

Example:

O/P: 55

class Test{

static int count=10; static void access(){ SOP(x);

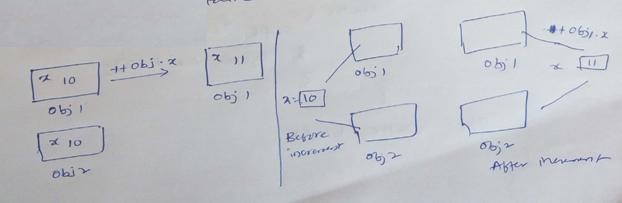
}

public static void main(String args[]){ Test.access()

}}

# Difference between instance variables and class variables:-

* + An instance variable is a variable whose separate copy is available to each objects
  + A class variable is a variable whose single copy in memory is shared by all objects
  + Instance variables are created in the objects on heap memory.
  + Class variables (static variables) are stored on method area.



# static block:-

A static block is block of statements declared as static. Syntax:

static{

//code

}

- JVM executes a static block on a highest priority basis. This means JVM first goes to static block even before it looks for main() method in a program.

Example:

class Test{ static{

SOP(“static block”);

}

public static void main(String args[]){ SOP(“main block”);

}}

O/P: static block

main block

# This keyword:

**this** is a reference variable that refers to the current object. It is a keyword in java language represents current class object

# Usage of this keyword

* It can be used to refer current class instance variable.
* this() can be used to invoke current class constructor.
* It can be used to invoke current class method (implicitly)
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* It can also be used to return the current class instance.

# Use of this keyword in java:

The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then JVM get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

**"this"** keyword can be use in two ways.

* this . (this dot)
* this() (this off)

# this . (this dot)

which can be used to differentiate variable of class and formal parameters of method or constructor.

# Syntax

this.data member of current class.

# Example without using this keyword

class Employee

{

int id;

String name;

Employee(int id,String name)

{

id = id;

name = name;

}

void show()

{

System.out.println(id+" "+name);

}

public static void main(String args[])

{

Employee e1 = new Employee(111,"rgukt"); Employee e2 = new Employee(112,"iiit");

# Output

0 null

0 null

e1.show();

e2.show();

}

}

In the above example, parameter (formal arguments) and instance variables are same that is why we are using **"this"** keyword to distinguish between local variable and instance variable.

# Example of this keyword in java

class Employee

{

int id;

String name;

Employee(int id,String name)

{

this.id = id; this.name = name;

}

void show()

{

System.out.println(id+" "+name);

}

public static void main(String args[])

{

Employee e1 = new Employee(111,"rgukt"); Employee e2 = new Employee(112,"iiit"); e1.show();

e2.show();

}

}

# Output

1. rgukt
2. iiit

# Difference between this and super keyword

**Super keyword** is always pointing to base class (scope outside the class) features and

**"this"** keyword is always pointing to current class (scope is within the class) features.

# Example when no need of this keyword

class Employee

{

int id;

String name;

Employee(int i,String n)

{

id = i; name = n;

}

void show()

{

System.out.println(id+" "+name);

}

public static void main(String args[])

{

Employee e1 = new Employee(111,"rgukt"); Employee e2 = new Employee(112,"iiit"); e1.show();

e2.show();

}

}

# Output

1. rgukt
2. iiit

In the above example, no need of use this keyword because parameter (formal arguments) and instance variables are different. This keyword is only use when parameter (formal arguments) and instance variables are same.

# Method Overloading in Java

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as **method overloading**.

# Use of Overloading in Java:

Suppose we have to perform addition of given number but there can be any number of arguments, if we write method such as a(int, int)for two arguments, b(int, int, int) for three arguments then it is very difficult for you and other programmer to understand purpose or behaviors of method they cannot identify purpose of method. So we use method overloading to easily figure out the program. For example above two methods we can write sum(int, int) and sum(int, int, int) using method overloading concept.

# Syntax

class class\_Name

{

Returntype method()

{.........}

Returntype method(datatype1 variable1)

{.........}

Returntype method(datatype1 variable1, datatype2 variable2)

{.........}

Returntype method(datatype2 variable2)

{.........}

Returntype method(datatype2 variable2, datatype1 variable1)

{.........}

}

# Different ways to overload the method

There are two ways to overload the method in java

* 1. By changing number of arguments or parameters
  2. By changing the data type

# By changing number of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

# Example Method Overloading in Java

**Output**

30

60

class Addition

{

void sum(int a, int b)

{

System.out.println(a+b);

}

void sum(int a, int b, int c)

{

System.out.println(a+b+c);

}

public static void main(String args[])

{

Addition obj=new Addition(); obj.sum(10, 20);

obj.sum(10, 20, 30);

}

}

# By changing the data type

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

# Example Method Overloading in Java

class Addition

{

void sum(int a, int b)

{

System.out.println(a+b);

}

void sum(float a, float b)

{

System.out.println(a+b);

}

public static void main(String args[])

# Output

30

25.25

{

Addition obj=new Addition(); obj.sum(10, 20);

obj.sum(10.05, 15.20);

}

}

# Call by value and call by Object:

As per java specification everything in java is passed by value whether it‟s primitive data type of objects and java doesn‟t support pointers.

The changes being done in the called method, isn‟t affected in the calling method.

Ex:- passing by value

class PassbyValue{

public static void main(String args[]){ int num=3;

System.out.println(printData(num));

}

int printData(int n){ return n;

}

}

Ex:- passing by Object

class Car{

String name=”BMW”; void change(Car c){ c.name=”Baleno”;

}

public static void main(String args[]){ Car c = new Car();

System.out.println(“Before change: “+c.name); c.change( c);

System.out.println(“After change: “+c.name);

}}

O/P: Before change: BMW

After change: Baleno

* INHERITANCE
  + Inheritance is the process of taking the features (data members + methods) from one class to another class.
  + The class which is giving the features is known as base/Parent class.
  + The class which is taking the features is known as derived / child class.
  + The subclass inherits all of its instances variables and methods defined by the superclass and it also adds its own unique elements. Thus we can say that subclass is specialized version of superclass.

# Advantages:

1. Application development time is very less.
2. Redundancy (repetition) of the code is reducing. Hence we can get less memory cost and consistent results.
3. Instrument cost towards the project is reduced.
4. Reusability of code.
5. Code sharing.

# Syntax of Inheritance

class Subclass-Name extends Superclass-Name

{

//methods and fields

}

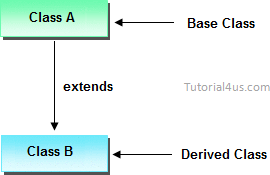
# Types of Inheritance

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

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. Hybrid inheritance

# Single inheritance

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



# Example of Single Inheritance

class Faculty

{

float salary=30000;

}

class Science extends Faculty

{

float bonous=2000;

public static void main(String args[])

{

Science obj=new Science(); System.out.println("Salary is:"+obj.salary); System.out.println("Bonous is:"+obj.bonous);

}

}

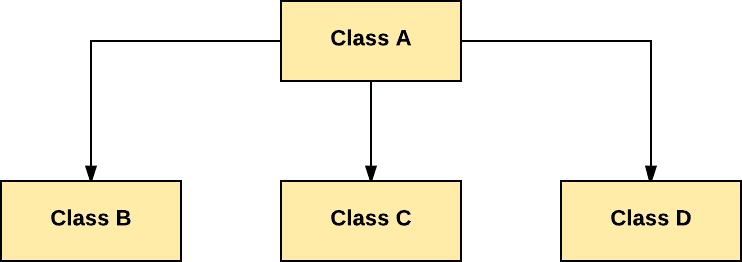
# Output

Salary is: 30000.0

Bonous is: 2000.0

# Hierarchical inheritance

Hierarchical Inheritance is one in which there exists single base class and n number of derived classes.



class A

{

public void methodA()

{

System.out.println("method of Class A");

}

}

class B extends A

{

public void methodB()

{

System.out.println("method of Class B");

}

}

class C extends A

{

public void methodC()

{

System.out.println("method of Class C");

}

}

class D extends A

{

public void methodD()

{

System.out.println("method of Class D");

}

}

class JavaExample

{

public static void main(String args[])

{

B obj1 = new B(); C obj2 = new C(); D obj3 = new D();

//All classes can access the method of class A obj1.methodA();

obj2.methodA(); obj3.methodA();

}

}

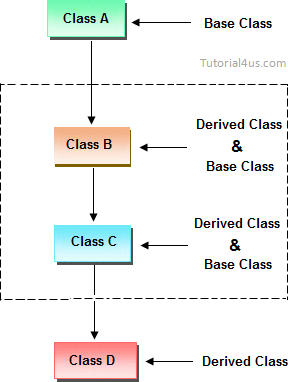
# Multilevel inheritances in Java

It is a ladder or hierarchy of single level inheritance. It means if Class A is extended by Class B and then further Class C extends Class B then the whole structure is termed as Multilevel Inheritance. Multiple classes are involved in inheritance, but one

class extends only one. The lowermost subclass can make use of all super classes‟ members.

# Single base class + single derived class + multiple intermediate base classes. Intermediate base classes

An intermediate base class is one in one context with access derived class and in another context same class access base class.



Hence all the above three inheritance types are supported by both classes and interfaces.

# Example of Multilevel Inheritance

class Faculty

{

float total\_sal=0, salary=30000;

}

class HRA extends Faculty

{

float hra=3000;

}

class DA extends HRA

{

float da=2000;

}

class Science extends DA

{

float bonous=2000;

public static void main(String args[])

{

Science obj=new Science(); obj.total\_sal=obj.salary+obj.hra+obj.da+obj.bonous; System.out.println("Total Salary is:"+obj.total\_sal);

}

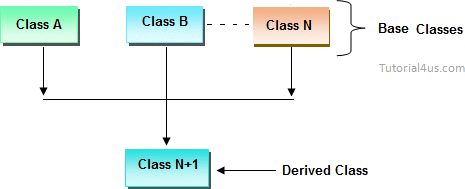
}

# Output

Total Salary is: 37000.0

# Multiple inheritance

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

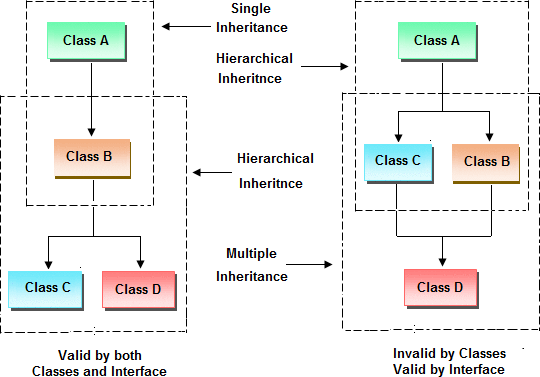


The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

# Hybrid inheritance

Combination of any inheritance type

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.



# Inheriting the feature from base class to derived class

In order to inherit the feature of base class into derived class we use the following syntax

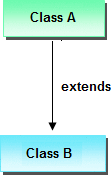
# Syntax

class ClassName-2 extends ClasssName-1

{

variable declaration; Method declaration;

}



# Explanation

* 1. ClassName-1 and ClassName-2 represents name of the base and derived classes respectively.
  2. extends is one of the keyword used for inheriting the features of base class into derived class it improves the functionality of derived class.

# Important Points for Inheritance:

* In java programming one derived class can extends only one base class because java programming does not support multiple inheritance through the concept of classes, but it can be supported through the concept of Interface.
* Whenever we develop any inheritance application first create an object of bottom most derived class but not for top most base class.
* When we create an object of bottom most derived class, first we get the memory space for the data members of top most base class, and then we get the memory space for data member of other bottom most derived class.
* Bottom most derived class contains logical appearance for the data members of all top most base classes.
* If we do not want to give the features of base class to the derived class then the definition of the base class must be preceded by final hence final base classes are not reusable or not inheritable.
* If we are do not want to give some of the features of base class to derived class than such features of base class must be as private hence private features of base class are not inheritable or accessible in derived class.
* Data members and methods of a base class can be inherited into the derived class but constructors of base class can not be inherited because every constructor of a class is made for initializing its own data members but not made for initializing the data members of other classes.

# Example of Inheritance

class Faculty

{

float salary=30000;

}

class Science extends Faculty

{

float bonous=2000;

public static void main(String args[])

{

Science obj=new Science(); System.out.println("Salary is:"+obj.salary); System.out.println("Bonous is:"+obj.bonous);

}

}

# Output

Salary is: 30000.0

Bonous is: 2000.0

# Why multiple inheritance is not supported in java?

Due to ambiguity problem java does not support multiple inheritance at class level.

# Example

class A

{

void disp()

{

System.out.println("Hello");

}

}

class B

{

void disp()

System.out.println("How are you ?");

}

}

class C extends A,B //suppose if it were

{

Public Static void main(String args[])

{

C obj=new C();

obj.disp();//Now which disp() method would be invoked?

}

}

In above code we call both class A and class B disp() method then it confusion which class method is call. So due to this ambiguity problem in java do not use multiple inheritance at class level, but it support at interface level.

# Final keyword in java

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level.

In java language final keyword can be used in following way.

1. Final at variable level
2. Final at method level
3. Final at class level

# Final at variable level

Final keyword is used to make a variable as a constant. This is similar to const in other language. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

# Final Keyword in java Example

public class Circle

{

public static final double PI=3.14159;

public static void main(String[] args)

{

System.out.println(PI);

}

}

# Final at method level

It makes a method final, meaning that sub classes can not override this method.

The compiler checks and gives an error if you try to override the method. When we want to restrict overriding, then make a method as a final.

# Example

public class A

{

public void fun1()

{

.......

}

public final void fun2()

{

.......

}

}

class B extends A

{

public void fun1()

{

.......

}

public void fun2()

{

// it gives an error because we can not override final method

}

}

# Example of final keyword at method level Example

class Employee

{

final void disp()

{

System.out.println("Hello Good Morning");

}

}

class Developer extends Employee

{

void disp()

{

System.out.println("How are you ?");

}

}

class FinalDemo

{

public static void main(String args[])

{

Developer obj=new Developer();

# Output

obj.disp();

}

}

It gives an error

# Final at class level

It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

# Example

public final class A

{

......

......

}

public class B extends A

{

// it gives an error, because we can not inherit final class

}

# Example of final keyword at class level Example

final class Employee

{

int salary=10000;

}

class Developer extends Employee

{

void show()

{

System.out.println("Hello Good Morning");

}

}

class FinalDemo

{

public static void main(String args[])

{

Developer obj=new Developer(); Developer obj=new Developer();

# Output

Output:

obj.show();

}

}

It gives an error

# Super keyword:

Super is a reference variable that is used to refer immediate parent class object. Uses of super keyword:

* 1. super is used to refer immediate parent class variable.
  2. super() is used to invoke immediate parent class constructors
  3. super is used to invoke immediate parent class method

# Example to call immediate Parent class variable using Super:

package javalab; class Inherit{

int i=10;

}

class Inherit2 extends Inherit{ int i=20;

void method(){

System.out.println("Value of a parent class is:"+***super.i***); System.out.println("Value of child class is: "+i);

}

}

public class InheritTest {

public static void main(String args[]){ Inherit2 obj=new Inherit2(); obj.method();

}

}

# Example to call immediate Parent class constructor using Super:

The super() keyword can be used to invoke the Parent class constructor. super() is added in each class constructor automatically by compiler. As default

constructor is provided by compiler automatically but it also adds super() for the first statement. If you are creating your own constructor and you don‟t have super() as the first statement, compiler will provide super() as the first statement of the constructor.

# Example:

package javalab; class Inherit{

Inherit(){ int i=10;

System.out.println("Base class constructor is invoked:"+i);

}

}

class Inherit2 extends Inherit{ int i=20;

Inherit2(){ super();

}

void method(){

//System.out.println("Value of a parent class is:"+super.i); System.out.println("Value of child class is: "+i);

}

}

public class InheritTest {

public static void main(String args[]){ Inherit2 obj=new Inherit2(); obj.method();

}

}

# Method Overriding in Java

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**. Here we will discuss about **Overriding in Java**. Without Inheritance method overriding is not possible.

# Advantage of Java Method Overriding

* Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method Overriding is used for Runtime Polymorphism

# Rules for Method Overriding

* method must have same name as in the parent class.
* method must have same parameter as in the parent class.
* must be IS-A relationship (inheritance).

# Example Method Overriding in Java

class Walking

{

void walk()

{

System.out.println("Man walking fastly");

}

}

class OverridingDemo

{

public static void main(String args[])

{

Man obj = new Man(); obj.walk();

}

}

# Output

Man walking

Problem is that I have to provide a specific implementation of walk() method in subclass that is why we use method overriding.

# Example of method overriding in Java

In this example, we have defined the walk method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

# Example

class Walking

{

void walk()

{

System.out.println("Man walking fastly");

}

}

class Man extends walking

{

void walk()

{

System.out.println("Man walking slowly");

}

}

# Output

class OverridingDemo

{

public static void main(String args[])

{

Man obj = new Man(); obj.walk();

}

}

Man walking slowly

# Accessing properties of base class with respect to derived class object

class A

{

int x; void f1()

{ x=10;

System.out.println(x);

}

void f4()

{

System.out.println("this is f4()"); System.out.println(" ");

}

}

class B extends A

{

int y; void f1()

{

int y=20; System.out.println(y);

System.out.println("this is f1()"); System.out.println(" ");

}

};

class C extends A

{

int z; void f1()

{ z=10;

System.out.println(z); System.out.println("this is f1()");

}

};

class Overide

{

public static void main(String[] args)

{

A a1=new B(); a1.f1();

a1.f4();

A c1=new C(); c1.f1();

c1.f4();

}

}

# Example of Implement overriding concept

class Person

{

String name;

void sleep(String name)

{

this.name=name;

System.out.println(this.name +"is sleeping+8hr/day");

}

void walk()

{

System.out.println("this is walk()"); System.out.println(" ");

}

};

class Student extends Person

{

void writExams()

{

System.out.println("only student write the exam");

}

void sleep(String name)

{

super.name=name;

System.out.println(super.name +"is sleeping 6hr/day"); System.out.println(" ");

}

};

class Developer extends Person

{

public void designProj()

{

System.out.println("Design the project");

}

void sleep(String name)

{

super.name=name;

System.out.println(super.name +"is sleeping 4hr/day"); System.out.println(" ");

}

};

class OverideDemo

{

public static void main(String[] args)

{

Student s1=new Student(); s1.writExams(); s1.sleep("student"); s1.walk();

Developer d1=new Developer(); d1.designProj(); d1.sleep("developer");

}}

# Difference between Overloading and Overriding

|  |  |  |
| --- | --- | --- |
|  | **Overloading** | **Overriding** |
| 1 | Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is  known as Overloading. | Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known  as Overriding. |
| 2 | Arguments of method must be different at  least arguments. | Argument of method must be same  including order. |
| 3 | Method signature must be different. | Method signature must be same. |
| 4 | Private, static and final methods can be  overloaded. | Private, static and final methods can  not be override. |
| 6 | Also known as compile time polymorphism or static polymorphism or  early binding. | Also known as run time polymorphism or dynamic polymorphism or late  binding. |
| 7 | Overloading can be exhibited both are  method and constructor level. | Overriding can be exhibited only at  method label. |
| 8 | The scope of overloading is within the  class. | The scope of Overriding is base class  and derived class. |
| 9 | Overloading can be done at both static  and non-static methods. | Overriding can be done only at non-  static method. |
| 10 | For overloading methods return type may  or may not be same. | For overriding method return type  should be same. |