* In Inheritance one class acquires the properties (methods and fields) of another.
* Using inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

**extends** is the keyword used to inherit the properties of a class.

**Examples:-** Calculator and ScientificCalculator

Calculation cal=new Inheritance\_Demo();

demo.addition(a, b);

demo.multipication(a, b);//wrong, using super class ref you cant call subclass methods

* Here, When an object is created for ScientificCalculator, its invokes the super class constructor and creates an object for the super class i.e Calculator.
* using the subclass reference variable you can access the properties of the subclass aswell as superclass.
* using the superclass reference variable you can access only superclass properties.

**Note:**

* A subclass inherits all the members (fields, methods, and nested classes) from its superclass.
* Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

**The super keyword:-**

The **super** keyword is similar to **this** keyword following are the scenarios where the super keyword is used.

* It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
* It is used to **invoke the superclass** constructor from subclass.

## 

## 

## Differentiating the members:-

* If a class is inheriting the properties of another class.
* And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

super.variable

super.method();

**SAMPLE CODE:-**

class Super\_class{

int num=20

//display method of superclass

public void display(){

System.out.println("This is the display method of superclass");

}

}

public class Sub\_class extends Super\_class {

int num=10;

//display method of sub class

public void display(){

System.out.println("This is the display method of subclass");

}

public void my\_method(){

//Instantiating subclass

Sub\_class sub=new Sub\_class();

//Invoking the display() method of sub class

sub.display(

//Invoking the display() method of superclass

super.display();

//printing the value of variable num of subclass

System.out.println("value of num in sub class:"+ sub.num);

//printing the value of variable num of superclass

System.out.println("value of num in super class:"+ super.num);

}

public static void main(String args[]){

Sub\_class obj = new Sub\_class();

obj.method();

}

}

**Invoking Superclass constructor**

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the super class.

But if you want to call a parametrized constructor of the super class, you need to use the super keyword as shown below.

super(values);

class Superclass{

int age;

Superclass(int age)

this.age=age;

}

public void getAge(){

System.out.println("The value of the age in super class is: " +age);

}

}

public class Subclass extends Superclass {

Subclass(int age){

super(age);

public static void main(String argd[])

Subclass s= new Subclass(24

s.getAge();

IS-A Relationship:

IS-A is a way of saying : This object is a type of that object.

Let us see how the **extends** keyword is used to achieve inheritance.

public class Animal{

}

public class Mammal extends Animal{

}

public class Reptile extends Animal{

}

public class Dog extends Mammal{

}

Now considering IS-A relationship, we can say :

* Mammal IS-A Animal
* Reptile IS-A Animal
* Dog IS-A Mammal
* Hence : Dog IS-A Animal as well

## Example:

public class Dog extends Mammal{

public static void main(String args[])

Animal a = new Animal();

Mammal m = new Mammal();

Dog d = new Dog();

System.out.println(m instanceof Animal);

System.out.println(d instanceof Mammal);

System.out.println(d instanceof Animal);

}

}

## The instanceof Keyword:

Let us use the **instanceof** operator to check determine whether Mammal is actually an Animal, and dog is actually an Animal

interface Animal{

class Mammal implements Animal{}

public class Dog extends Mammal{

public static void main(String args[]){

Mammal m = new Mammal();

Dog d = new Dog();

System.out.println(m instanceof Animal);

System.out.println(d instanceof Mammal);

System.out.println(d instanceof Animal);

}

}

This would produce the following result:

true  
true  
true

## HAS-A relationship:

* These relationships are mainly based on the usage. This determines whether a certain class **HAS\_A** certain thing.
* This relationship helps to reduce duplication of code as well as bugs.

**Lets us look into an example:**

public class Speed{}

public class Vehicle{}

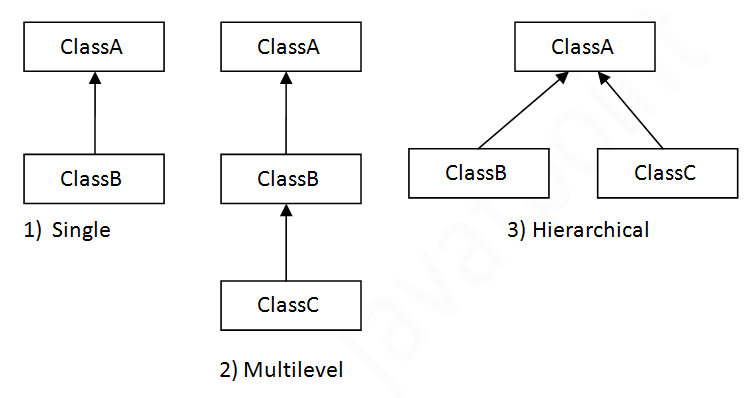
class Van extends Vehicle{

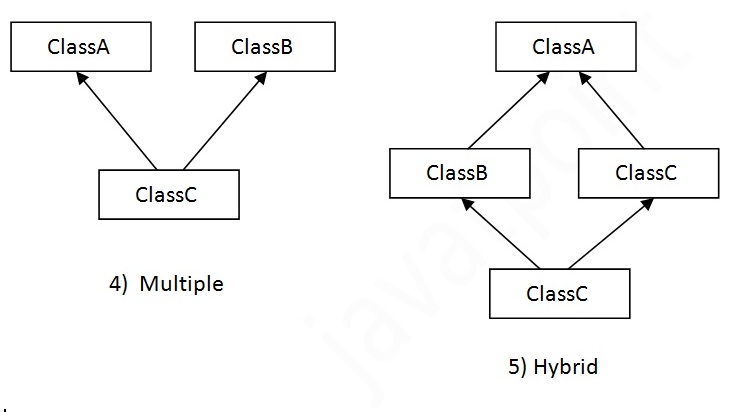
private Speed sp;

}

**This shows that class Van HAS-A Speed.**

* By having a separate class for Speed, we do not have to put the entire code that belongs to speed inside the Van class., which makes it possible to reuse the Speed class in multiple applications.
* In Object-Oriented feature, the users do not need to bother about which object is doing the real work.
* To achieve this, the Van class hides the implementation details from the users of the Van class.
* So basically what happens is the users would ask the Van class to do a certain action and the Van class will either do the work by itself or ask another class to perform the action.

**NOTE:** 



**NOTE:-**

* A very important fact to remember is that Java does not support multiple inheritance.
* This means that a class cannot extend more than one class.
* Therefore following is illegal:

**public class extends Animal, Mammal{}**

* However, a class can implement one or more interfaces.
* This has made Java get rid of the impossibility of multiple inheritance.