**Inheritance-**

* It is the mechanism that allows child classes to access the properties or behaviors of parent class.
* The process of creating the new class by using the existing class functionality called as Inheritance.
* Inheritance means simply reusability.
* Multiple classes can inherit from the same parent class.
* Inheriting classes can add features beyond those inherited from the parent class to allow for unique behavior.
* Inheritance represents IS-A relationship.
* Inheritance is implemented by using extends keywords.

Advantages of Inheritance:

• The main advantage of it is code reusability:

Inheritance allows you to reuse the features of an existing class an unlimited number of times across any class that inherits that class.

You can keep consistent functionality across all objects of the same type without rewriting the code.

Example- IS Relationship

Class Policy {

}

Class TermPolicy extends Policy {

}

In this example,

Policy is super class/parent class and

TermPolicy is sub class/child class

Where TermPolicy **IS A** Policy.

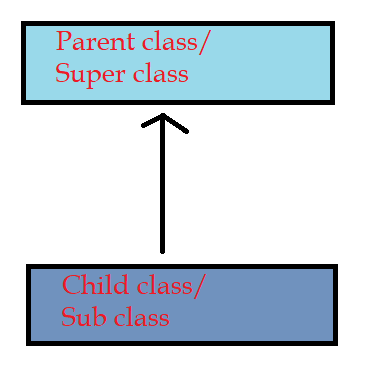
**Note-**

All the parent members are derived into child class but they are depends upon the below

-To check the access specifiers

-Members does not exist into sub class.

**UML Diagram-**



Super Class->Parent Class->Base Class-> Old Class

Sub Class-> Child Class-> Derived Class -> New Class

**Note**-

1. A child class cannot extends to multiple parent classes.
2. Inherit the classes by using extends keywords.
3. Whenever we create the object of subclass then all the member will get called super class as well as sub class.
4. Why we use inheritance that is for code reusability, reusability means we can reuse existing class features such as variables and method, etc.
5. We cannot extend the final class.

**When to use?**

If we want to extends or increase of features of class then go for inheritance.

**Business requirement-**

**Why inheritance?**

**Example 1**

Consider, there are two classes one for child and one for parent. The properties like surname, and father name will be same for every child. Also writing these properties again and again for every child is not a good practice because it is the case of duplicate code.

So, by using inheritance we can solve this problem. As when inheritance is a mechanism where child classes can access properties of parent classes so these properties like fatherName and Surname will be available for every child class which inherited the parent class.

package com.velocity;

public class Parent {

private String fatherName;

private String surname;

public String getFatherName() {

return fatherName;

}

public void setFatherName(String fatherName) {

this.fatherName = fatherName;

}

public String getSurname() {

return surname;

}

public void setSurname(String surname) {

this.surname = surname;

}

}

package com.velocity;

public class Child extends Parent {

private String name;

private String bike;

private String mobile;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getBike() {

return bike;

}

public void setBike(String bike) {

this.bike = bike;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

}

**Example 2**

Suppose we have one class which contain the fields like, firstname, lastname, address, city, mobile number and

In future we got the requirement to add the PAN number then what option we have below-

1. Modify the attributes/fields in existing class but this is not good option it will increase the testing for that class.
2. Add the attributes in the new class, in this the good option we can also reduce the testing efforts for this.

**How the class will looks like**

Class Parent {

String firstname;

String lastname;

String address;

String city;

String mobilenumber;

}

Class Child extends Parent {

String pancard;

}

**Note**

We cannot assign parent class reference to child class-

All the members of super class will be directly inherited into sub class and they are eligible and depends on access specifiers only.

**Dynamic dispatch-**

The process of assigning the child class reference to parent class called as “Dynamic dispatch.”

**Example-**

Class X {

}

Class Y extends X {

}

Class Test {

Public static void main(string args[]){

X x= new Y(); // Here we are assigning the child reference new Y() to parent class .

}

Inheritance Example-

**Scenario 1**

**package** com.inheritance;

**class** X {

**int** a = 10;

**int** b = 20;

**void** m1() {

System.***out***.println("Class X- m1() method");

}

**void** m2() {

System.***out***.println("Class X- m2() method");

}

}

**package** com.inheritance;

**class** Y **extends** X {

**int** b = 30;

**int** c = 40;

**void** m2() {

System.***out***.println("Class Y- m2() method");

}

**void** m3() {

System.***out***.println("Class Y- m3() method");

}

}

**package** com.inheritance;

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

//Scenario- 1

X x=**new** X();

System.***out***.println(x.a);

System.***out***.println(x.b);

System.out.println(x.c);

x.m1();

x.m2();

x.m3();

//Scenario-2

Y y = **new** Y();

System.***out***.println(y.a);

System.***out***.println(y.b);

System.***out***.println(y.c);

y.m1();

y.m2();

y.m3();

//Scenario-3

X x = **new** Y();

System.***out***.println(x.a);

System.***out***.println(x.b);

//System.out.println(x.c);

x.m1();

x.m2();

//x.m3();

//Scenario-4 (Note 3rd and 4th scenario are same)

X x = **new** X();

Y y = **new** Y();

x = y;

System.***out***.println(x.a);

System.***out***.println(x.b);

System.out.println(x.c);

x.m1();

x.m2();

x.m3();

//Scenario-5- Note- this is equivalent to 2nd scenario

X x = **new** Y();

Y y = **new** Y();

y = (Y) x;

System.***out***.println(y.a);

System.***out***.println(y.b);

System.***out***.println(y.c);

y.m1();

y.m2();

y.m3();

//Scenario-6

Y y= new X();

}

}