**Inheritance**

**“**Inheriting the members of parent class to child class **except private, final and static members** is called inheritance**”.**

The parent is also called as super class and the child is also called as sub class.

Super class- when object is created for super class, only non static members will be loaded into an object.

Sub class-when object is created for sub class first parent non static members will be loaded and then child non static members will be loaded.

Constructors are not members, so they are not inherited by sub class,

The inheritance can be achieved by using keyword **extends**. The sub class should extend the super class.

With the reference of sub class object we can access both the inherited member as well as sub class members.

**Types of java**

1. **Single level inheritance** – one super class, one sub class

**PARENT**

**CHILD**

1. **Multi level inheritance**- one super class followed by sub classes at each level

**GRAND PARENT**

**CHILD**

**PARENT**

**CHILD**

1. **Hierarchical inheritance:** one super class followed by multiple children

**GRAND PARENT**

**SON1**

**SON1**

1. **Hybrid inheritance:** combination of 2 or more inheritance

**CHILD**

**SON1**

**SON1**

**GRAND PARENT**

**CHILD**

1. **Multiple inheritances**: multiple parents and one child are not allowed in java through classes.

Example

**class** A{

**private** **int** k=123;

**int** i=100;

**static** **int** *j*=200;

**void** print(){

System.***out***.println("Print method");

}

**static** **void** display(){

System.***out***.println("Display method");

}

}

**class** B **extends** A{

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

System.***out***.println(A.*j*);

A.*display*();

System.***out***.println(B.*j*); //possible

B.*display*(); //possible

A a1 = **new** A();

System.***out***.println(a1.i);

a1.print();

B b1 = **new** B();

System.***out***.println(b1.i);

b1.print();

//System.out.println(a1.k);

//System.out.println(b1.k);

}

}

**Uses of inheritance – to optimize code and reusability code.**

**Super**

* **It’s a keyword used to refer immediate parent class object**
* **Use to refer immediate parent class instance variable**
* **Use to invoke parent class methods**
* **Use to invoke parent class constructor**

**super() :**During inheritance, when a child does not call parent constructor, compiler automatically calls **super()** constructor call.

super()- default constructor, parameter less constructor

**this**

this is a keyword in java where it works as reference to current class object.

The keyword this may be used only in the following contexts:

* this keyword with field(Instance Variable)
* this keyword with Method

**this keyword with field**

this keyword can be very useful in the **handling of Variable Hiding**. We cannot create two local variables with the same name. However it is legal to create one global variable & one local variable or Method parameter with the same name. In this scenario the local variable will hide the global variable this is called **Variable Hiding**.

Example

public class ThisVariables {

int globalVariable = 20;

void variableHide(int globalVariable){

ThisVariables tv2 = new ThisVariables();

globalVariable = globalVariable;

System.*out*.println(tv2.globalVariable);

}

public static void main(String[] args) {

ThisVariables tv1 = new ThisVariables();

tv1.variableHide(30);

System.*out*.println(tv1.globalVariable);

}

}

**this keyword with Method**

**this** keyword can also be used inside **Methods to call another Method from same Class**.

Example

**public** **class** ThisAsMethod {

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

Demo1 obj = **new** Demo1();

obj.i = 10;

obj.method();

}

}

**class** Demo1 **extends** ThisAsMethod {

**int** i;

**void** method() {

this.method1();

}

**void** method1() {

System.***out***.println(i);

}

}