**Inheritance**

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

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

Constructors are not members, so they are not inherited by sub class, but the constructor of the super class can be invoked from the 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.

Multilevel inheritance is supported in java.

Multiple inheritances 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** **class** Demo1{

**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);

}

}

**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 Constructor
* this keyword with Method

**this keyword with field (Instance/Non-static)**

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 (Instance/Non-static) global variable & one local variable or Method parameter with the same name. In this scenario the local variable will hide the (Instance/Non-static) 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() {

method1(**this**);

}

**void** method1(Demo1 t) {

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

}

}

**this keyword with Constructor**

**“this”** keyword can be used inside the constructor to call another overloaded constructor in the same Class. This is called the Explicit Constructor Invocation. This occurs if a Class has two overloaded constructors, one without argument and another with argument. Then the **“this”** keyword can be used to call constructor with argument from the constructor without argument. this is required as the constructor cannot be called explicitly.

Example

**class** Demo {

Demo(**int** i){

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

}

Demo(){

**this**(10);

System.***out***.println("No argument Demo class Constructor");

}

}

**class** ThisConstructor **extends** Demo{

ThisConstructor(){

**super**();

System.***out***.println("No argument ThisConstructor class Constructor");

}

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

ThisConstructor TC = **new** ThisConstructor();

System.***out***.println("Main Method");

}

}

**Note:**

**this** keyword can only be the first statement in Constructor.

A constructor can have either this or super keyword but not both.

What will the compiler do?

class X{

}

class Y extends X{

}

Compiler does the following

class X{

X(){

}

}

class Y extends X{

Y(){

super();

}

}

Example

class G{

}

class H extends G{

H(int a, int b){

super();//this will be put by compiler by default

}

}

Since super class does not have a constructor. It will first create a default constructor in the super class and in sub class uses super();

class X{

X(String str){

}

}

class Y extends X{

Y(){

super();

}

}

**class Y extends X{**

**Y(){**

**super(“Hello”);**

**}**

**}**

**Note:**

Every class we create in java will be sub class to the Object class. Object class is the super most class in java. If any class is not extending super class by default will always extend Object class

While inheriting object from super class, chain of constructors will be involved. The constructor of a sub class calls constructor of the super class. The constructor of the super class in turn calls constructor of its super class. This is known as chain of constructors.

The java compiler makes an implicit call to the super class constructor by using default constructor of super class or when no argument constructor is present.

If default constructor /no argument constructor is not available ,then the constructor should be called explicitly using super statement along with argument(s) as expected.

The super statement should be always the first statement in the constructor

Using super statement we can call any of the super class constructor.

Whenever the java compiler makes a call to the super class, when super is not mentioned it uses super statement implicitly.

Example

**package** inheritance;

**class** C{

**int** i,j;

C(**int** i,**int** j){

**this**.i = i;

**this**.j = j;

}

**void** print(){

System.***out***.println("i = "+ i + "\nj = "+j);

}

}

**class** D **extends** C{

**int** x,y;

D(**int** x,**int** y){

**super**(x,y);

**this**.x=x;

**this**.y=y;

}

**void** display(){

System.***out***.println("x = "+ x +"\ny = "+y);

}

}

**class** E **extends** D{

**int** a,b;

E(**int** a,**int** b){

**super**(a,b);

**this**.a=a;

**this**.b=b;

}

**void** show(){

System.***out***.println("a = "+ a +"\nb = "+b);

}

}

**public** **class** Demo {

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

E e1 = **new** E(100,200);

e1.print();

e1.display();

e1.show();

}

}

Output:

i = 100

j = 200

x = 100

y = 200

a = 100

b = 200

**Object count:**

**class** Super{

**static** **int** *counter*;

Super(){

*counter*++;

}

}

**class** Sub **extends** Super{

**static** **int** *counter*;

Sub(){

*counter*++;

}

}

**public** **class** ObjectCount {

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

Super a1 = **new** Super();

**new** Super();

**new** Super();

System.***out***.println(Super.*counter*);

Sub b1 = **new** Sub();

**new** Sub();

**new** Sub();

System.***out***.println(Sub.*counter*);

System.***out***.println(Super.*counter*);

}

}

Output: 3

3

6

**Why we need to create our own constructor**

If we need to create an object of a class based on specific criteria then we need to create our own constructor

Example (Assignment)

If we need to create object of student class which takes name and student mail-id then we can create our own constructors.