**Function**

**Virtual Function**

**Method Overriding.**

**It is a OOPS concept, so supported in both C++ and Java.**

**Giving new implementation of base class method into derived class and the calling of this new implemented function with derived class's object is called Function Overriding.**

**Java Program-1:**

class **Base**

{

public void show()

{

System.out.println("This Base Class method");

}

}

class **Derived** extends **Base**

{

public void show()

{

System.out.println("This Derived Class method");

}

}

public class **MyClass** {

public static void main(String args[])

{

Base b= new Base();

B.show() ;

Derived d=new Derived();

d.show();}}

output: This Base Class method

This Derived Class method

**C++ Program-1:**

#include<iostream>

using namespace std;

class base

{

public: void show()

{cout<< "This non-virtual function of base class" <<endl; }

};

class derived:public base

{

public: void show()

{cout<< "This non-virtual function od derived class" <<endl; }

};

int main()

{base b1;

b1.show();

derived d;

d.show();}

**Output**: This non-virtual function of base class

This non-virtual function of derived class

Now upto this, we just call base class method using base class object and call Derived class method using derived class object.

But Now we want to call derived class method that was implemented, using base class object.

Now we want to call /access derived class method using Pointer of of base class(in C++ we use pointer of base class),in Java we use base class reference to call derived class method.

**Main thing** is that , In C++,class method is non-virtual by default, so make them virtual first before calling it using base class object, then create pointer of base class and address of derived class to it and call,then it will call derived class method.

(The base class object must be of pointer type so that we can dynamically replace the address of base class function with derived class function. This is how we can achieve "Runtime Polymorphism".)

In Java class method is virtual by default, so need to make base class method as virtual, we can directly assign object of derived to reference of base call and call derived class method.

**Giving new implementation of derived class method into base class and the calling of this new implemented function with base class's object is done by making base class function as virtual function( In c++, method is non-virtual so make them virtual first ,In java method is virtual by default so no need to make virtual in base class)**

**Virtual function is used in situation, when we need to invoke derived class function using base class pointer.**

If we doesn't use virtual keyword in base class, base class pointer will always execute function defined in base class.

**C++ Program-1:**

#include<iostream>

using namespace std;

class base

{public:

void print ()

{ cout<< "Non virtaul function of base class" <<endl; }

virtual void show()

{cout<< "This virtual function of base class" <<endl; }

};

class derived:public base

{

public:

void print ()

{ cout<< "Non virtaul function " <<endl; }

void show()

{cout<< "This virtual function of derived class" <<endl; }

};

int main()

{base b1;

b1.show();

derived d1;

d1.show();

base \*b2;

derived d2;

b2=&d2;

b2->show();

b1.print();// Non virtual function.

}

**Ouput:**This virtual function of base class

This virtual function of derived class

This virtual function of derived class

Non virtaul function of base class.

**Java Program-2**

class Base

{

public void show()// if declare "Final void show();" then compile time error occur,bz final method can not be ovrride.

{

System.out.println("This Base Class method");

}

}

class Derived extends Base

{

public void show()

{

System.out.println("This Derived Class method");

}

}

public class MyClass {

public static void main(String args[])

{

Derived d=new Derived();

d.show();

Base b=new Base();

b.show();

Base b1=new Derived ();

b1.show();

}

**Ouput:** **This Derived Class method**

**This Base Class method**

**This Derived Class method**