**Write a brief note on the other use of interfaces.**

We can use an interface as a reference data type. If we define a reference variable of interface type, then the object that is assigned to it must be an instance of the class that implements the interface. With this we can provide limited access to the data members of the object.

This concept is implemented in the below example. In this example, the class “Materials” implements multiple interfaces. When an object of Materials class is created, methods of all the interfaces are available, but by assigning the address of Materials class object to the reference variable of interface type, only the members of the that interface are accessible through that reference variable.

**Program**

interface BasicJavaMaterials{

void getBasicJavaMaterials();

}

interface CoreJavaMaterials{

void getCoreJavaMaterials();

}

interface JavaMaterials extends BasicJavaMaterials, CoreJavaMaterials{

void getAdvancedJavaMaterials();

}

interface AndroidMaterials{

void getAndroidMaterials();

}

interface WebFrameworksMaterials{

void getWebFrameworksMaterials();

}

**class Materials** implements JavaMaterials, AndroidMaterials, WebFrameworksMaterials {

public void getBasicJavaMaterials(){

System.out.println("Basic Java Materials access");

}

public void getCoreJavaMaterials(){

System.out.println("Core Java Materials access");

}

public void getAdvancedJavaMaterials(){

System.out.println("Advanced Java Materials access");

}

public void getAndroidMaterials(){

System.out.println("Android Materials access");

}

public void getWebFrameworksMaterials(){

System.out.println("Web Frameworks Materials access");

}

}

**class Test**{

public static void main(String[] args){

JavaMaterials jm1 = new Materials();

jm1.getBasicJavaMaterials();

jm1.getCoreJavaMaterials();

jm1.getAdvancedJavaMaterials();

}

}

