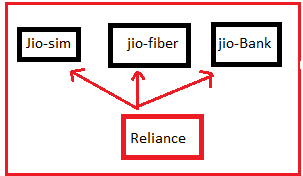
Note:- **The** **Java does not support multiple inheritance with classes**.

Example:



class jiosim{}

class jiobank{}

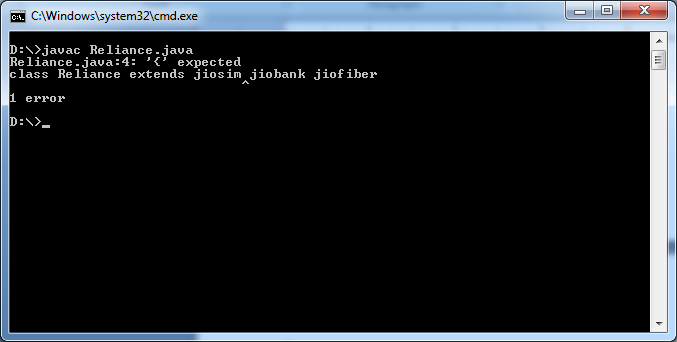
class jiofiber{}

class Reliance extends jiosim jiobank jiofiber

{

}

Output:-



This problem has been overcame by Interface**. In java , multiple inheritance can be achieved through the “Interface”.**

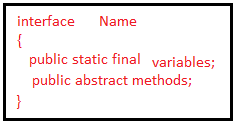
Def1: Interface is also one type class.

Def2: It is fully un-implemented class.

Def3: It is pure abstract class because it contains only abstract methods.

Def4: It is specification/ agreement document between object user and object implementer.

Syntax:



The Interface contains

Public static final variables

Public abstract methods.

1. Interface does not allow private and protected members.

2. We can also create empty interface.

3. Interface can not be declared as final.

4.Interface can not be initialized, but its reference variable can be created for storing its subclass objects references to develop loosely coupled user application.

Examples:-1

interface Reliance

{

int a=10;

public void display();

}

Output:-No CE.

2.

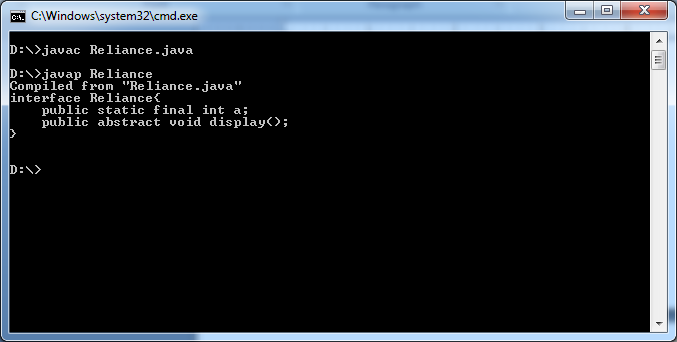
interface Reliance

{

int a=10;

void display();

}

****

* Interface can have only static final variables, even if we create variable as non-static, non-final variable, compiler converts them as static and final variable.
* Even if we create a methods as non-public and non-abstract, compiler converts them as public and abstract methods.
* Interface does not has super class or super interface.

Example3:-

interface Reliance

{

private int a=10;

void display();

}



The interface does not allow private or protected members.

Example4:

interface jio

{

int a=10;

void display();

}

class Reliance implements jio

{

public void display(){}

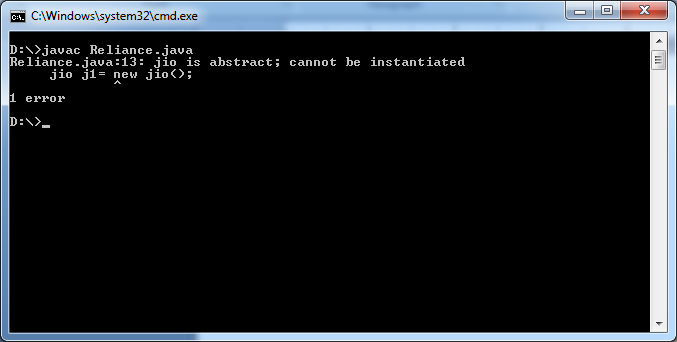
public static void main(String arg[])

{

jio j1= new jio();

}

}



Example5:

interface jio

{

int a=10;

void display();

}

class Reliance implements jio

{

public void display(){System.out.println("Subclass");}

public static void main(String arg[])

{

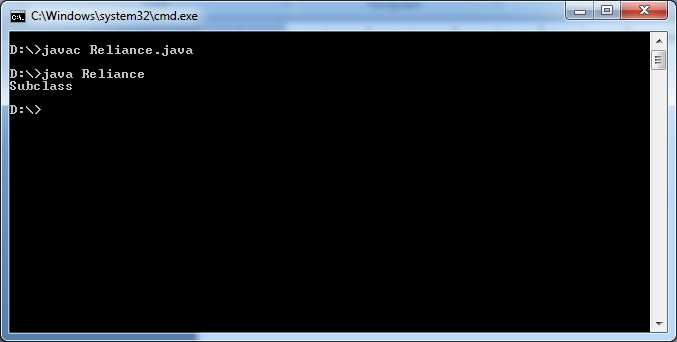
jio j1= new Reliance();

j1.display();

}

}

Output:-



The class which should provides a body to all abstract methods of interface is called implementation class of interface.

Example:-

interface Shape

{

public abstract void area();

public abstract void perimeter();

}

class Rectangle implements Shape{

public void area()

{

System.out.println("Area of Rectangle formula: l\*b");

}

public void perimeter(){

System.out.println("Perimeter of Rectangle formula:2\*(l+b)");

}

}

class Square implements Shape{

public void area()

{

System.out.println("Area of Square formula: s\*S");

}

public void perimeter()

{

System.out.println("Perimeter of Square Formula:4\*s");

}

}

class Reliance

{

public static void main(String arg[])

{

Rectangle r1=new Rectangle();

Square s1=new Square();

r1.area();

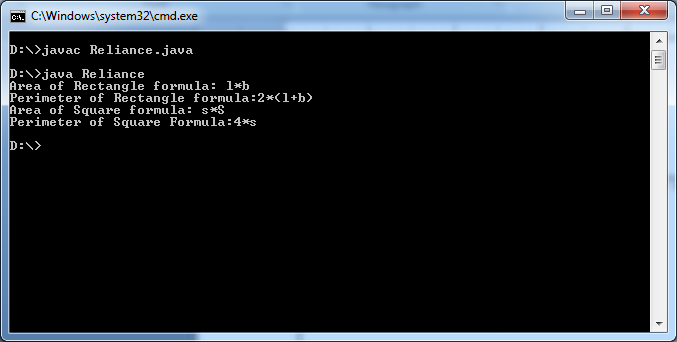
r1.perimeter();

s1.area();

s1.perimeter();

}

}



Note:- If we don’t want to implement all abstract methods of interface in subclass then we should declare subclass as abstract class otherwise we get CE.

interface Shape

{

public abstract void area();

public abstract void perimeter();

}

class Reliance implements Shape

{

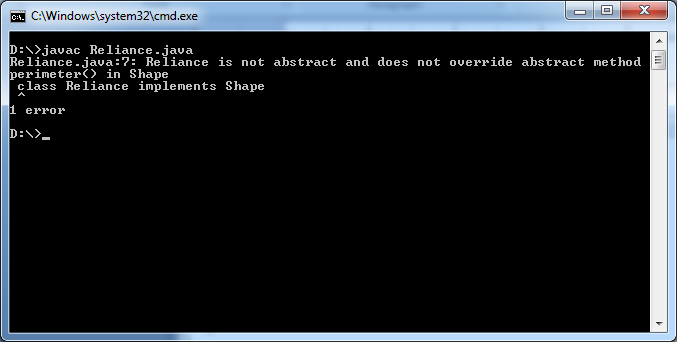
public void area(){}

public static void main(String arg[])

{

}

}



interface Shape

{

public abstract void area();

public abstract void perimeter();

}

abstract class Reliance implements Shape

{

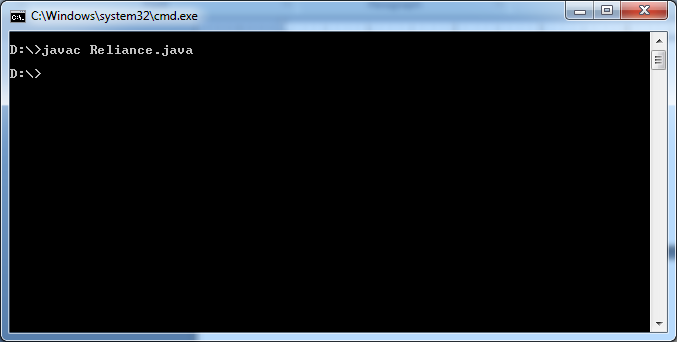
public void area(){}

public static void main(String arg[])

{

}

}



**1.Java8 Features Related to Interface:**

**1.1.Default Method:-**java8 onwards, The programmer can declare **default methods** in interface along with public static final variables and abstract methods. Default methods are also known as **“Defender Methods”**. The default method is concrete method but not abstract method.

Example:-

Public interface one{

Public default void m1(){

System.out.println(“First Default Method”);

}

}

Note:- The default methods are visible in implementation class. Based on implementation class, the default method can be overridden or default method can be used directly.

**1.1.1.Default Method with multiple inheritance:**

**Example:**

Interface A{

Public default void m1()

{

System.out.println(“Class A Default Method”);

}

}

Interface B{

Public default void m1()

{

System.out.println(“Class B Default Method”);

}

}

Class C implements A,B{

Public static void main(String args[]){

C c1=new C();

C1.m1();

}}

In above example, c1.m1() leads to Error. JVM will get confusion about which m1() method to be executed.

Solution:- The solution is providing the new implementation to default method in implementation class.

Class C implements A,B{

Public void m1(){

System.out.println(“ Implementation provided to default method”);

}

Public static void main(String args[]){

C c1=new C();

C1.m1();

}}

**1.2.Static Methods:-** Java8 onwards, the java developer can write the static methods inside interface along with public static final variables,abstract mehods and default methods. The static methods are concrete methods but not abstract methods.  **By default,these methods are not visible in implementation class of interface. The static method is not accessed by implementation class object. It is only accessed with interface name.**

**Example:**

Interface A{

Public static void m1(){

System.out.println(“ Static method in interface”);

}

}

Class B implements A{

Public static void main(String args[]){

B b1=new B();

// B1.m1(); It leads to Error.

A.m1();

}

}

Note: - Based on Subclass requirement, we can define exactly same method in implementation class. It is valid but not overriding.

Interface A{

Public static void m1(){

System.out.println(“ Static method in interface”);

}

}

Class B implements A{

Public static void m1(){

System.out.println(“ Static method in interface has been overridden in implementation class.”);

}

Public static void main(String args[]){

B b1=new B();

B1.m1();

A.m1();

}

}

**1.3. Java 1.8 onwards , we can write main() method inside interface and hence we can run interface directly from the command prompt.**

**Example:**

Interface A{

Public static void main(String arg{])

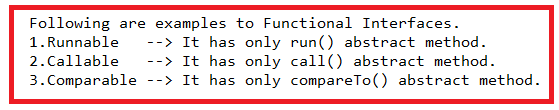
{

System.out.println(“Rock”);

}

}

**1.4.Functional Interface:-** The Interface which has exactly one abstract method is classed Functional Interface. The Functional interface can have multiple default methods and static methods.



Example:1

Interface I1{

Public void m1();

}

Example:-2

Interface I1{

Public void m1();

Public default void display(){}

Public static void display1(){}

}

**2. Java9 Features Related to Interface:-**

**2.1. Private Methods:- java9 onwards,** Interface allows private methods also. These private methods inside interface should have body. Privte methods inside the interface can be **either static or non-static**. The private method is not accessible in implementation class of interface.

Advantage:

------------

1. Code Reusability.

**2.2.Need of Private Method:-**

**interface** eatable {

**public** **default** **void** m1() {

System.***out***.println("Peel off the fruit");

System.***out***.println("m1 method");

}

**public** **default** **void** m2() {

System.***out***.println("Peel off the fruit");

System.***out***.println("m2 method");

}

}

All default methods have common logic. By common logic, code length increased. We separate the common logic into private method and The private method is called in default methods which requires that functionality. By doing like this, code length will decrease.

**interface** eatable {

**private** **void** m3() {

System.***out***.println("Peel off the fruit");

}

**public** **default** **void** m1() {

System.***out***.println("m1 method");

M3();

}

**public** **default** **void** m2() {

System.***out***.println("m2 method");

M3();

}

}

Example:1

**interface** eatable {

**private** **static** **void** m3() {

System.***out***.println("Peel off the fruit");

}

**private** **void** m4() {

System.***out***.println("eat fruit");

}

**public** **default** **void** m1() {

System.***out***.println("m1 method");

*m3*();

m4();

}

**public** **default** **void** m2() {

System.***out***.println("m2 method");

*m3*();

m4();

}

}

**class** suku **implements** eatable{

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

{

suku s=**new** suku();

s.m1();

s.m2();

}

}

Output:-

---------

m1 method

Peel off the fruit

eat fruit

m2 method

Peel off the fruit

eat fruit