Multiple Inheritance using Interface :

In a class we have a constructor so, it is providing ambiguity issue but inside an interface we don't have constructor so multiple inheritance is possible using interface as shown in the program below.

The Implementer class constructor's super keyword will directly move to Object class constructor.

```
//Program :
package com.rani.mi;

interface Alpha
{
    void m1();
}

interface Beta
{
    void m1();
}

class Implementor implements Alpha, Beta
{
    @Override
    public void m1()
    {
        IO.println("Multiple Inheritance is Possible");
    }
}
public class MultipleInheritanceDemo
{
    public static void main(String[] args)
    {
        new Implementor().m1();
    }
}
```

Extending an interface :

One interface can extend another interface but it can't implement an interface as shown in the program.

```
package com.rani.interface_demo;

interface A
{
    void m1();
}

interface B extends A
{
    void m1();
}

class Implementor implements B
{
    @Override
    public void m1()
    {
        IO.println("m1 method is overridden");
    }

    @Override
    public void m1()
    {
        IO.println("m1 method is overridden");
    }
}

public class ExtendingInterface {
    public static void main(String[] args)
    {
        Implementor i = new Implementor();
        i.m1();
        i.m1();
    }
}
```

Java 8 Features :

Interface from JDK 1.8V

What is the limitation of abstract method in project maintenance :

The major maintenance problem with interface was, if we add any new abstract method at the later stage of development, then in existing interface has no idea about this new abstract method so that abstract method declaration will create problem in existing implementation class as the abstract class is unable to create object.

We need to provide implementation for all the abstract methods available inside an interface whether it is required or not?

To avoid this maintenance problem Java software people introduced default method inside an interface.

What is default method :

It is possible to write default method (default keyword + method body) inside an interface from JDK 1.8V.

A default method we can declare only inside an interface but not in a class.

A default method must be declared with default keyword and contains method body.

By default access modifier of a default method is public.

A default method cannot be marked as abstract, final or static.

A default method may be overridden by a class that implements the interface.(Without any boundaries)

It is mainly used to provide "DEFAULT IMPLEMENTATION"

```
//Program :
package com.rani.java_8_Features;

public interface Vehicle
{
    void run();
    void horn();

    default void digitalMeter() //JDK 1.8 [Access modifier is public]
    {
        IO.println("Default Implementation");
    }
}

package com.rani.java_8_Features;

public class Car implements Vehicle
{
    @Override
    public void run()
    {
        IO.println("Car is running");
    }

    @Override
    public void horn()
    {
        IO.println("HM HM");
    }

    @Override
    public void digitalMeter() //JDK 1.8
    {
        IO.println("Car is having digital meter facility");
    }

}

package com.rani.java_8_Features;

public class Bike implements Vehicle
{
    @Override
    public void run()
    {
        IO.println("Bike is running");
    }

    @Override
    public void horn()
    {
        IO.println("PEEP PEEP");
    }

}

package com.rani.java_8_Features;

public class DefaultMethodDemo
{
    public static void main(String[] args)
    {
        Vehicle vehicle = null;

        vehicle = new Car();  vehicle.run();  vehicle.horn();  vehicle.digitalMeter();
        vehicle = new Bike();  vehicle.run();  vehicle.horn();
    }
}
```

What was approach to handle this abstract method situation before Java 8 [1996 to 2014] :

```
interface Vehicle                          interface Battery extends Vehicle
{                                          {
    void run();                                void battery();
}                                          }
class Car implements Vehicle               class BatteryCar extends Car implements Battery
{                                          {
    @Override                                  @Override
    public void run()                          public void battery()
    {                                          {
    }                                          }
}                                          }
```

Note : From JDK 1.8V, Java software people added so many default methods inside an existing interface to provide backward compatibility support

Multiple Inheritance using default method :

Multiple inheritance is possible in java by using default method also inside an interface, here we need to use super keyword to differentiate the super interface methods.

Before java 1.8, we have abstract method inside an interface but now we can write method body(default method) inside an interface. Abstract method inside an interface we need to take super keyword with interface name(Alpha.super m1())

Whenever a class is implementing from two OR more than two super interface and if the interfaces are containing same default method name then we have an ambiguity issue so, compulsory we need to override the method.

```
package com.rani.md_using_default_method;

interface Alpha
{
    default void m1()
    {
        IO.println("m1 default method of Alpha interface");
    }
}

interface Beta
{
    default void m1()
    {
        IO.println("m1 default method of Beta interface");
    }
}

class Implementor implements Alpha,Beta
{
    @Override
    public void m1()
    {
        Beta.super.m1();
        Alpha.  super.m1();
        IO.println("MI is possible by using default method");
    }
}

public class MIUsingDefaultMethod {

    public static void main(String[] args)
    {
        Implementor i = new Implementor();
        i.m1();
    }

}
```