Interface as a return type of the Method :

* As we know we can take method return type as a class which is known as Factor Method.

Example :

```
public class Test
{
   public Test accept()
   {
      return new Test();   // OR  return null     OR   return this;
   }
}
```

* If we take interface as a return type of the method then we will get more flexibility to return multiple values.

Example :

```
public HotDrink accept()
{
    return new Tea();   //OR  return new Coffee();     OR  return new Boost();
}
```

Working with interface using Anonymous inner class :

```
package com.ravi.interface_demo;

interface Student
{
    void writeExam();
}

public class InterfaceAnonymous
{
    public static void main(String[] args)
    {
          Student placement = new Student()
          {
           @Override
           public void writeExam()
           {
               System.out.println("Placement batch students are writing exam");

           }

          };

          Student regular = new Student()
          {
           @Override
           public void writeExam()
           {
               System.out.println("Regular batch students are writing exam");

           }

          };

          placement.writeExam();
          regular.writeExam();

    }

}
```

Multiple Inheritance by using interface :

* The main purpose of interface to provide Loose coupling and Dynamic polymorphism.

* It does not support constructor so, We can achive multiple inheritance.

Example :

```
interface A                         interface B
{                                   {
     void m1();                         void m1();
}                                   }

            class C extends Object implements A, B
            {
                    C()
                    {
                       super();          --> Object()
                    }                        {
            }                                }
```

Program :

```
package com.ravi.mi;

interface Alpha
{
    void m1();
}

interface Beta
{
    void m1();
}

class Implementer implements Alpha, Beta
{

    @Override
    public void m1()
    {
        System.out.println("Multiple Inheritance We can achieve using interface");
    }
}
public class MultipleInheritance
{
    public static void main(String[] args)
    {
        Implementer i = new Implementer();
        i.m1();

    }

}
```

Extending interface :

* It is possible that one interface can extend another interface as shown in the program.

```
package com.ravi.interface_demo;

interface A
{
    void m1();
}

interface B extends A
{
    void m2();
}

class MyClass implements B
{

    @Override
    public void m1()
    {
        System.out.println("M1 method Overridden");
    }

    @Override
    public void m2()
    {
        System.out.println("M2 method Overridden");
    }
}
public class InterfaceDemo
{
    public static void main(String[] args)
    {
        MyClass m = new MyClass();
        m.m1();
        m.m2();
    }
}
```

JAVA 8 features :

* Java 8V was introduced by Oracle Corporation in March 2014.
* It is a very popular feature in the industry because by using Java 8v features we can mainly write concise coding.

Problem with adding new abstract method in an existing interface :

* Whenever we add any new abstract method inside an existing interface then overriding of this method (implementation of this abstract method) is compulsory in all the implementer classes.

* It is one kind of boundation because overriding is compulsory otherwise the implementer class will become abstract class.

* IT industry was suffering from this problem, Java software people (before Java 8V) had provided a solution for this problem which is as follows :

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

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 boundation]

//Program :

```
package com.ravi.java_8_features;

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

    //Default implementation
    default void digitalMeter() //JDK 1.8
    {
       System.out.println("Digital Meter Facility is coming soon..");
    }
}

package com.ravi.java_8_features;

public class Car implements Vehicle
{

   @Override
   public void run()
   {
      System.out.println("Car is running");
   }

   @Override
   public void horn()
   {
      System.out.println("Car Has Horn");
   }

   @Override
   public  void digitalMeter() //JDK 1.8
    {
      System.out.println("Car is having Digital Meter");
    }
}

package com.ravi.java_8_features;

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

    @Override
    public void horn()
    {
       System.out.println("Bike Has Horn");
    }
}

package com.ravi.java_8_features;

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

      v = new Car();  v.run(); v.horn(); v.digitalMeter();
      System.out.println("................");
      v = new Bike();  v.run();  v.horn(); v.digitalMeter();
   }
}
```