Interface

Interface is same like as abstract class in java

Q. Why use the interface?

1) To Achieve 100% abstraction
2) To Achieve Dynamic Polymorphism
3) To Achieve Multiple Inheritance

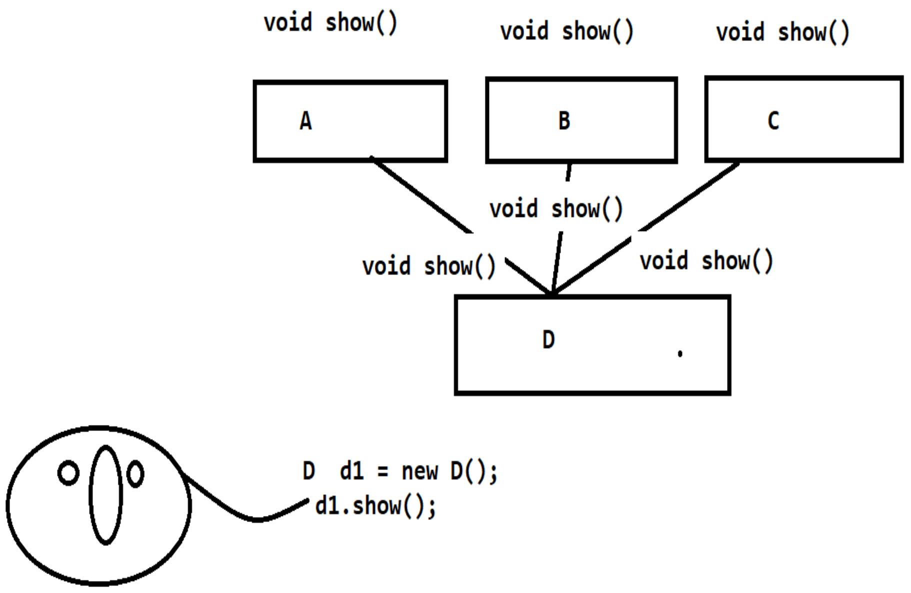
Q. Why Java Not Use the Classes For Multiple Inheritance

Java not use the classes for multiple inheritance because of diamond problem

Q. What is the diamond problem?

Multiple inheritances means more than one parent classes and single child class called as multiple inheritances.
In this case there is possibility in different parent may be contain the same name method and if we create object of child class and try to call the method whose name same in different parent class then compiler may be get confused called as diamond problem.

Example



If we think about above diagram then we have the three different parent class names as A B and C
And we have only one child class name as D and Three parent class contain method name as show()
Which is inherited in D class and if we create the object of D class and try to call the method show ()
Then compiler gets confused and if we want to solve this problem we have the interface concept.

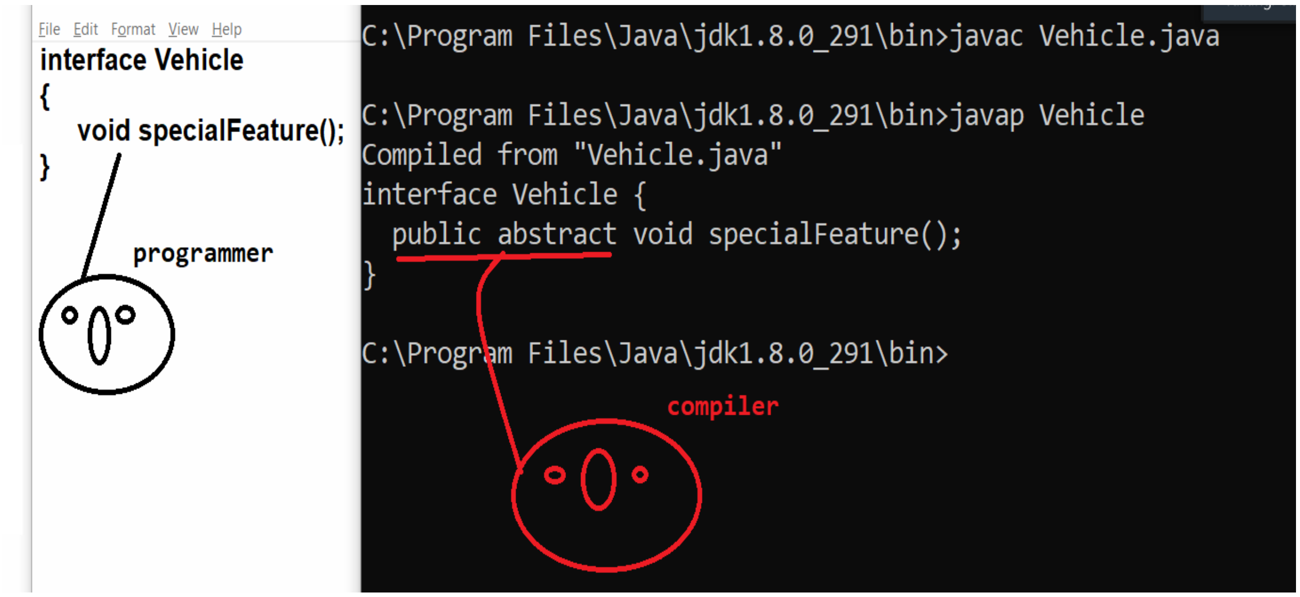
How to Declare the

```
syntax:
interface interfacename
{
      returntype functionname(arguments);
}
```

Example

```
interface Vehicle
{
    void specialFeature();
}
```

Note: we cannot create the object of interface because it is internally abstract class and we cannot write the logic of interface method because it is internally public abstract   and abstract method cannot have logic so we cannot write the logic of interface method.

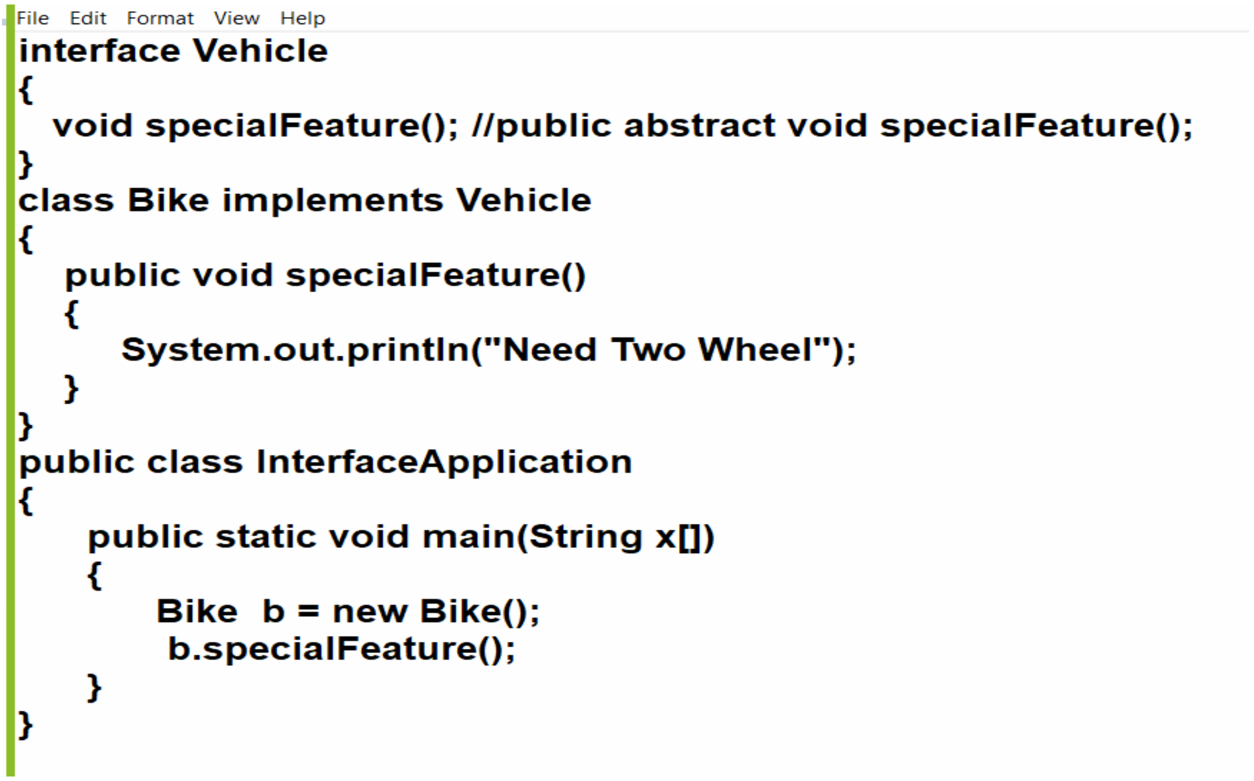


```
interface Vehicle
{
    void specialFeature();
}
```

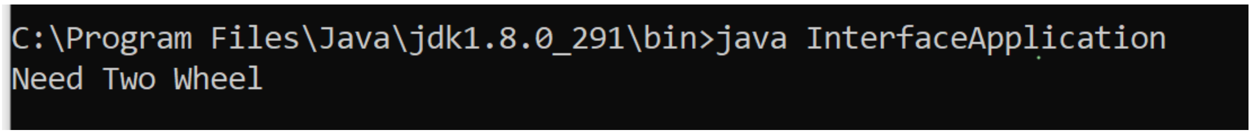
```
C:\Program Files\Java\jdk1.8.0_291\bin>javac Vehicle.java

C:\Program Files\Java\jdk1.8.0_291\bin>javap Vehicle
Compiled from "Vehicle.java"
interface Vehicle {
  public abstract void specialFeature();
}

C:\Program Files\Java\jdk1.8.0_291\bin>
```

Q. If interface method cannot have logic Then where we can Write logic of interface method?

If we want to write the logic of interface method then we need to implement the interface in any another class and override its method and write its logic.



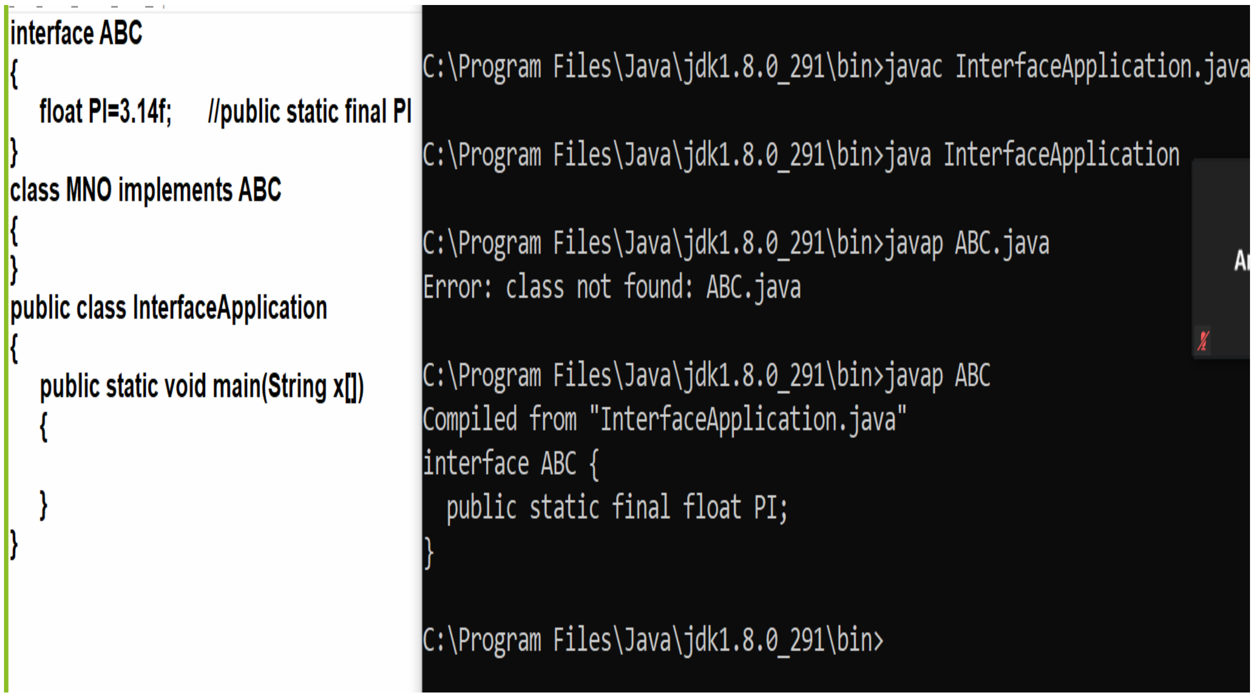
```
interface Vehicle
{
  void specialFeature(); //public abstract void specialFeature();
}
class Bike implements Vehicle
{
   public void specialFeature()
   {
      System.out.println("Need Two Wheel");
   }
}
public class InterfaceApplication
{
     public static void main(String x[])
     {
          Bike  b = new Bike();
           b.specialFeature();
     }
}
```



```
C:\Program Files\Java\jdk1.8.0_291\bin>java InterfaceApplication
Need Two Wheel
```

If we want to work with interface in java we have the some important points

1) Interface cannot create its object
2) Interface method cannot have definition
3) Interface variables are by default public static final
 Means if we want to declare the variable within interface we must have to initialize some value in it.
If we declare the variable within interface and if we not initialize value then java compiler will generate error to us



```
interface ABC
{
    float PI=3.14f;      //public static final PI
}
class MNO implements ABC
{
}
public class InterfaceApplication
{
    public static void main(String x[])
    {

    }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InterfaceApplication.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InterfaceApplication

C:\Program Files\Java\jdk1.8.0_291\bin>javap ABC.java
Error: class not found: ABC.java

C:\Program Files\Java\jdk1.8.0_291\bin>javap ABC
Compiled from "InterfaceApplication.java"
interface ABC {
  public static final float PI;
}

C:\Program Files\Java\jdk1.8.0_291\bin>
```

4) If interface contain more than one methods then all method must be override where interface get implemented

```
interface ABC
{
   public void s1();
   public void s2();
   public void s3();
}
class MNO implements ABC
{
   public void s1()
   { System.out.println("I required this method");
   }
   public void s2()
  {
  }
```

```
   public void s3()
   {
   }
}
class PQR implements ABC
{
      public void s2()
      { System.out.println("I required this method");
      }
     public void s1()
     {
     }
    public void s3()
     {
     }
}
class STV implements ABC
{
     public void s2()
     { System.out.println("I required this");
     }
     public void s1()
     {
     }
     public void s3()
     {
     }
}
public class AbsInfApp
{
    public static void main(String x[])
    {
   }
}
```

Note: this is the major limitation of interface methods or abstract method in java.
If we want to solve this problem in java we have the adapter class concept.

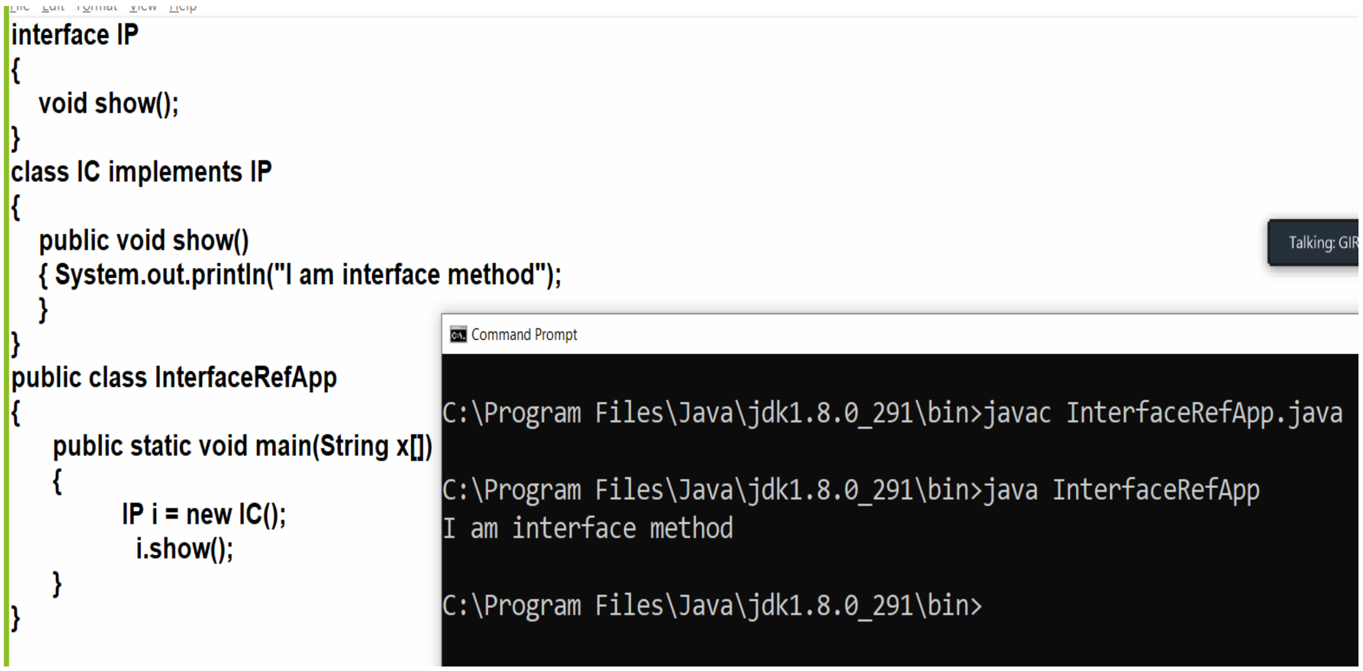
Q. what is the adapter class ?

Adapter class is intermediater class which contain the all methods of interface and it is able to provide the specific method to interface implementor classes called as adapter class.

Example

```
interface ABC
{
   public void s1();
   public void s2();
   public void s3();
}
class ADP implements ABC
{
   public void s1()
   {
   }
  public void s2()
  {
  }
  public void s3()
   {
   }
}
class MNO extends ADP
{
   public void s1()
   { System.out.println("I required this method");
   }

}
class PQR extends ADP
{
     public void s2()
     { System.out.println("I required this method");
     }
}
class STV extends ADP
{
    public void s3()
    { System.out.println("I required this");
    }
 }
public class AbsInfApp
{
   public static void main(String x[])
   {
           MNO  m = new MNO();
            m.s1();
          PQR  p = new PQR();
              p.s2();
          STV s=new STV();
             s.s3();
   }
}
```

5) Interface cannot create its object but can create its reference if we want to create the reference of interface we must be create the object of its implementor class.



```
interface IP
{
   void show();
}
class IC implements IP
{
   public void show()
   { System.out.println("I am interface method");
   }
}
public class InterfaceRefApp
{
     public static void main(String x[])
     {
          IP i = new IC();
           i.show();
     }
}
```

```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InterfaceRefApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InterfaceRefApp
I am interface method

C:\Program Files\Java\jdk1.8.0_291\bin>
```

The Major goal of interface reference is to achieve dynamic polymorphism or loose coupling.

Example of Loose Coupling

```
import java.util.*;
interface Value
{    void setValue(int x,int y);
     void performOperation();
}
class Add implements Value
{  int first,second;
   public void setValue(int x,int y)
   {
       first=x;
       second=y;
   }
   public void performOperation()
   {
        System.out.println("Addition is  "+(first+second));
   }
}
class Mul implements Value
{   int first,second;
     public void setValue(int x,int y)
     {
          first=x;
          second=y;
     }
    public void performOperation()
    { System.out.println("Multiplication is  "+(first*second));
     }
}
class Calculator
{
    void performCal(Value v)
    {
          v.setValue(100,200);
          v.performOperation();
    }
}
public class LooseCouplingApp
{
    public static void main(String x[])
    {
         Scanner xyz = new Scanner(System.in);
          Value v=null;
         Calculator c = new Calculator();
      System.out.println("1:Addition");
   System.out.println("2:Multiplication");
   System.out.println("Enter your choice");
   int choice=xyz.nextInt();
    switch(choice)
    {
        case 1:
         v = new Add();
         c.performCal(v);
       break;
        case 2:
         v = new Mul();
         c.performCal(v);
        break;
       default:
         System.out.println("Wrong choice");
    }
  }
}
```

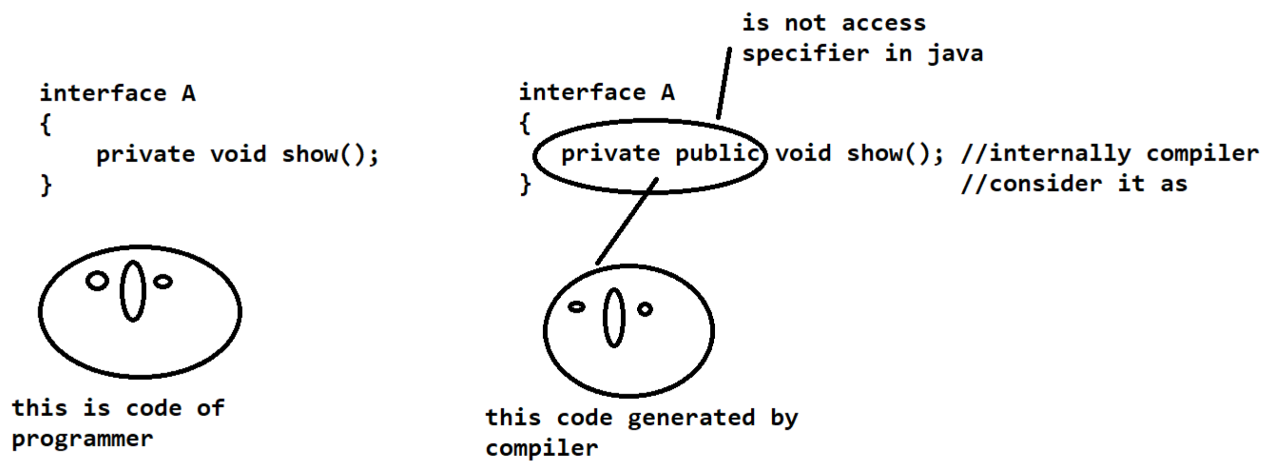
6) We cannot declare the interface method as private, protected, static and final

private: there are two reason we cannot declare interface method as private.

i) private method cannot support to inheritance as well as not support to overriding and interface cannot work without inheritance and overriding.

ii) interface is method is by default public and if we declare it as private then internally compiler consider it as private public and in private public is not access specifier so we cannot declare interface method as private.

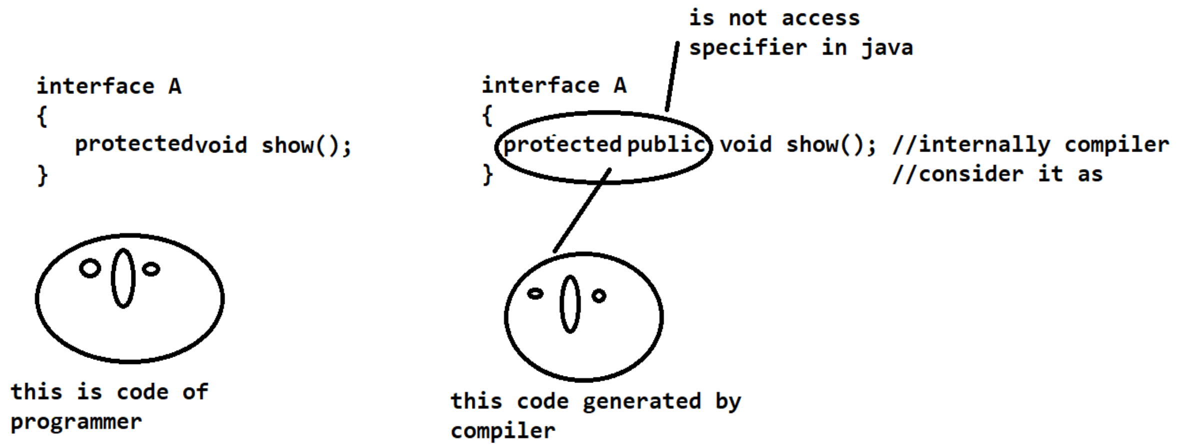
Above statement meaning shown in following diagram



```
interface A
{
    private void show();
}
```

```
interface A
{
  private public void show(); //internally compiler
}                             //consider it as
```

Protected: we cannot declare interface method as protected because interface method is by default public and if we declare it as protected then internally compiler consider it as protected so in java there is no public protected access specifier.



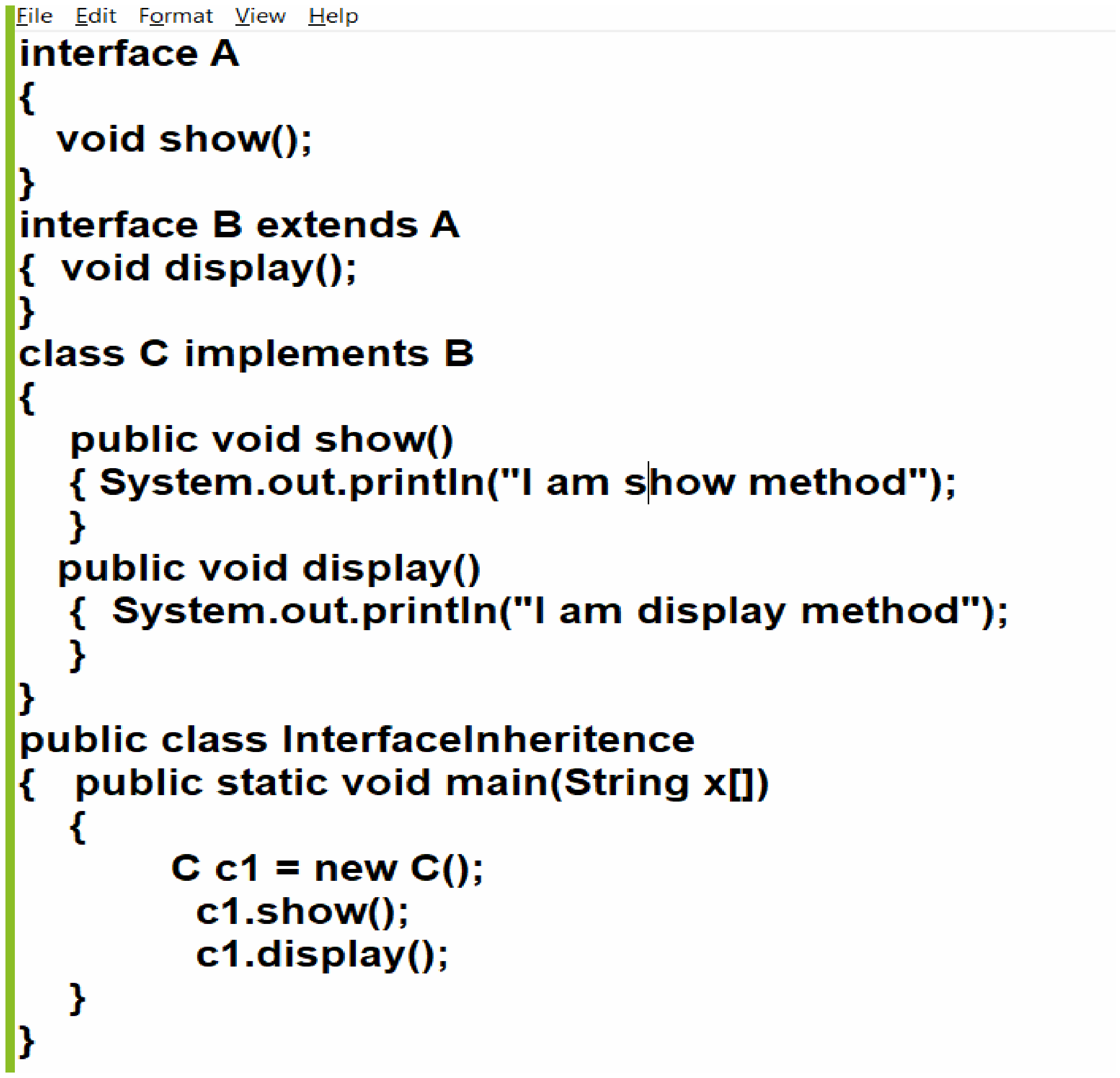
```
interface A
{
   protectedvoid show();
}
```

```
interface A
{
 protected public void show(); //internally compiler
}                              //consider it as
```

final: we cannot declare interface method as final because final method cannot override in child class and interface method must be

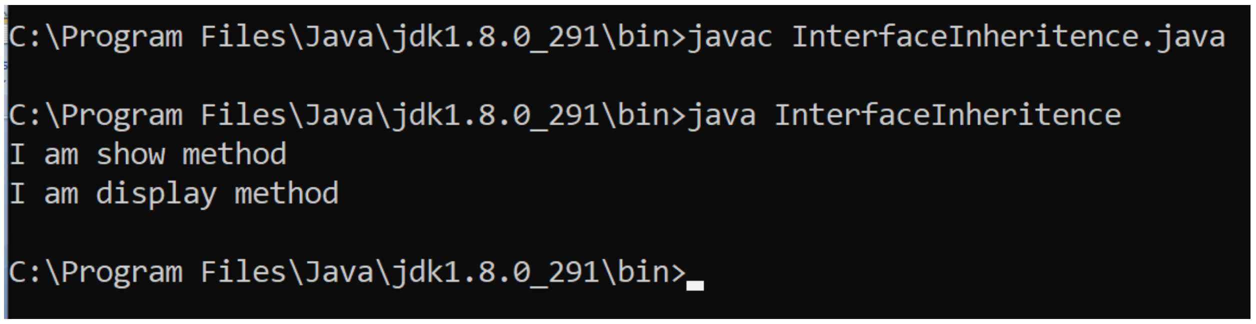
override in child class so we cannot declare interface method as final
static: static method must have definition and interface method cannot have definition.

7) If we want to inherit the interface to interface we have the extends keyword



```
interface A
{
   void show();
}
interface B extends A
{  void display();
}
class C implements B
{
    public void show()
    { System.out.println("I am show method");
    }
   public void display()
    {  System.out.println("I am display method");
    }
}
public class InterfaceInheritence
{   public static void main(String x[])
    {
            C c1 = new C();
              c1.show();
              c1.display();
    }
}
```

Output:



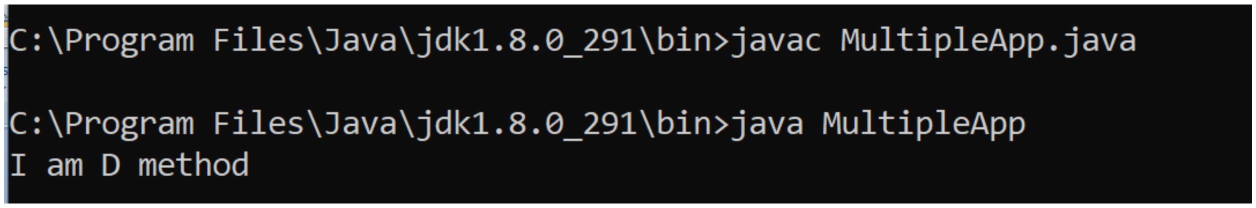
```
C:\Program Files\Java\jdk1.8.0_291\bin>javac InterfaceInheritence.java

C:\Program Files\Java\jdk1.8.0_291\bin>java InterfaceInheritence
I am show method
I am display method

C:\Program Files\Java\jdk1.8.0_291\bin>
```

How To Achieve Multiple Inheritance using java

Multiple Inheritance means more than one parent and single child called as multiple inheritance.
In multiple parent there should be only one parent class and remaining are interfaces.

Example

```
interface A
{
   void show();
}
interface B
{
    void show();
}
class C
{
    void show()
   { System.out.println("I am c method");
    }
}
class D extends C implements A,B
{
    public void show()
    {
        System.out.println("I am D method");
    }
}
public class MultipleApp
{
   public static void main(String x[])
   {
         D d1 = new D();
           d1.show();
   }
}
```



```
C:\Program Files\Java\jdk1.8.0_291\bin>javac MultipleApp.java

C:\Program Files\Java\jdk1.8.0_291\bin>java MultipleApp
I am D method
```