# OBJECT ORIENTED PROGRAMMING USING JAVA

## OUTLINE

- INTERFACE IN JAVA

## INTERFACE

- The interface in java is **a mechanism to achieve abstraction.**
- An interface in **Java is a blueprint of a class.** It specify what a class must do  but not how.
- An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with **abstract methods, an interface** may also **contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.**
- **It is used to achieve abstraction and multiple inheritance in Java.**
- A **class implements an interface**, thereby **inheriting the abstract methods** of the interface.

## DEFINITION

- Any **service requirement specification** or **any contract between client and service provider** or **100 % pure abstract class** is nothing but interface.

**Examples**

- **Case 1:** Requirement Specification to develop **Servlet**
- **Case 2:** Client is asking to develop a IIITDM automation system.
- **Case 3:** Creating an interface for corona disease

## EXAMPLE 1: WEB SERVER

Servlet API

By

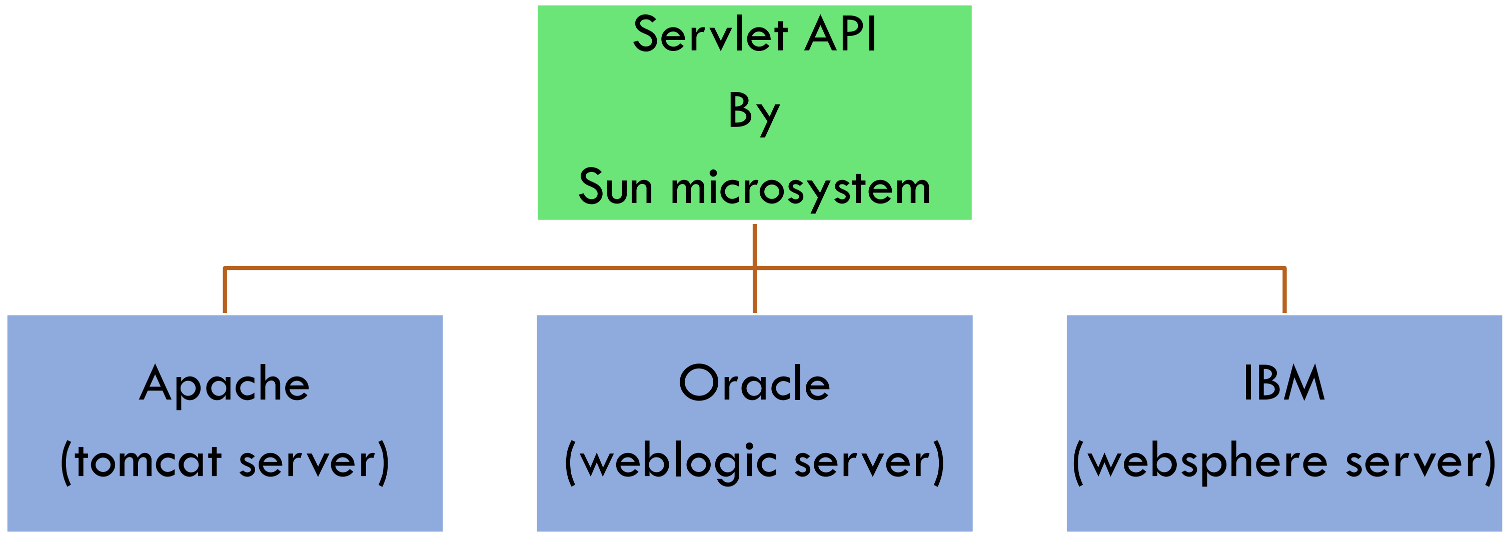
Sun microsystem

- Servlet API(application programming interface)
- For servlet API :define to develop service specification of web server by sunmicrosystem)

**SERVER VENDOR IS RESPOSIBLE TO IMPLEMENT**
**For Example:** ORACLE, APACHE, IBM

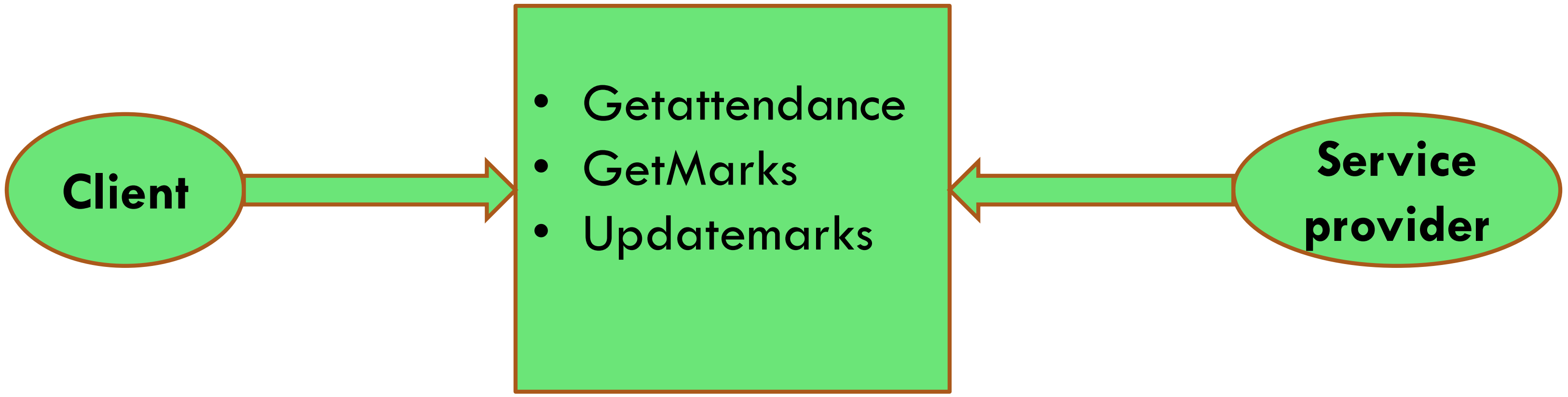
## EXAMPLE 1: WEB SERVER

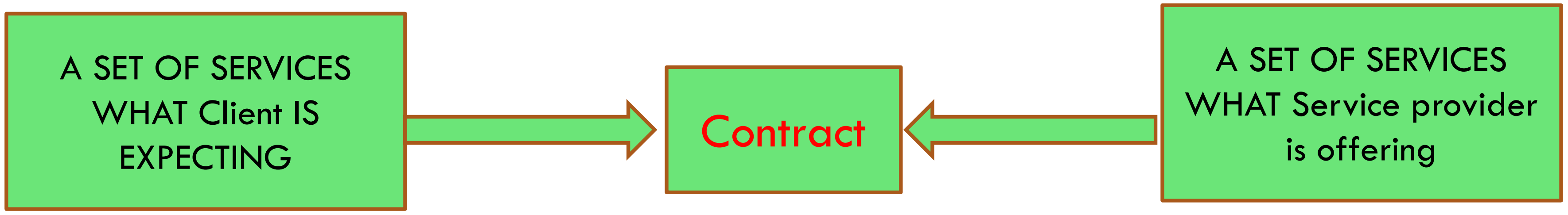
**Who will implement ??**



It means anything that run on apache will run on oracle and IBM also because all of them are using the same interface.

## EXAMPLE 2: IIITDM AUTOMATION SYSTEM




A SET OF SERVICES WHAT Client IS EXPECTING
Contract
A SET OF SERVICES WHAT Service provider is offering

## EXAMPLE 3

```
interface corona_disease
{
void symptoms();
void  vaccine();
void precaution();
}
```

Abstract Methods

## EXAMPLE 3

```
interface corona_disease
{
void symptoms();
void  vaccine();
void precaution();
}
```

```
class covid19 implements corona_disease
{
void symptoms()
{
// write implementation code
}
}
```

**Compile time error**

## EXAMPLE 3

```
interface corona_disease
{
void symptoms();
void  vaccine();
void precaution();
}
```

```
abstract class covid19 implements
corona_disease
{
void symptoms()
{
// write implementation code
}
}
```

- Inside interface:  **method is public and abstract** whether we are declaring or not.
- Hence wherever we are implementing a interface method compulsory we should declared as public other wise we will get compile time error.

## EXAMPLE 3

```
interface corona_disease

{

void symptoms();

void  vaccine();

void precaution();

}
```

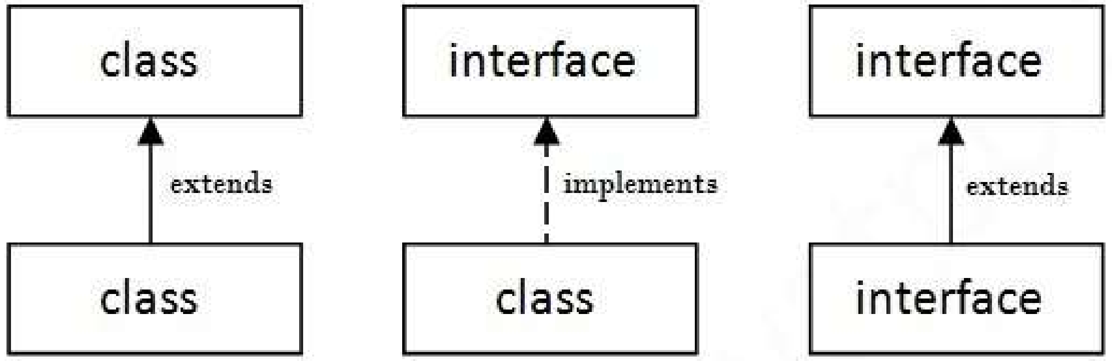
```
class covid19 implements corona_disease

{

void symptoms()
{
// write implementation code
}
void  vaccine(){}
void precaution(){}

}
```

- Inside interface:  **method is public and abstract** whether we are declaring or not.
- Hence wherever we are implementing a interface method compulsory we should declared as public other wise we will get compile time error.

## PRINTABLE INTERFACE HAS ONLY ONE METHOD, ITS IMPLEMENTATION IS PROVIDED IN A CLASS.

```
interface printable{
void print();  //ABSTRACT METHOD
}
class A6 implements printable{
public void print()
{System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }  }
```

## IS-A /HAS A DIAGRAM



a class extends another class,

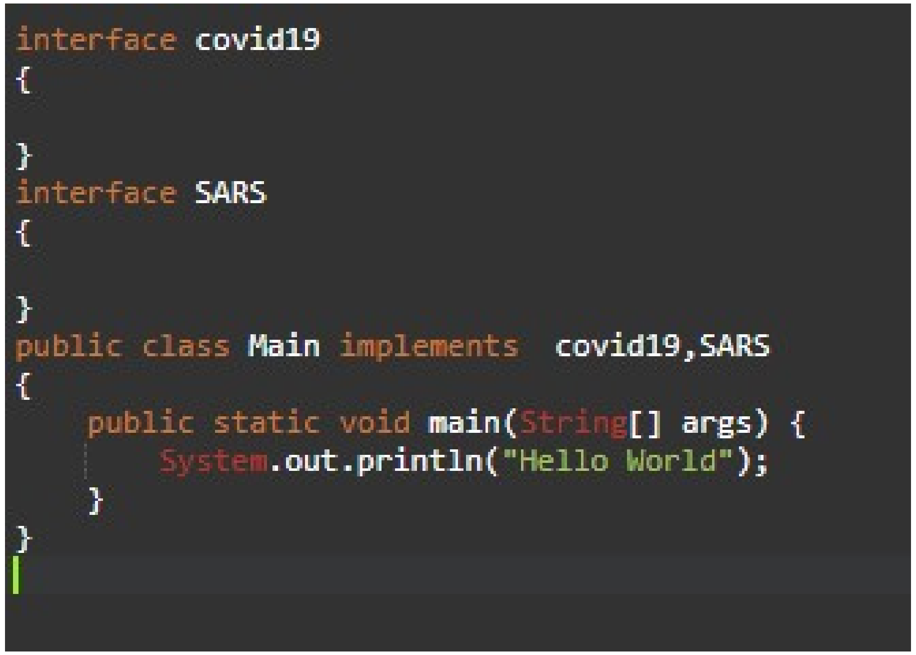
an interface extends another interface

but a **class implements an interface.**

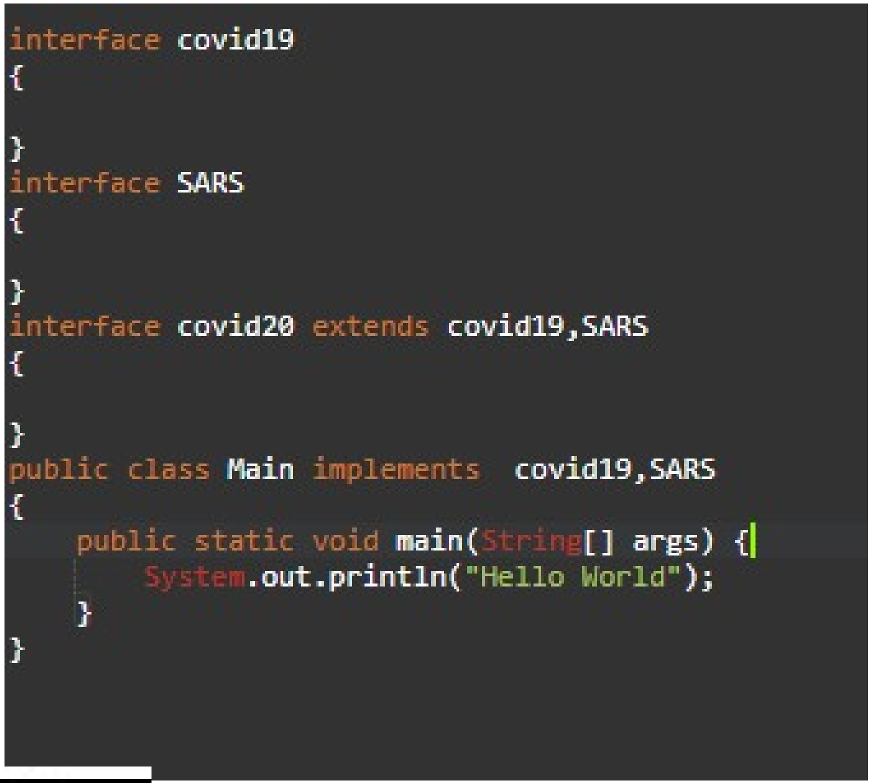
## RULES: EXTENDS VS IMPLEMENTS

- A class can extend only one class at a time
- A class can implement any number of interfaces
- An interface can extend any number of interfaces simultaneously
- A class can implement any number of interfaces simultaneously
- A class can extend and implement any number of interfaces simultaneously

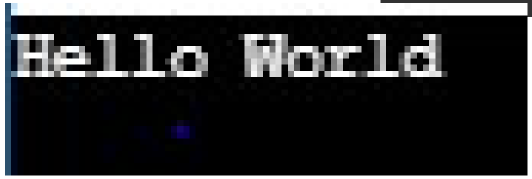
## EXTENDS VS IMPLEMENTS



```java
interface covid19
{

}
interface SARS
{

}
public class Main implements  covid19,SARS
{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

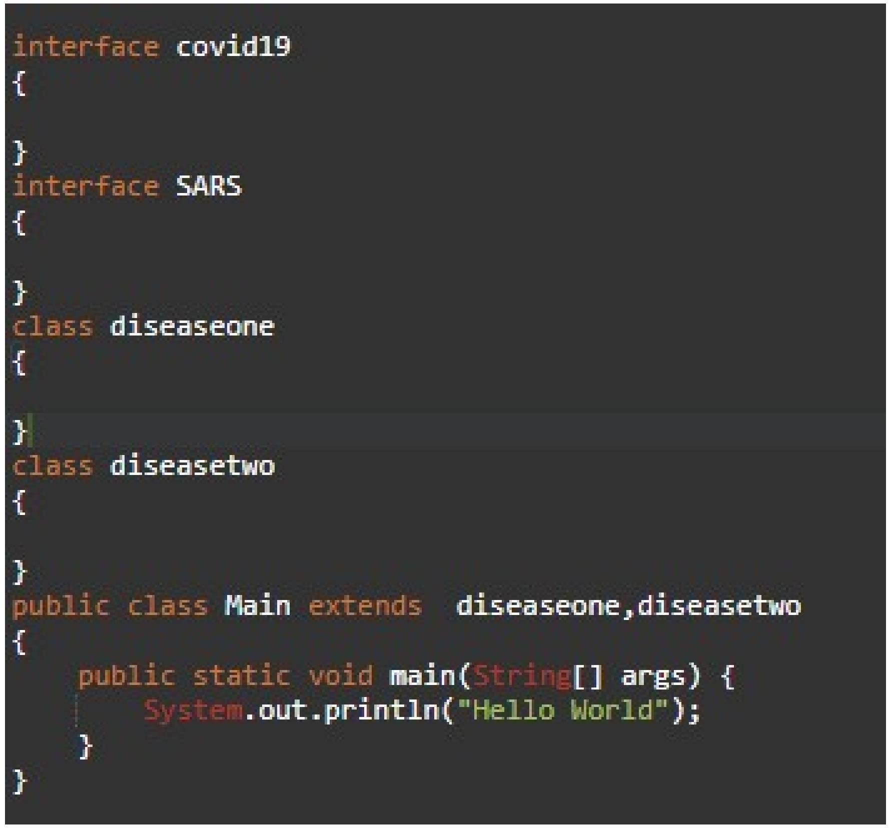


```java
interface covid19
{


}
interface SARS
{


}
interface covid20 extends covid19,SARS
{


}
public class Main implements  covid19,SARS
{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

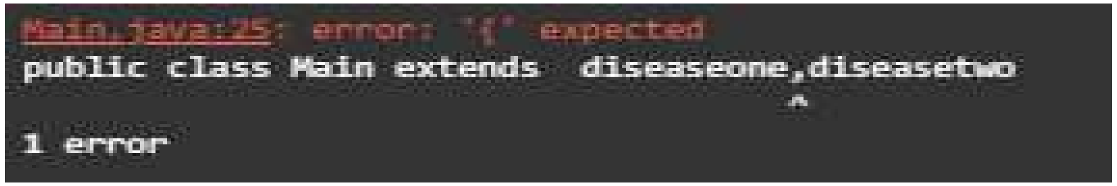


```
Hello World
```

## EXTENDS VS IMPLEMENTS

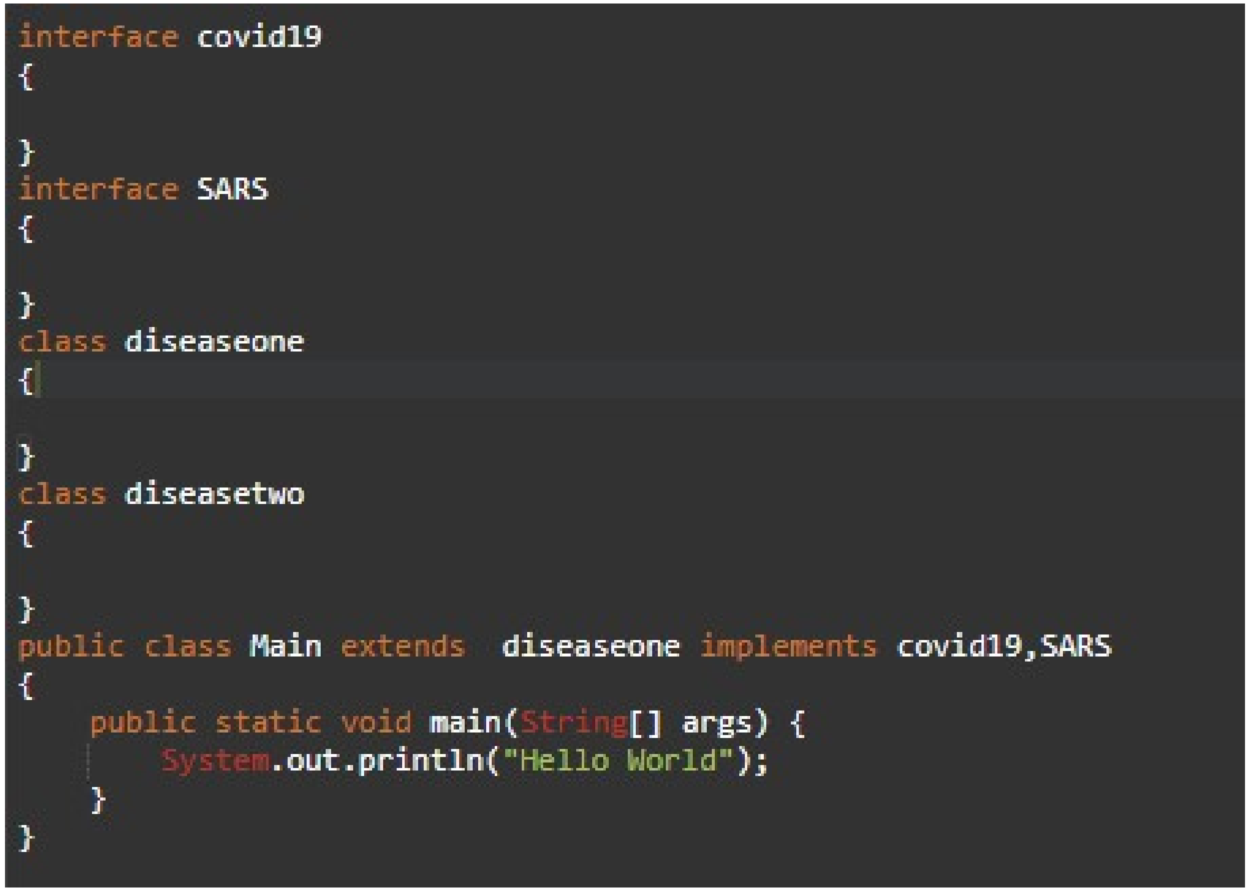


```
interface covid19
{

}
interface SARS
{

}
class diseaseone
{

}
class diseasetwo
{

}
public class Main extends  diseaseone,diseasetwo
{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

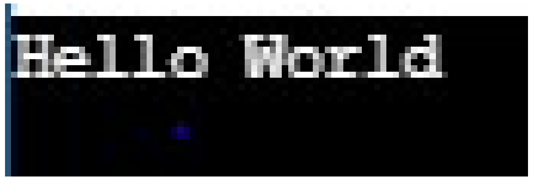


```
Main.java:25: error: '{' expected
public class Main extends  diseaseone,diseasetwo
                                     ^
1 error
```

## EXTENDS VS IMPLEMENTS



```
interface covid19
{

}
interface SARS
{

}
class diseaseone
{

}
class diseasetwo
{

}
public class Main extends  diseaseone implements covid19,SARS
{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```



```
Hello World
```

**Note: A class can extend another class and can implement any number of interfaces simultaneously.**

## WHICH ONE IS VALID?

- A class can extend any number of classes at a time.
- A class can implement only one interface at a time.
- An interface can extend only one interface at a time.
- An interface can implement any no. of interfaces simultaneously.
- A class can extend and can implement another interfaces but not both simultaneously.

## WHICH ONE IS VALID?

- A class can extend any number of classes at a time (**invalid**)
- A class can implement only one interface at a time (**invalid**)
- An interface can extend only one interface at a time (**invalid**)
- An interface can implement any no. of interfaces simultaneously. (**invalid**)
- A class can extend and can implement another interfaces but not both simultaneously (**invalid**)

## INTERFACE METHODS

- Every method present inside interface  is always **Public and abstract** whether we are declaring or not.

**Why public???**
Ans: To make this method available to every implementation class.

**Why abstract???**
Ans: implementation class is responsible to provide implementation

It means inside interface if you are writing method as:

**void m1();== public void m1();== abstract void m1();==abstract public void m1();   all are equal**

## MCQ

**Which of the following declaration are valid inside interface??**

public void m1(){}

private void m1(){}

protected void m1(){}

static void m1();

abstract public void m1();

## INTERFACE METHODS

**Which of the following declaration are valid inside interface??**

public void m1(){}

private void m1(){}

protected void m1(){}

static void m1();

abstract public void m1();

## INTERFACE VARIABLE

- An interface **can contain variable.**
- The main purpose of interface variable is to define **requirement level constants**
- Every interface variable is **public static final.**

**EXAMPLE: college automation system**
whenever name is allowed : **IIITDM**
whenever location is allowed : **Jabalpur**

Why final??
Ans: To make it constant

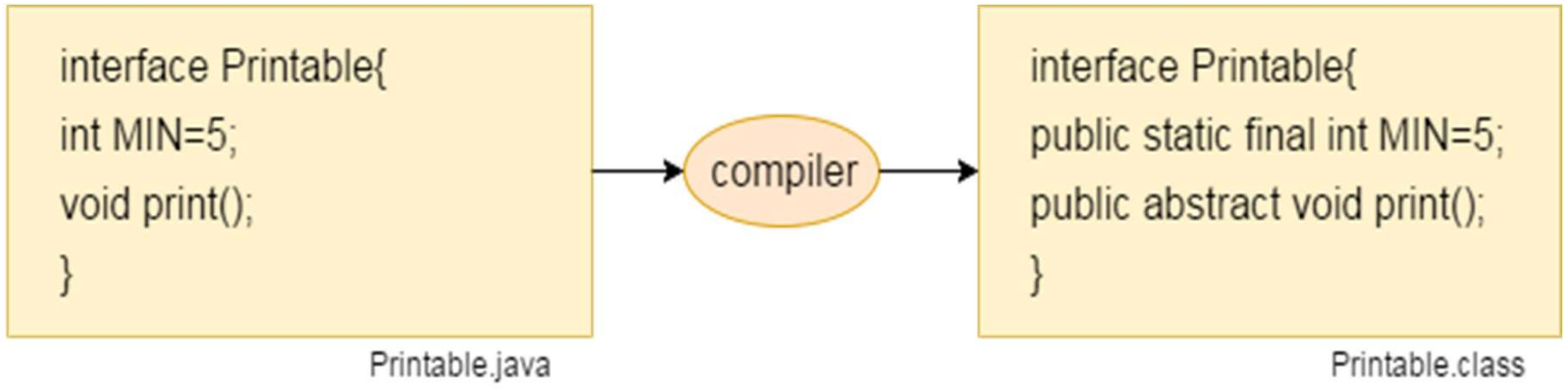
Why static??
Ans: Without creation of object implementation can access this variable

Why public??
Ans: To make this variable available to every implementation class

## FOLLOWING DECLARATION VARIABLES ARE EQUAL INSIDE INTERFACE??

int x=10; ==public int x=10; == static int x=10; ==  final int x=10; == public static  int x=10;

## INTERFACE DURING COMPILATION



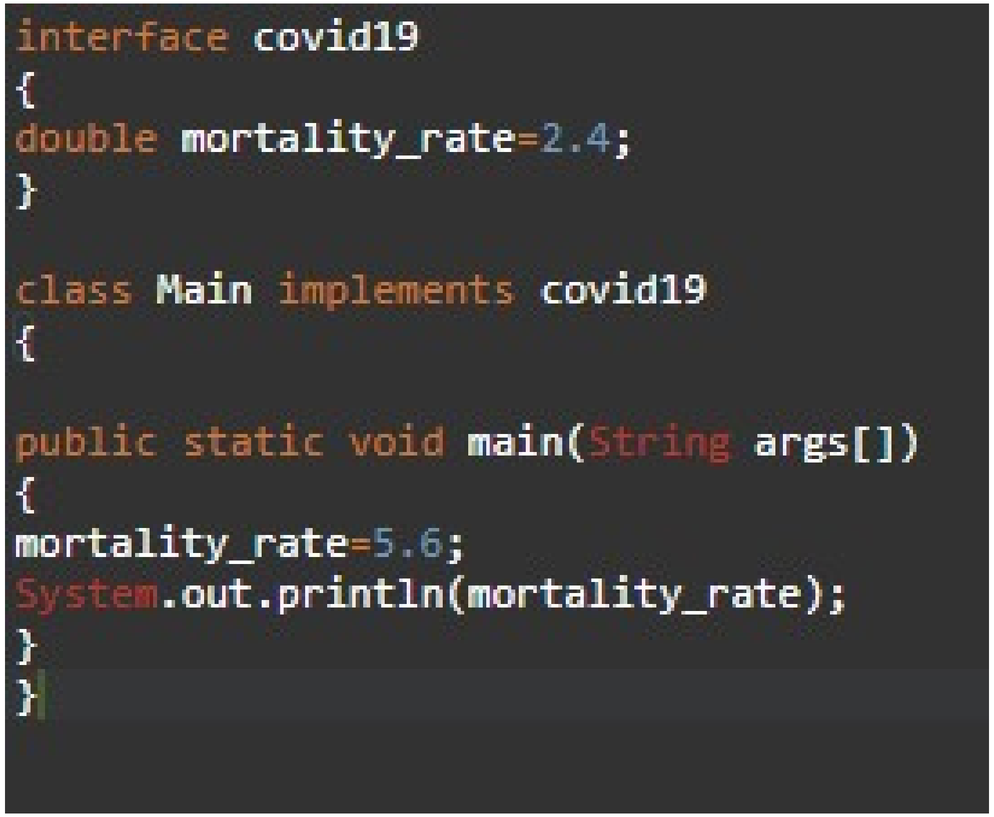
```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

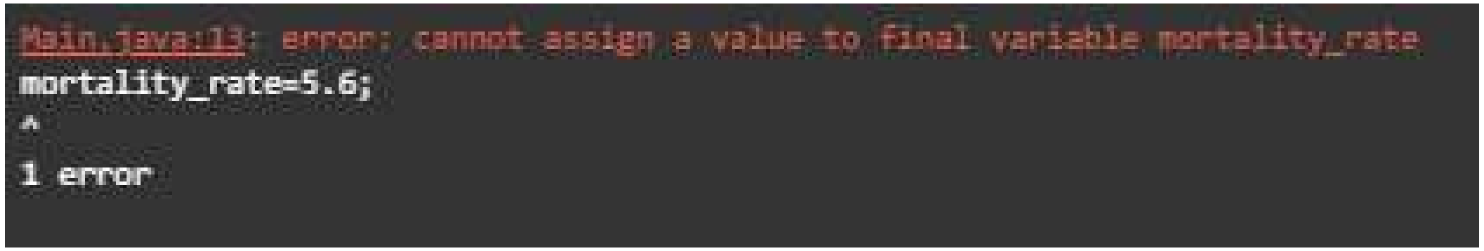
- The **interface** keyword is used to declare an interface.

- Interface fields are **public, static and final** by default, and methods are **public and abstract.**

## PREDICT THE OUTPUT??



```
interface covid19
{
double mortality_rate=2.4;
}

class Main implements covid19
{

public static void main(String args[])
{
mortality_rate=5.6;
System.out.println(mortality_rate);
}
}
```



```
Main.java:13: error: cannot assign a value to final variable mortality_rate
mortality_rate=5.6;
^
1 error
```

## INTERFACE VS CLASS

### SIMILARITES

1. An interface can contain any number of methods.
2. An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
3. The byte code of an interface appears in a .class file.
4. Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

For importing interfaces from other packages

**Import packagename.interfacename;**

**Or**

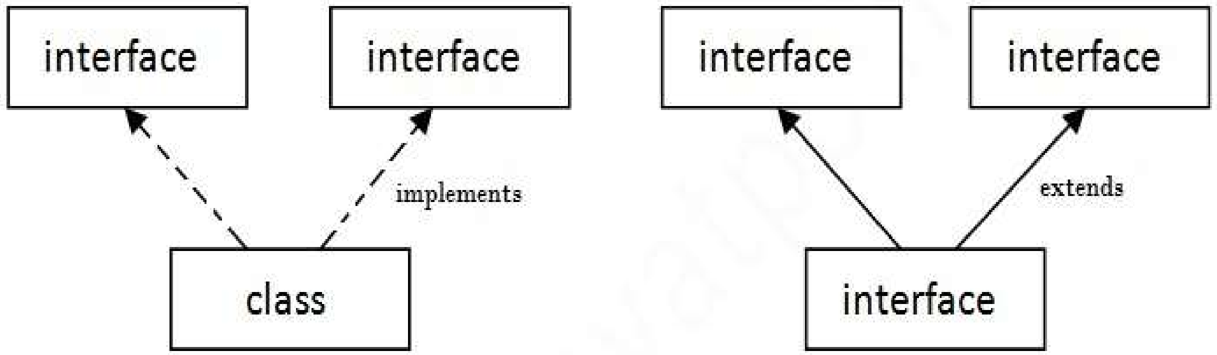
**Import packagename.*;**

### DIFFERENCES

1. You cannot instantiate an interface.
2. An interface does not contain any constructors.
3. All of the methods in an interface are abstract (for earlier version of java).
4. An interface **cannot contain instance fields**. The only fields that can appear in an interface must be declared both **static and final.**
5. **An interface is not extended by a class**; it is **implemented by a class.**
6. An interface can **extend multiple interfaces.**

## MULTIPLE INHERITANCE IN JAVA BY INTERFACE

If a class implements multiple interfaces, or an interface extends multiple interfaces
i.e. known as multiple inheritance.



Multiple Inheritance in Java

## MULTIPLE INHERITANCE IN JAVA BY INTERFACE

```
interface Printable{    //First Interface
void print();  }

interface Showable{    //Second Interface
void show();  }

class A7 implements Printable,Showable{   //Class implements both the interface

public void print()
{System.out.println("Hello");}

public void show()
{System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 } }
```

## NOTE

- Multiple inheritance is not supported in case of class because of ambiguity.
- But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

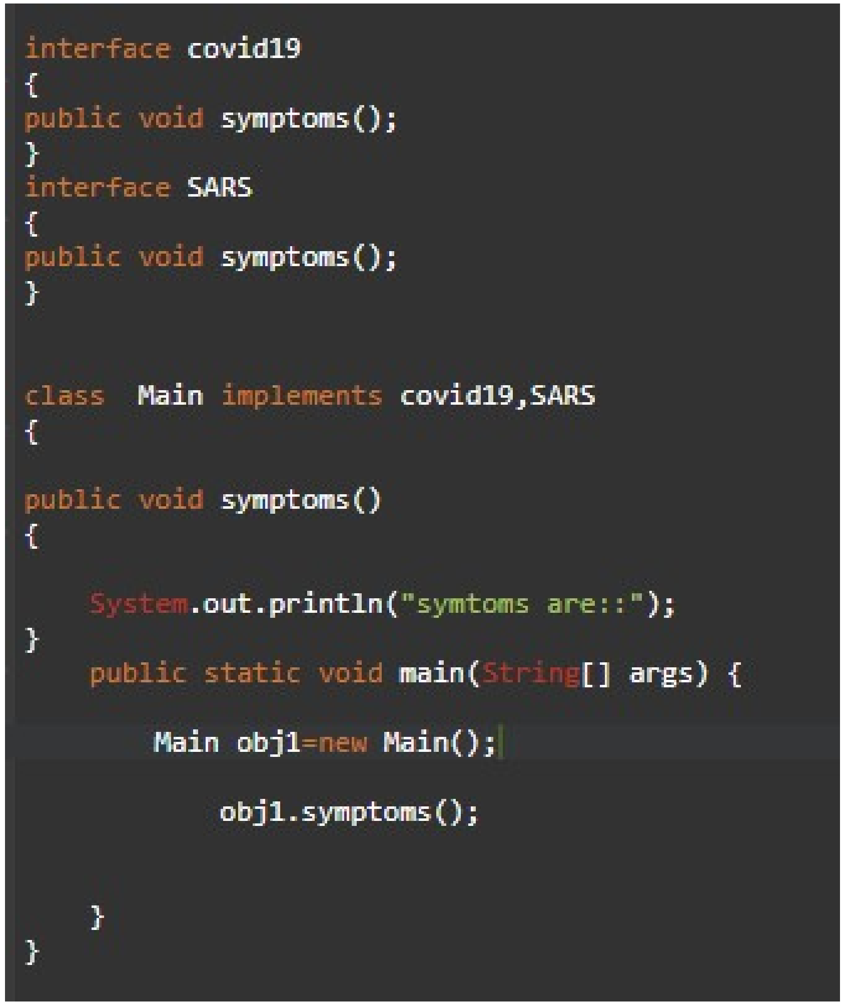
## INTERFACE NAMING CONFLICTS:

1. Method naming conflicts
2. Variable naming conflicts

# METHOD NAMING CONFLICTS

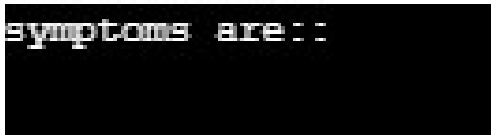
- A java class can implement any number of interfaces

**Case 1:** if two interface contains a method with a same signature and same return type then in the implementation class we have to provide implementation for only one method



```
interface covid19
{
public void symptoms();
}
interface SARS
{
public void symptoms();
}


class  Main implements covid19,SARS
{

public void symptoms()
{

    System.out.println("symtoms are::");
}
    public static void main(String[] args) {

        Main obj1=new Main();

            obj1.symptoms();


    }
}
```

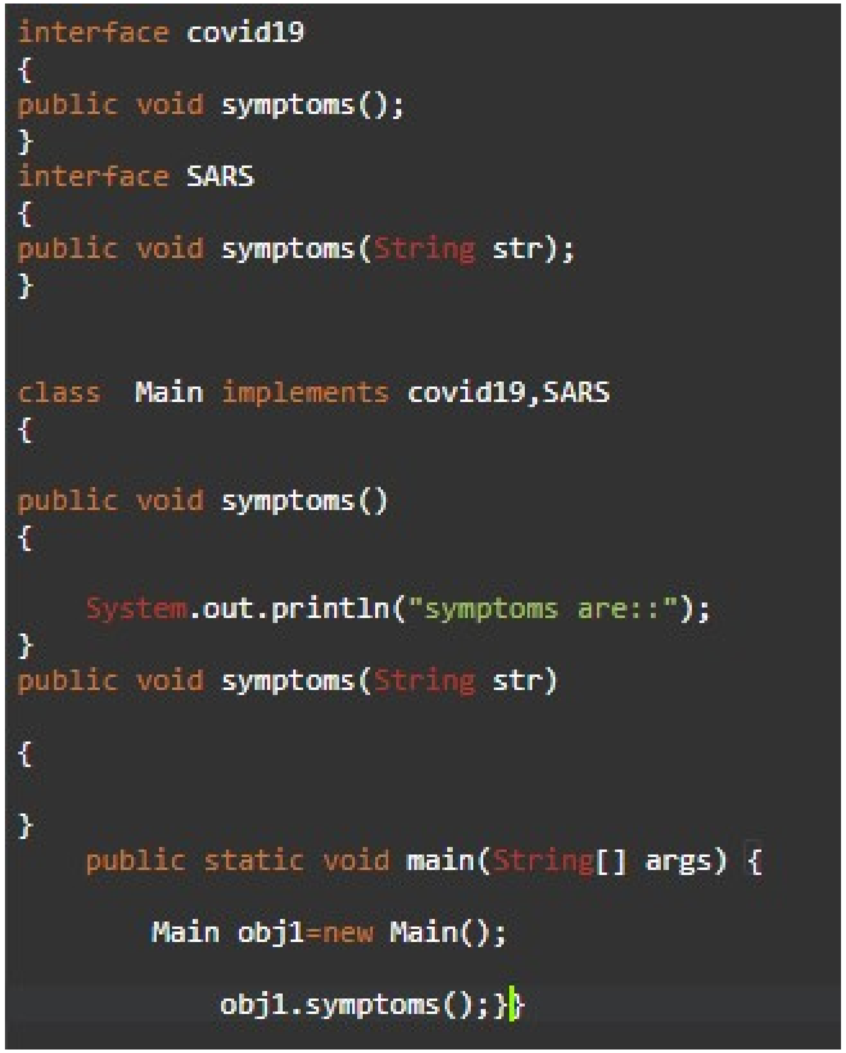
**Output**



```
symptoms are::
```

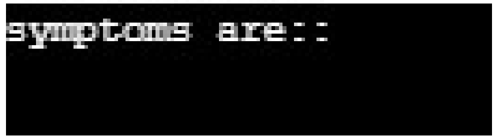
## METHOD NAMING CONFLICTS

**Case 2:** If two interface contains a method with a same name but arguments types and same return type then in the implementation class we have to provide implementation for both methods and these method act as overloaded methods.



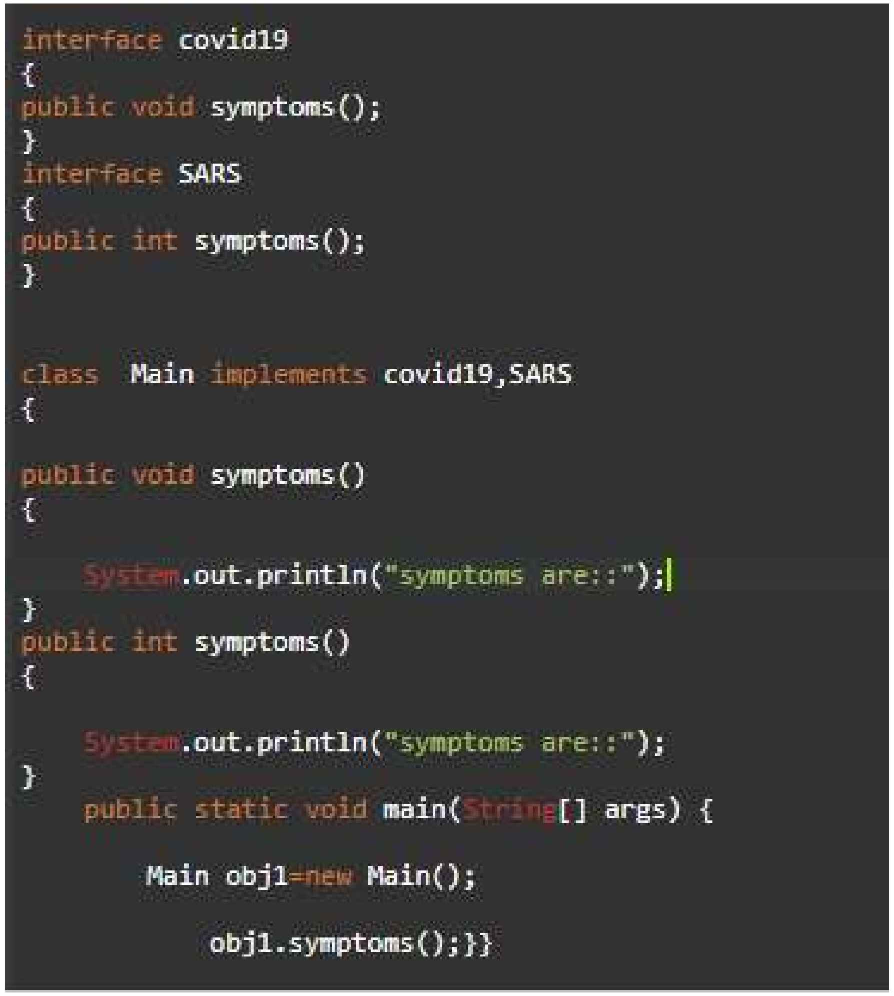
```
interface covid19
{
public void symptoms();
}
interface SARS
{
public void symptoms(String str);
}


class  Main implements covid19,SARS
{

public void symptoms()
{

    System.out.println("symptoms are::");
}
public void symptoms(String str)

{

}
    public static void main(String[] args) {

        Main obj1=new Main();

            obj1.symptoms();}}
```

**Output**



```
symptoms are::
```

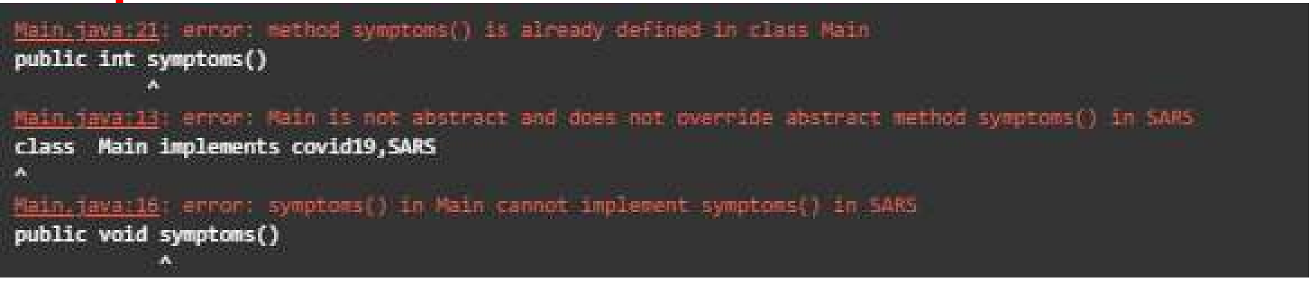
## METHOD NAMING CONFLICTS



```
interface covid19
{
public void symptoms();
}
interface SARS
{
public int symptoms();
}


class  Main implements covid19,SARS
{

public void symptoms()
{

    System.out.println("symptoms are::");
}
public int symptoms()
{

    System.out.println("symptoms are::");
}
    public static void main(String[] args) {

        Main obj1=new Main();

            obj1.symptoms();}}
```

**Case 3:** if two interface contains a method with a same signature but different return types then it is impossible to implement both the interface simultaneously.

**Output**



```
Main.java:21: error: method symptoms() is already defined in class Main
public int symptoms()
           ^
Main.java:13: error: Main is not abstract and does not override abstract method symptoms() in SARS
class  Main implements covid19,SARS
^
Main.java:16: error: symptoms() in Main cannot implement symptoms() in SARS
public void symptoms()
            ^
```

## QUERY??

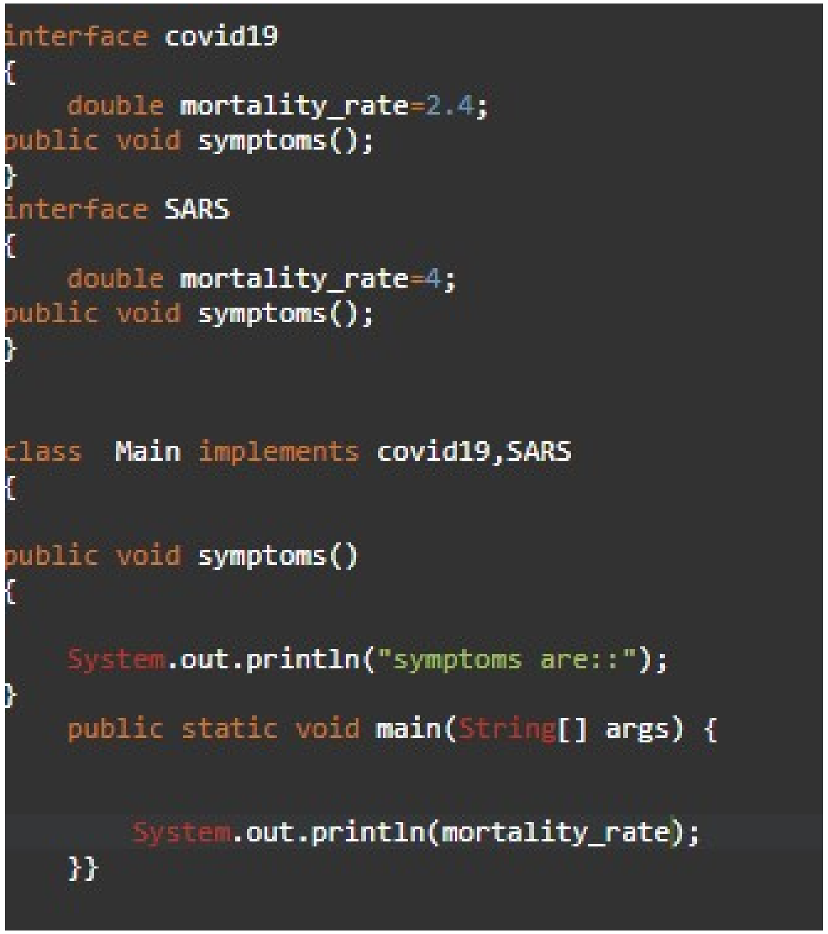
Question:: Then how can we implement??

Ans: It is **impossible to implement both the interfaces simultaneously.**

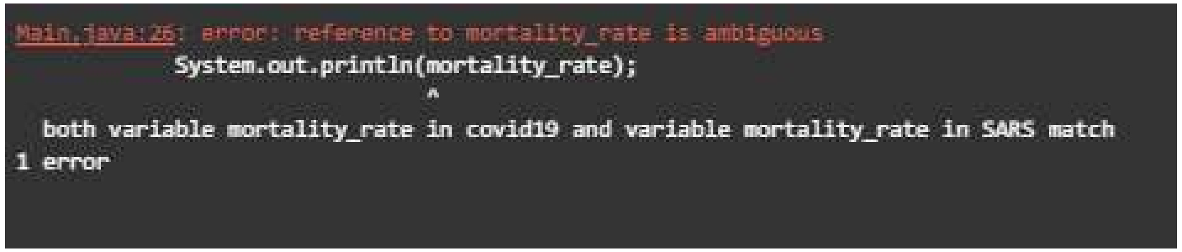
Question:: a java class can implement any number of interfaces simultaneously

Ans: **Yes, except one case.**

## VARIABLE NAME CONFLICT:: PROBLEM

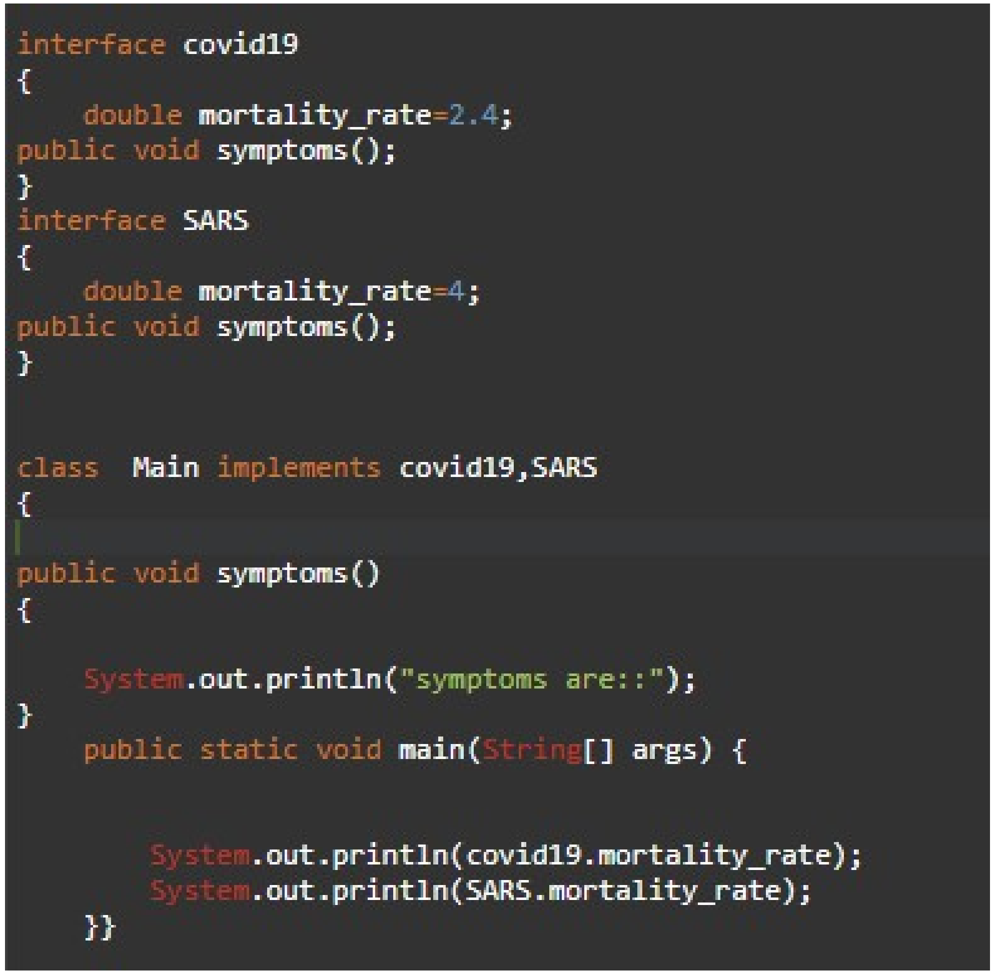


```
interface covid19
{
    double mortality_rate=2.4;
public void symptoms();
}
interface SARS
{
    double mortality_rate=4;
public void symptoms();
}


class  Main implements covid19,SARS
{

public void symptoms()
{

    System.out.println("symptoms are::");
}
    public static void main(String[] args) {


        System.out.println(mortality_rate);
    }}
```

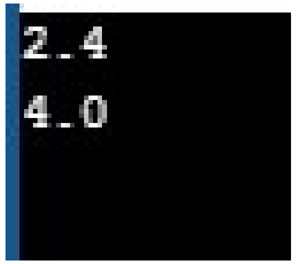


```
Main.java:26: error: reference to mortality_rate is ambiguous
            System.out.println(mortality_rate);
                               ^
  both variable mortality_rate in covid19 and variable mortality_rate in SARS match
1 error
```

## VARIABLE NAME CONFLICT:: SOLUTION



```
interface covid19
{
    double mortality_rate=2.4;
public void symptoms();
}
interface SARS
{
    double mortality_rate=4;
public void symptoms();
}


class  Main implements covid19,SARS
{

public void symptoms()
{

    System.out.println("symptoms are::");
}
    public static void main(String[] args) {


        System.out.println(covid19.mortality_rate);
        System.out.println(SARS.mortality_rate);
    }}
```



```
2.4
4.0
```

# INTERFACE INHERITANCE

A class implements interface but one interface extends another interface .

```
interface Printable{
void print();
}
interface Showable extends
 Printable{
void show();
}
```

```
class TestInterface4 implements Showable{
public void print()
{System.out.println("Hello");}


public void show()
{System.out.println("Welcome");}


public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 } }
```

## DEFAULT METHOD IN INTERFACE

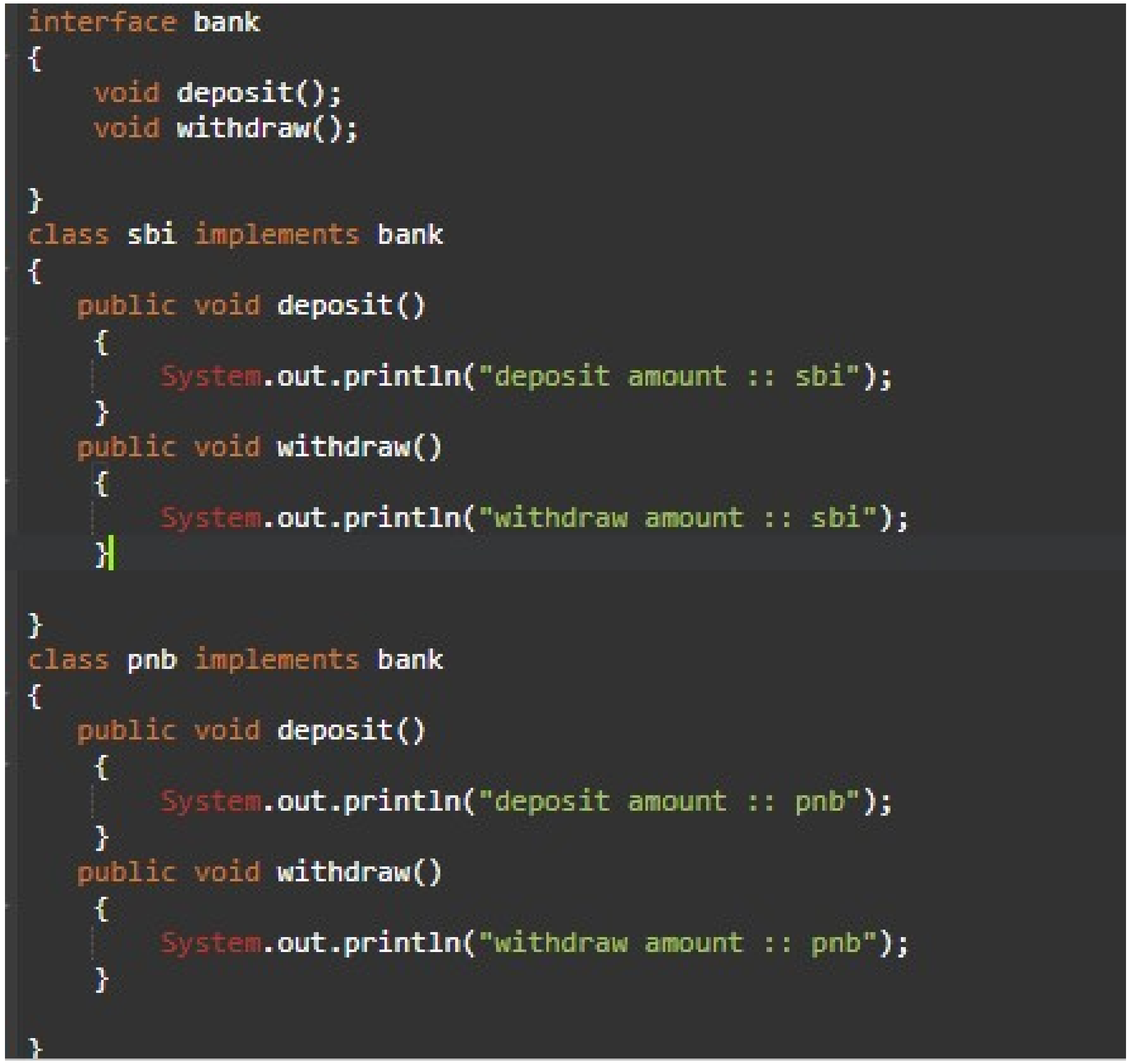
```
interface Drawable{
void draw();
default void msg() //Default Method
{System.out.println("default method");
}
}
class Rectangle implements Drawabl
e{
public void draw()
{System.out.println("drawing rectangl
e");}  }
```

```
class TestInterfaceDefault{
public static void main(String ar
gs[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

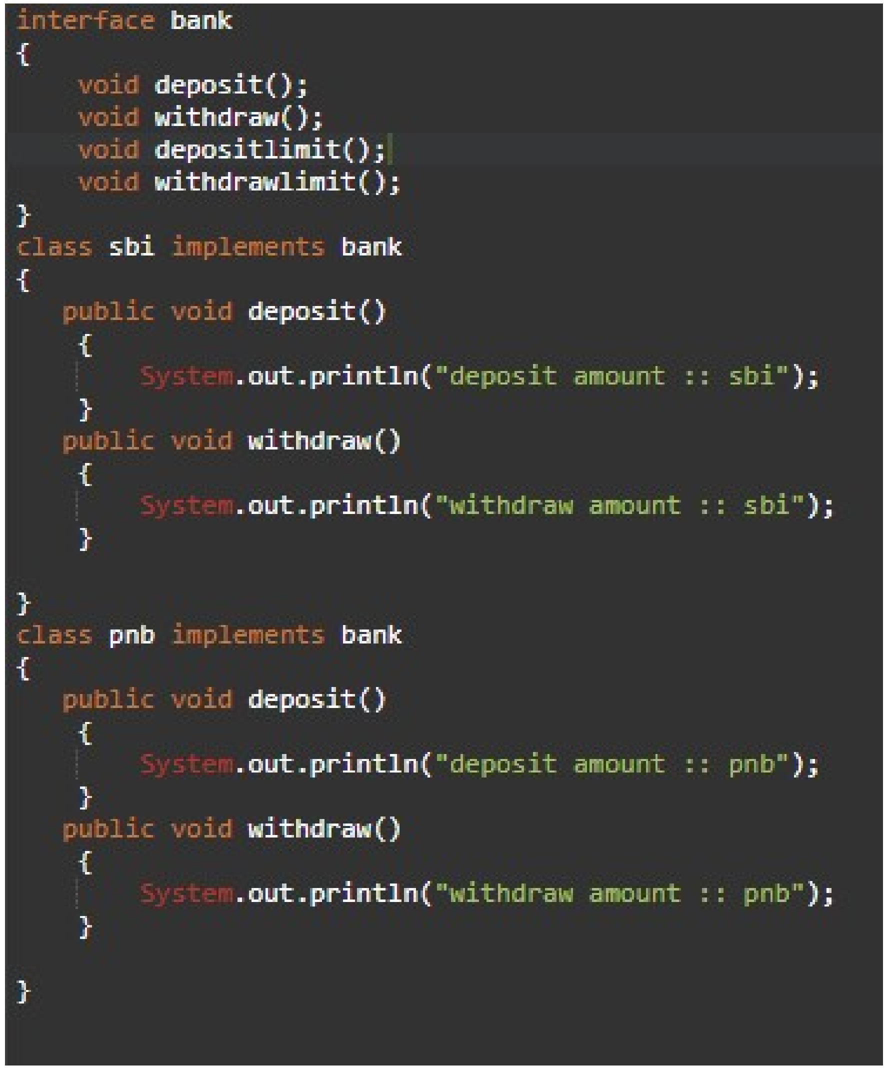
## DEFAULT METHOD IN INTERFACE

- Before Java 8, interfaces could have only abstract methods. The implementation of these methods has to be provided in a separate class. So, if a new method is to be added in an interface, then its implementation code has to be provided in the class implementing the same interface.
- To overcome this issue, Java 8 has introduced the concept of default methods which allow the interfaces to have methods with implementation without affecting the classes that implement the interface.

## DEFAULT METHOD IN INTERFACE:: NEED

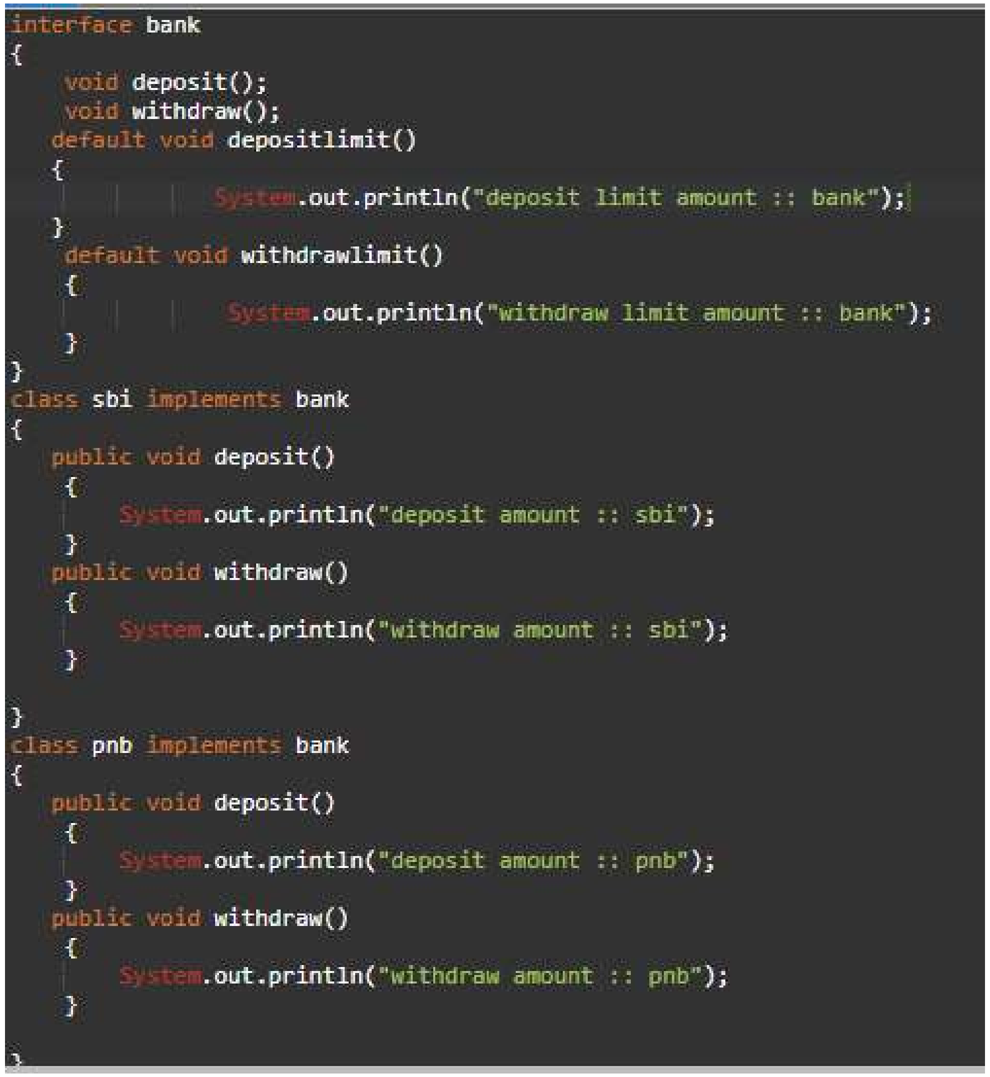


```
interface bank
{
    void deposit();
    void withdraw();

}
class sbi implements bank
{
   public void deposit()
    {
        System.out.println("deposit amount :: sbi");
    }
   public void withdraw()
    {
        System.out.println("withdraw amount :: sbi");
    }

}
class pnb implements bank
{
   public void deposit()
    {
        System.out.println("deposit amount :: pnb");
    }
   public void withdraw()
    {
        System.out.println("withdraw amount :: pnb");
    }

}
```



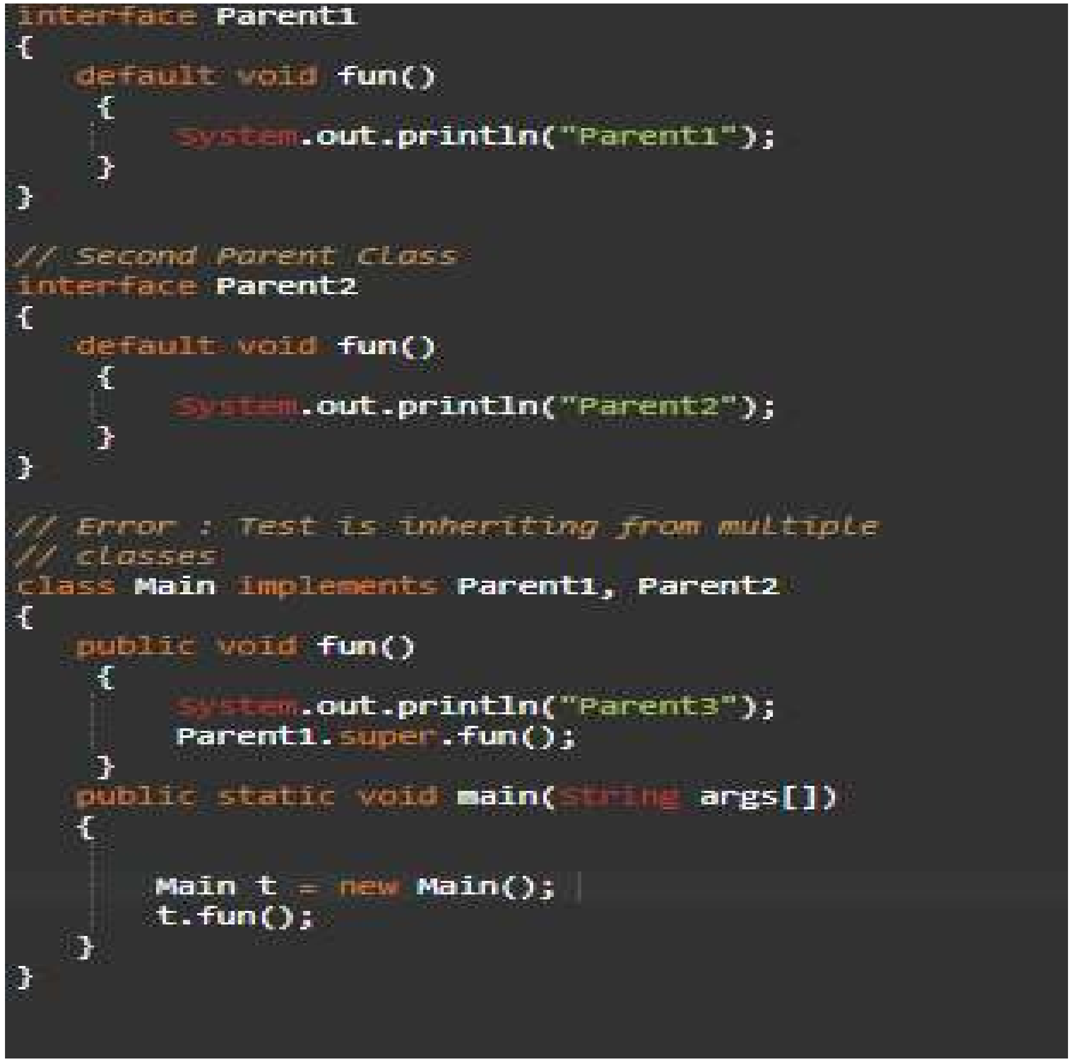
```
interface bank
{
    void deposit();
    void withdraw();
    void depositlimit();
    void withdrawlimit();
}
class sbi implements bank
{
   public void deposit()
    {
        System.out.println("deposit amount :: sbi");
    }
   public void withdraw()
    {
        System.out.println("withdraw amount :: sbi");
    }

}
class pnb implements bank
{
   public void deposit()
    {
        System.out.println("deposit amount :: pnb");
    }
   public void withdraw()
    {
        System.out.println("withdraw amount :: pnb");
    }

}
```

## DEFAULT METHOD IN INTERFACE:: SOLUTION



```
interface bank
{
     void deposit();
     void withdraw();
    default void depositlimit()
    {
                System.out.println("deposit limit amount :: bank");
    }
     default void withdrawlimit()
     {
                 System.out.println("withdraw limit amount :: bank");
     }
}
class sbi implements bank
{
    public void deposit()
     {
         System.out.println("deposit amount :: sbi");
     }
    public void withdraw()
     {
         System.out.println("withdraw amount :: sbi");
     }

}
class pnb implements bank
{
    public void deposit()
     {
         System.out.println("deposit amount :: pnb");
     }
    public void withdraw()
     {
         System.out.println("withdraw amount :: pnb");
     }

}
```

## DEFAULT METHOD IN INTERFACE (MULTIPLE INHERITANCE)



```
interface Parent1
{
    default void fun()
     {
         System.out.println("Parent1");
     }
}

// Second Parent Class
interface Parent2
{
    default void fun()
     {
         System.out.println("Parent2");
     }
}

// Error : Test is inheriting from multiple
// classes
class Main implements Parent1, Parent2
{
    public void fun()
     {
         System.out.println("Parent3");
         Parent1.super.fun();
     }
    public static void main(String args[])
    {

        Main t = new Main();
        t.fun();
    }
}
```

**Output:**
**Parent3**
**Parent1**

## STATIC METHOD IN INTERFACE

```
interface Drawable{
void draw();
static int cube(int x)
{return x*x*x;}
}
class Rectangle implements Drawable{
public void draw()
{System.out.println("drawing rectangle");}
}
```

```
class TestInterfaceStatic{
public static void main(String args[])
{
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}
}
```

## NESTED INTERFACE IN JAVA

- An interface can have another interface i.e. known as nested interface.

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```

## NESTED INTERFACE WHICH IS DECLARED WITHIN THE INTERFACE

```
interface Showable{
  void show();
  interface Message{
   void msg();
  }
}
class TestNestedInterface1 implements Showable.Message
{
 public void msg()
{System.out.println("Hello nested interface");}
public static void main(String args[]){
  Showable.Message message=new TestNestedInterface1();
  message.msg();
 } }
```

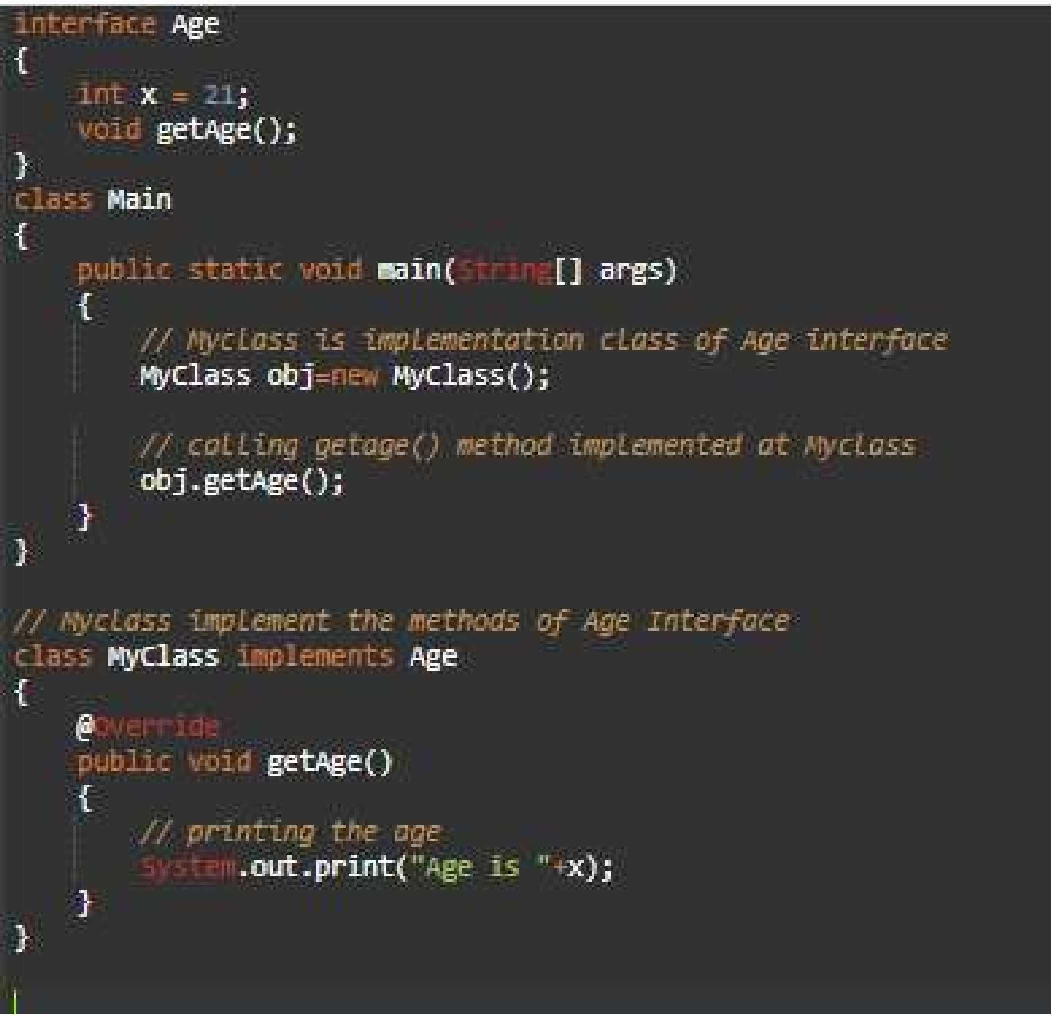
## NESTED INTERFACE WHICH IS DECLARED WITHIN THE CLASS

```
class A{
  interface Message{
   void msg();
  }
}
class TestNestedInterface2 implements A.Message{
 public void msg()
{
System.out.println("Hello nested interface");
}
public static void main(String args[]){
  A.Message message=new TestNestedInterface2();
  message.msg();
 }
}
```

- Interfaces can have only public and default access specifiers when declared outside any other class
- This interface declared in a class can either be default, public, protected not private.
- While implementing the interface, we mention the interface as **c_name.i_name** where **c_name** is the name of the class in which it is nested and **i_name** is the name of the interface itself.

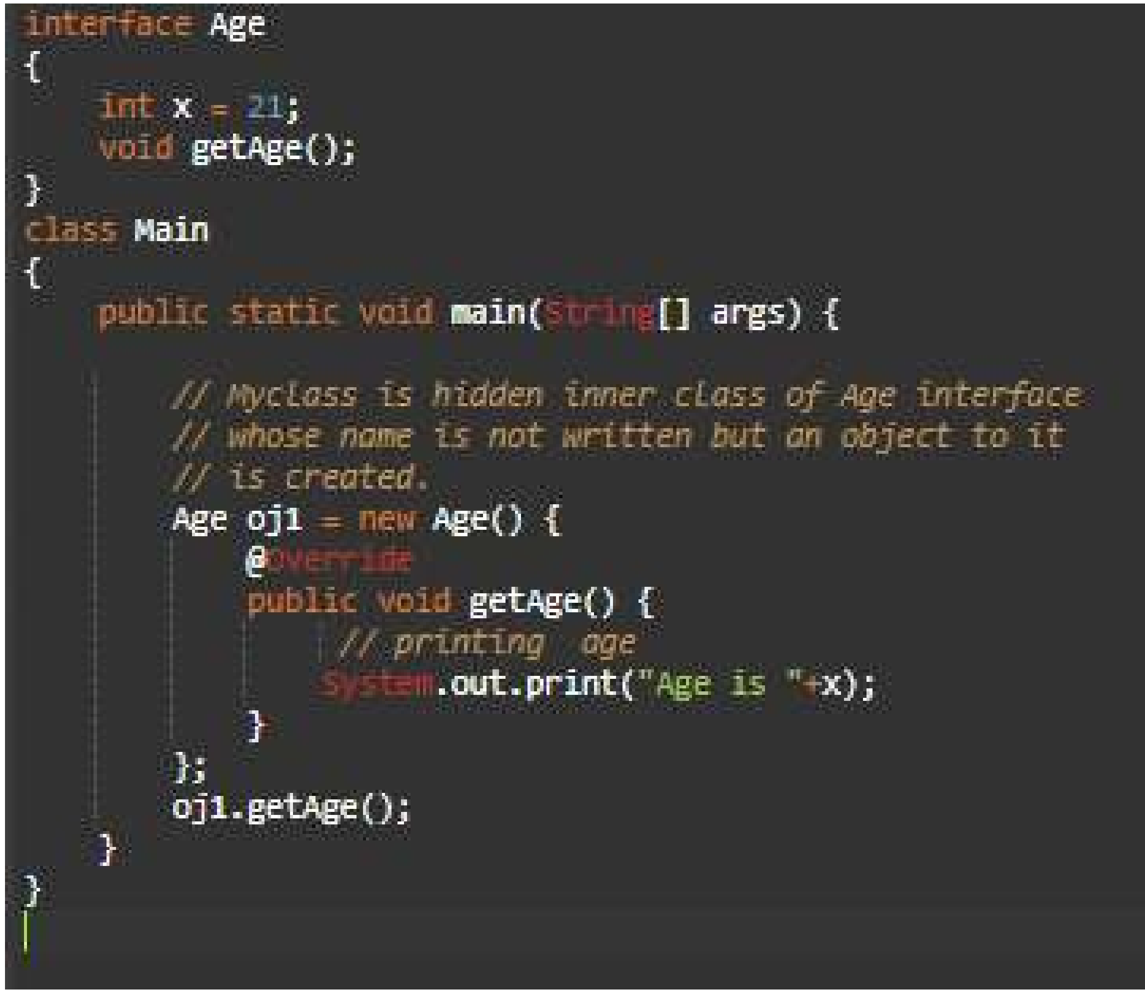
| Abstract class | Interface |
|---|---|
| 1) Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java 8, it can have default and static methods also. |
| 2) Abstract class doesn't support multiple inheritance. | Interface supports multiple inheritance. |
| 3) Abstract class can have final, non-final, static and non-static variables. | Interface has only static and final variables. |
| 4) Abstract class can provide the implementation of interface. | Interface can't provide the implementation of abstract class. |
| 5) The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| 6) An abstract class can extend another Java class and implement multiple Java interfaces. | An interface can extend another Java interface only. |
| 7) An abstract class can be extended using keyword "extends". | An interface can be implemented using keyword "implements". |
| 8) A Java abstract class can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)Example:<br>public abstract class Shape{<br>public abstract void draw();<br>} | Example:<br>public interface Drawable{<br>void draw(); |

## ANONYMOUS CLASS



```java
interface Age
{
    int x = 21;
    void getAge();
}
class Main
{
    public static void main(String[] args)
    {
        // Myclass is implementation class of Age interface
        MyClass obj=new MyClass();

        // calling getage() method implemented at Myclass
        obj.getAge();
    }
}

// Myclass implement the methods of Age Interface
class MyClass implements Age
{
    @Override
    public void getAge()
    {
        // printing the age
        System.out.print("Age is "+x);
    }
}
```

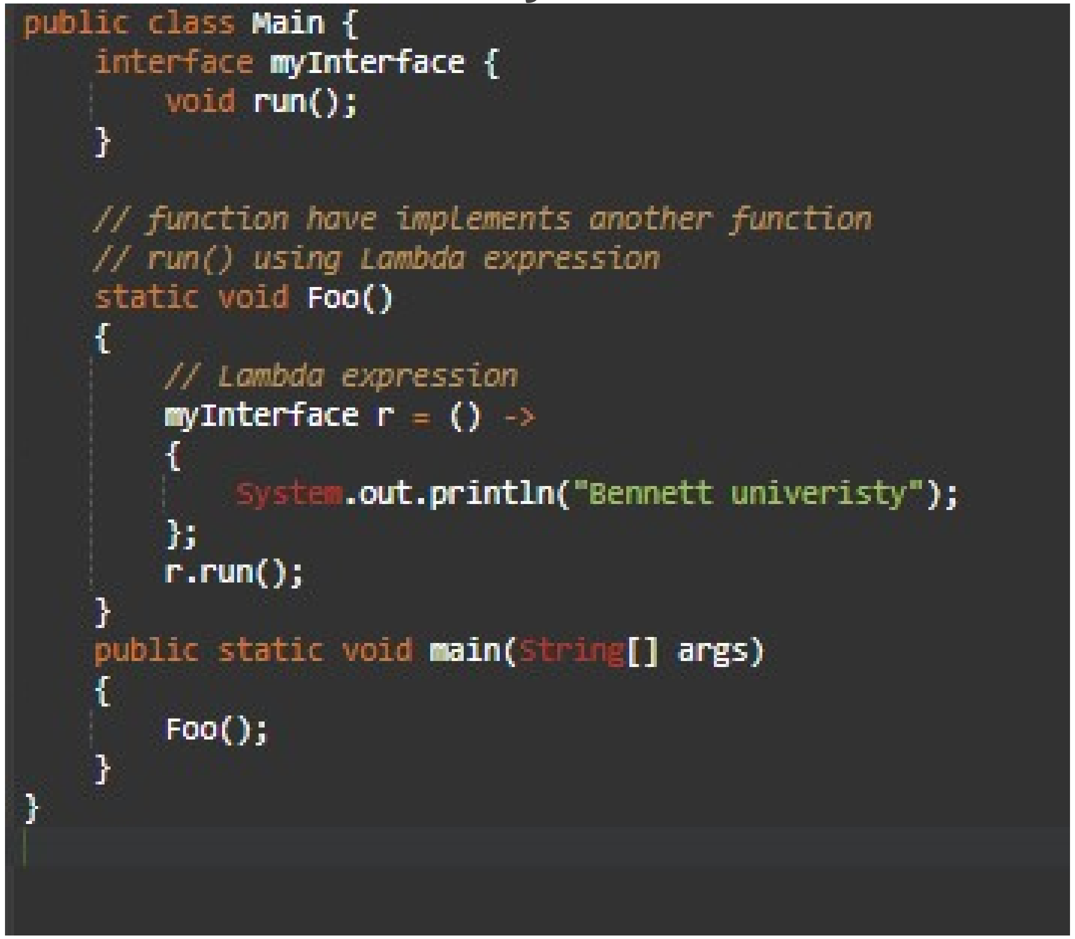
**APPROACH 1**



```java
interface Age
{
    int x = 21;
    void getAge();
}
class Main
{
    public static void main(String[] args) {

        // Myclass is hidden inner class of Age interface
        // whose name is not written but an object to it
        // is created.
        Age oj1 = new Age() {
            @Override
            public void getAge() {
                // printing  age
                System.out.print("Age is "+x);
            }
        };
        oj1.getAge();
    }
}
```

**APPROACH 2**

## DOES JAVA SUPPORTS METHOD INSIDE METHOD??



```
public class Main {
    interface myInterface {
        void run();
    }

    // function have implements another function
    // run() using Lambda expression
    static void Foo()
    {
        // Lambda expression
        myInterface r = () ->
        {
            System.out.println("Bennett univeristy");
        };
        r.run();
    }
    public static void main(String[] args)
    {
        Foo();
    }
}
```

**Lambda expressions** basically express instances of functional interfaces (An interface with single abstract method is called functional interface. An example is java.lang.Runnable). lambda expressions implement the only abstract function and therefore implement functional interfaces.

## MCQ

**Q: X extends y**

1 both x and y should be classes

2 both x and y should be interfaces

3 both x and y can be either classes/interfaces

4 no restriction

## MCQ

**Q: X extends y**

1 both x and y should be classes

2 both x and y should be interfaces

3 both x and y can be either classes/interfaces

4 no restriction

## MCQ

**Q: X extends y, z**

1 both x,y and z should be classes

2 both x,y and z should be interfaces

3 both x,y and z can be either classes/interfaces

4 no restriction

## MCQ

**Q: X extends y, z**

1 both x,y and z should be classes

2 both x,y and z should be interfaces

3 both x,y and z can be either classes/interfaces

4 no restriction

## MCQ

**Q: X implements  y, z**

1 both x,y and z should be classes

**2 both x,y and z should be interfaces**

3 both x,y and z can be either classes/interfaces

4 x should be a class and y,z should be interfaces

## MCQ

**Q: X implements  y, z**

1 both x,y and z should be classes

**2 both x,y and z should be interfaces**

3 both x,y and z can be either classes/interfaces

4 x should be a class and y,z should be interfaces

## MCQ

**Q: X extends y implements z**

1 both x,y and z should be classes

2 both x,y and z should be interfaces

3 both x,y and z can be either classes/interfaces

4 x  and y should be  classes and z should be interfaces

## MCQ

**Q: X extends y implements z**

1 both x,y and z should be classes

2 both x,y and z should be interfaces

3 both x,y and z can be either classes/interfaces

4 x  and y should be  classes and z should be interfaces

## MCQ

**Q: X implements y extends  z**

1 both x,y and z should be classes

2 both x,y and z should be interfaces

3 both x,y and z can be either classes/interfaces

4 x  and y should be  classes and z should be interfaces

5 compile time error

## MCQ

**Q: X implements y extends  z**

1 both x,y and z should be classes

2 both x,y and z should be interfaces

3 both x,y and z can be either classes/interfaces

4 x  and y should be  classes and z should be interfaces

5 compile time error

THANK YOU

?