## Interface

Interface just defines what a class must do without saying anything about its implementation.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface without going into its implementation details, which are left for the person implementing the interface.

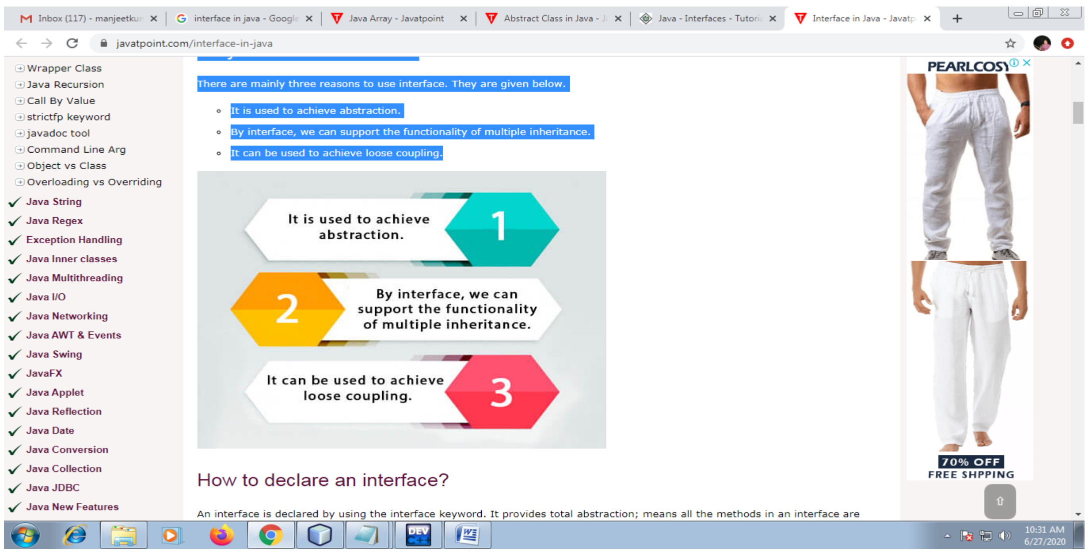
Interface is used to support the functionality of multiple inheritance.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

## Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.



There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

It is used to achieve abstraction. 1

2 By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling. 3

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are

Loose coupling is a design goal that seeks to reduce the inter-dependencies between components of the system with the goal of reducing the risk that the changes in one component will require changes in any other component. Loose coupling is much more generic concept intended to increase the flexibility of system, make it more maintainable and makes the entire framework more stable.

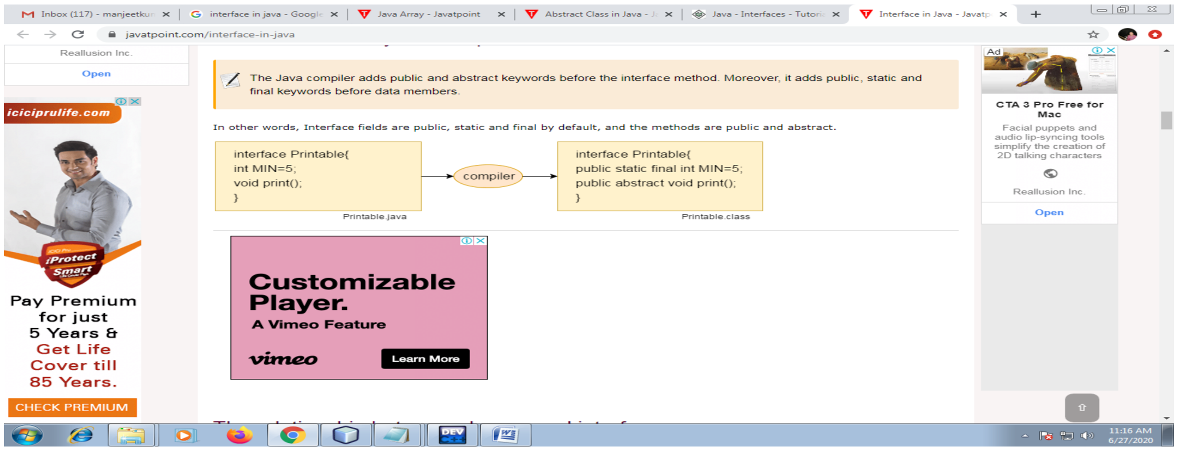
## Declare  interface

An interface is declared by using the interface keyword. A class that implements an interface must implement all the methods declared in the interface.

## Syntax:

```
interface  <interface_name>{



    // declare constant fields

    // declare methods that abstract

    // by default.

}
```

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


Inbox (117) - manjeetkun
interface in java - Google
Java Array - Javatpoint
Abstract Class in Java - J
Java - Interfaces - Tutoria
Interface in Java - Javatp
javatpoint.com/interface-in-java
Reallusion Inc.
Open
iciciprulife.com
iProtect Smart
Pay Premium for just 5 Years & Get Life Cover till 85 Years.
CHECK PREMIUM
Customizable Player.
A Vimeo Feature
vimeo
Learn More
Ad
CTA 3 Pro Free for Mac
Facial puppets and audio lip-syncing tools simplify the creation of 2D talking characters
Reallusion Inc.
Open
11:16 AM
6/27/2020

The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.

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

Printable.java

compiler

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

Printable.class

Eg:

In this example, the Printable interface has only one method, and its implementation is provided in the Test class.

```
interface Printable{

void display();

}
```

## Implementing Interfaces

When a class implements an interface, you can think of the class as signing a **contract**, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the “**implements”** keyword to implement an interface.

```
class  Test  implements  Printable{

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



public static void main(String args[]){

Test  obj = new Test  ();

obj.display();

 }

}
```

Eg:

```
interface Drawable{

void draw();

}


class Rectangle implements Drawable{

public void draw(){System.out.println("drawing rectangle");}

}

class Circle implements Drawable{

public void draw(){System.out.println("drawing circle");}

}

class TestInterface{

public static void main(String args[]){

Drawable d=new Circle();

d.draw();

}}
```

**Eg:**

```
import java.util.*;

interface Tax
{
final int trate=5;
void calculateTax();
}

class Book implements Tax
{
int bid, bprice;
int priceAfterTax;
public void set_data()
{
        Scanner scan=new Scanner(System.in);
        System.out.println("Enter Book ID:");
        bid=scan.nextInt();
        System.out.println("Enter Book Price:");
        bprice=scan.nextInt();
}
public void calculateTax()
{
priceAfterTax =bprice+(bprice*trate)/100;
}
void display()
{
System.out.println("Book Price:"+ bprice);
System.out.println("Price after Tax:"+ priceAfterTax);
}
public static void main(String args[])
{
Book b=new Book();
b.set_data();
b.calculateTax();
b.display();
}

}
```

## Multiple inheritance in Java by interface

If a class implements multiple interfaces, **( or )** one class and one interface, **( or )** an interface extends multiple interfaces, it is known as multiple inheritance.

Eg_1:

```
interface A{

void m1();

}

interface B{

void m2();

}

class TestMultipleInheritance implements A , B{

public void m1(){System.out.println("Hello");}
public void m2(){System.out.println("Welcome");}

public static void main(String args[]){

TestMultipleInheritance obj = new TestMultipleInheritance();

obj.m1();

obj.m2();

 }

}
```

**Eg_2. Develop a Java program to demonstrate the use of multiple inheritance.**

```
import java.util.*;

interface Tax

{

final int trate=5;

void calculateTax();

}

class Publication

{

   String pname;

   public Publication(String n)

   {

      pname=n;
```

```
    }

}

class Book extends Publication implements Tax

{

int bid, bprice;

int priceAfterTax;

 public Book(int id,String pn,int bp)

{

        super(pn);

            this.bid=id;

        this.bprice=bp;


}


public void calculateTax()

{

priceAfterTax =bprice+(bprice*trate)/100;

}

void display()

{

System.out.println("Book Id \t PublicationName \tBook Price \t Price_After_Tax" );

System.out.println(""+bid+ "\t" +pname+ "\t" + "\t\t"+ bprice+ "\t\t"+ priceAfterTax);
```

```
}

public static void main(String args[])

{

Book b=new Book(1001,"BPB Publication",100);

b.calculateTax();

b.display();

}


}
```

**Extending an Interface**

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

When a class implements an interface that inherits another interface, it must provide implementations for all the methods defined within the interface inheritance chain.

Eg:

```
interface A
{
void m1();
void m2();
}
interface B extends A
{
void m1();
}

class Test implements B
{

        public void m1()
        {
                System.out.println(“m1”);
        }

        public void m2()
        {
                System.out.println(“m2”);
        }

        public void m3()
        {
                System.out.println(“m3”);
        }

        Public static void main(String args[])
        {
                Test obj=new Test();
                obj.m1();
                obj.m2();
                obj.m3();
                t

        }

}
```

**Q:** Create **two interfaces** in java, **first interface** contain trate1 as attribute and calculateFirstTax as method and **second interface** contain trate2 as attribute and calculateSecondTax as method. Create Food class with proper attribute (i.e foodId, foodName and foodPrice) and method which implements above interface.

## Characteristics of Interface

1. Interface can be declared as abstract but it is seldom done.
2. Since interfaces are meant to be implemented by classes, interface members implicitly have public accessibility and the public modifier is omitted.
3. The methods are interface are all implicitly abstract and public. A method prototype has the same syntax as an abstract method.
4. Regardless of how many interfaces a class implements directly or indirectly, it only provides single implementations of a method that might have multiple declarations in the interface.
5. Method prototype declarations can also be overloaded as in the case of classes.