**Interface in Java**

**Extending, Implementing Interface**

An **interface in Java** is syntactically similar to a class but can have only abstract methods declaration and constants as members.

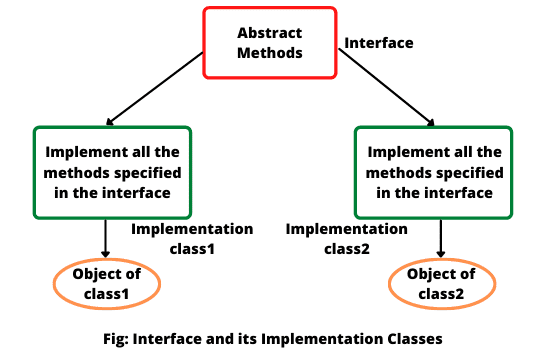
In other words, an interface is a collection of abstract methods and constants (i.e. static and final fields). It is used to achieve complete abstraction.

Every interface in java is abstract by default. So, it is not compulsory to write abstract keyword with an interface.

Once an interface is defined, we can create any number of separate classes and can provide their own implementation for all the abstract methods defined by an interface.

A class that implements an interface is called **implementation class**. A class can implement any number of interfaces in Java.

Every implementation class can have its own implementation for abstract methods specified in the interface as shown in the below figure.

.

Since the implementation classes will have all the methods with a body, it is possible to create an instance of implementation classes.

**Why do we use Interface?**

There are mainly five reasons or purposes of using an interface in Java. They are as follows:

1. In industry, architect-level people create interfaces, and then it is given to developers for writing classes by implementing interfaces provided.

2. Using interfaces is the best way to expose our project’s API to some other projects. In other words, we can provide interface methods to the third-party vendors for their implementation.

For example, HDFC bank can expose methods or interfaces to various shopping carts.

3. Programmers use interface to customize features of software differently for different objects.

4. It is used to achieve full abstraction in java.

5. By using interfaces, we can achieve the functionality of multiple inheritance.

**How to Declare Interface in Java?**

In Java, an interface is declared syntactically much like a class. It is declared by using the keyword interface followed by interface name. It has the following general form:

Syntax:

   accessModifier interface interfaceName

   {

     // declare constant fields.

     // declare methods that abstract by default.

  }

Before interface keyword, we can specify access modifiers such as public, or default with abstract keyword. Let’s understand the declaration of an interface with the help of an example.

public abstract interface MyInterfac

{

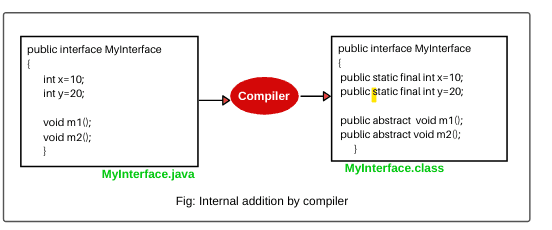
int x = 10; // public static final keyword invisibly present.

void m1(); // public and abstract keywords invisibly present.

  void m2();

}

As you can see in the above example, both methods m1() and m2() defined in interface are declared with no body and do not have public or abstract modifiers present. The variable x declared in MyInterface is like a simple variable.

Java compiler automatically adds public and abstract keywords before to all interface methods. Moreover, it also adds public, static, and final keywords before interface variables. Look at the below figure to understand better.  
  
Therefore, all the variables declared in an interface are considered as public, static, and final by default and acts like constant. We cannot change their value once they initialized.

**Note:**

a) Earlier to Java 8, an interface could not define any implementation whatsoever. An interface can only declare abstract methods.

b) Java 8 changed this rule. From Java 8 onwards, it is also possible to add a default implementation to an interface method.

c) To support lambda functions, Java 8 has added a new feature to interface. We can also declare default methods and static methods inside interfaces.

d) From Java 9 onwards, an interface can also declare private methods.

**Features of Interface**

There are following features of an interface in Java. They are as follows:

1. Interface provides pure abstraction in java. It also represents the Is-A relationship.

2. It can contain three types of methods: abstract, default, and static methods

3. All the (non-default) methods declared in the interface are by default abstract and public. So, there is no need to write abstract or public modifiers before them.

4. The fields (data members) declared in an interface are by default public, static, and final. Therefore, they are just public constants. So, we cannot change their value by implementing class once they are initialized.

5. Interface cannot have constructors.

6. The interface is the only mechanism that allows achieving multiple inheritance in java.

7. A Java class can implement any number of interfaces by using keyword implements.

8. Interface can extend an interface and can also extend multiple interfaces.

**Rules of Interface in Java**

Here are some key points for defining an interface in java that must be kept in mind. The rules are as follows:

1. An interface cannot be instantiated directly. But we can create a reference to an interface that can point to an object of any of its derived types implementing it.

2. An interface may not be declared with final keyword

3. It cannot have instance variables. If we declare a variable in an interface, it must be initialized at the time of declaration.

4. A class that implements an interface, must provide its own implementations of all the methods defined in the interface.

5. We cannot reduce the visibility of an interface method while overriding. That is, when we implement an interface method, it must be declared as public.

6. It can also be declared with empty body (i.e. without any members). For example, java.util package defines EventListener interface without a body.

7. An interface can be declared within another interface or class. Such interfaces are called nested interfaces in java.

8. A top-level interface can be public or default with the abstract modifier in its definition. Therefore, an interface declared with private, protected, or final will generate a compile-time error.

9. All non-default methods defined in an interface are abstract and public by default. Therefore, a method defined with private, protected, or final in an interface will generate compile-time error.

10. If you add any new method in interface, all concrete classes which implement that interface must provide implementations for newly added method because all methods in interface are by default abstract.

**Extending Interface in Java with Example**

Like classes, an interface can also extend another interface. This means that an interface can be sub interfaces from other interfaces.

The new sub-interface will inherit all members of the super interface similar to subclasses. It can be done by using the keyword “extends”. It has the following general form:

Syntax:

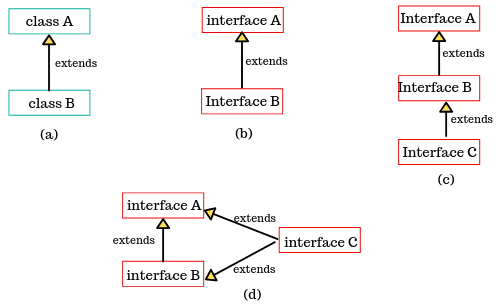
interface interfaceName2 extends interfaceName1

  {

    // body of interfaceName2.

}

Look at the below figure a, b, c, and d to understand better.



Let’s understand extending interface better with the help of different examples.

1. We can define all the constants into one interface and methods in another interface. We can use constants in classes where methods are not required. Look at the example below.

interface A

{

int x = 10;

int y = 20;

}

interface B extends A

{

void show();

}

The interface B would inherit both constants x and y into it.

2. We can also extend various interfaces together by a single interface. The general declaration is given below:

interface A

{

int x = 20;

int y = 30;

}

interface B extends A

{

void show();

}

interface C extends A, B

{

. . . . . . . .

}

**Key points:**

1. An interface cannot extend classes because it would violate rules that an interface can have only abstract methods and constants.  
2. An interface can extend Interface1, Interface2.

**Implementing Interface in Java with Example**

An interface is used as “superclass” whose properties are inherited by a class. A class can implement one or more than one interface by using a keyword implements followed by a list of interfaces separated by commas.

When a class implements an interface, it must provide an implementation of all methods declared in the interface and all its super interfaces.

Otherwise, the class must be declared abstract. The general syntax of a class that implements an interface is as follows:

Syntax:

1. accessModifier class className implements interfaceName

  {

    // method implementations;

    // member declaration of class;

   }

2. A more general form of interface implementation is given below.

  accessModifier class className extends superClass implements interface1, interface2,.. .

  {

     // body of className.

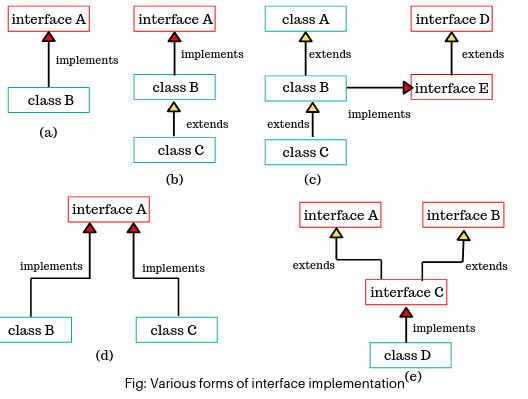
  }

This general form shows that a class can extend another class while implementing interfaces.

**Key points:**

1. All methods of interfaces when implementing in a class must be declared as public otherwise you will get a compile-time error if any other modifier is specified.  
2. Class extends class implements interface.  
3. Class extends class implements Interface1, Interface2…

The implementation of interfaces can have the following general forms as shown in the below figure.



**Class ---extends-- class**

**Class -- implements --interface**

**Interface---extends—interface**

**Interface –implements--class**

**Accessing Interface Variable in Java**

The interface is also used to declare a set of constants that can be used in multiple classes. The constant values will be available to any classes that implement interface because it is by default public, static, and final.

The values can also be used in any method as part of the variable declaration or anywhere in the class.

Let’s take various types of example programs related to all important concepts of interface.

**Java Interface Example Programs**

1. Let’s create a program where multiple classes implement the same interface to use constant values declared in that interface.

**Program source code 1:**

package interfacePrograms;

public interface ConstantValues

{

// Declaration of interface variables.

int x = 20;

int y = 30;

}

public class Add implements ConstantValues

{

int a = x;

int b = y;

void m1()

{

System.out.println("Value of a: " +a);

System.out.println("Value of b: " +b);

}

void sum()

{

int s = x + y;

System.out.println("Sum: " +s);

}

}

public class Sub implements ConstantValues

{

void sub()

{

int p = y - x;

System.out.println("Sub: " +p);

}

}

public class Test

{

public static void main(String[] args)

{

Add a = new Add();

a.m1();

a.sum();

Sub s = new Sub();

s.sub();

}

}

Output:

Value of a: 20

Value of b: 30

Sum: 50

Sub: 10

2. Let’s take an example program where class B implements an interface A.

**Program source code 2:**

package interfacePrograms;

public interface A

{

void msg(); // No body.

}

public class B implements A

{

// Override method declared in interface.

public void msg()

{

System.out.println("Hello Java");

}

void show()

{

System.out.println("Welcome you");

}

public static void main(String[] args)

{

B b = new B();

b.msg();

b.show(); // A reference of interface is pointing to objects of class B.

A a = new B();

a.msg();

// a.show(); // Compile-time error because a reference of interface can only call methods declared in it and implemented by implementing class.

// show() method is not part of interface. It is part of class B. When you will call this method, the compiler will give a compile-time error. It can only be called when you create an object reference of class B.

}

}

Output:

Hello Java

Welcome you

Hello Java

**Polymorphism in Java Interface**

Let’s create a program where multiple classes implement the same interface.

When two or more classes implement the same interface with different implementations then through the object of each class, we can achieve polymorphic behavior for a given interface. This is called polymorphism in interface.

In the above figure, (d) shows polymorphism in interfaces where class B and class C implement the same interface A.\

**Program source code 3:**

package interfacePrograms;

public interface Area

{

float pi = 3.14f; // Constant declaration.

float calculateArea(float x, float y);

}

public class Circle implements Area

{

public float calculateArea(float x, float y)

{

float areaOfCircle = pi \* x \* y; // Implementation.

return areaOfCircle;

}

}

public class Square implements Area

{

public float calculateArea(float x, float y)

{

float areaOfSquare = x \* y; // Implementation.

return areaOfSquare;

}

}

public class InterfaceTest

{

public static void main(String[] args)

{

Area a; // Creating interface reference.

a = new Circle(); // Creating object of circle.

float circle = a.calculateArea(20, 10.5f);

System.out.println("Area of circle: " +circle);

a = new Square(); // Creating object of square.

float square = a.calculateArea(20.5f, 10.5f);

System.out.println("Area of square: " +square);

}

}

Output:

Area of circle: 659.4

Area of square: 215.25

**Multilevel Inheritance by Interface**

Let’s take an example program where we will create a multilevel inheritance by interface. One interface extends an interface, that interface extends another interface, and a class implements methods of all interfaces, we can achieve multilevel inheritance by interfaces.

**Program source code 4:**

package interfacePrograms;

public interface Continent

{

void showContinent();

}

public interface Country

{

void showCountry();

}

public interface State

{

void showState();

}

public class City implements State

{

public void showContinent()

{

System.out.println("Asia");

}

public void showCountry()

{

System.out.println("India");

}

public void showState()

{

System.out.println("Jharkhand");

}

void showCity()

{

System.out.println("Dhanbad");

}

public static void main(String[] args)

{

City c = new City();

c.showContinent();

c.showCountry();

c.showState();

c.showCity();

}

}

Output:

Asia

India

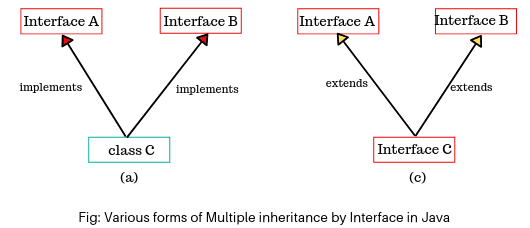
Jharkhand

Dhanbad

As you can observe that class City implements interface State. The interface State is inherited from interface Country and Country is inherited from interface Continent.

A class City also implements Country and Continent interfaces even though they are not explicitly included after the colon in the declaration of class City. The methods of all interfaces have been implemented in class City.

**Multiple Inheritance in Java by Interface**

When a class implements more than one interface, or an interface extends more than one interface, it is called multiple inheritance. Various forms of multiple inheritance are shown in the following figure.  
  
Let’s create a program to achieve multiple inheritance using multiple interfaces.

**Program source code 5:**

package multipleInheritancebyInterfaces;

public interface Home

{

void homeLoan();

}

public interface Car

{

void carLoan();

}

public interface Education

{

void educationLoan();

}

public class Loan implements Home, Car, Education

{

// Multiple inheritance using multiple interfaces.

public void homeLoan()

{

System.out.println("Rate of interest on home loan is 8.5%");

}

public void carLoan()

{

System.out.println("Rate of interest on car loan is 9.25%");

}

public void educationLoan()

{

System.out.println("Rate of interest on education loan is 10.45%");

}

public static void main(String[] args)

{

Loan l = new Loan();

l.homeLoan();

l.carLoan();

l.educationLoan();

}

}

Output:

Rate of interest on home loan is 8.5%

Rate of interest on car loan is 9.25%

Rate of interest on education loan is 10.45%

The above program contains three interfaces such as Home, Car, and Education. The class Loan inherits these three interfaces. Thus, we can achieve multiple inheritance in java through interfaces.

**In Java, Multiple Inheritance is not supported through Class but it is possible by Interface, why?**

As we have explained in the inheritance chapter, in multiple inheritance, subclasses are derived from multiple superclasses.

If two superclasses have the same method name then which method is inherited into subclass is the main confusion in multiple inheritance.

That’s why Java does not support multiple inheritance in case of class. But, it is supported through an interface because there is no confusion. This is because its implementation is provided by the implementation class.

Let’s understand it with the help of an example program.

**Program source code 6:**

package multipleInheritancebyInterface;

public interface AA

{

void m1();

}

public interface BB

{

void m1();

}

public class Myclass implements AA, BB

{

public void m1()

{

System.out.println("Hello Java");

}

public static void main(String[] args)

{

Myclass mc = new Myclass();

mc.m1();

}

}