# Interfaces in Java



An **Interface in Java** programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behaviour. A Java interface contains static constants and abstract methods.

The interface in Java is *a* mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**.

When we decide a type of entity by its behaviour and not via attribute we should define it as an interface.

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the behaviour.
- Interface do not have constructor.
- Represent behaviour as interface unless every sub-type of the class is guarantee to have that behaviour.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
- A Java library example is Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

**Syntax:**

```
interface {

    // declare constant fields
    // declare methods that abstract
    // by default.
```

To declare an interface, use the interface keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static, and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use implements keyword.

**Why do we use an Interface?**

- It is used to achieve total abstraction.
- Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.
- Any class can extend only 1 class but can any class implement infinite number of interface.
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

The reason is, abstract classes may contain non-final variables, whereas variables in the interface are final, public and static.

```
// A simple interface

interface Player
{
    final int id = 10;
```

## Difference Between Class and Interface

The major differences between a class and an interface are:

| S. No. | Class | Interface |
|---|---|---|
| 1. | In class, you can instantiate variables and create an object. | In an interface, you can't instantiate variables and create an object. |
| 2. | Class can contain concrete(with implementation) methods | The interface cannot contain concrete(with implementation) methods |
| 3. | The access specifiers used with classes are private, protected, and public. | In Interface only one specifier is used- Public. |

**Implementation:** To implement an interface we use the keyword **implements**

---

### Java

```
// Java program to demonstrate working of
// interface
```

```
interface In1 {

    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}

// A class that implements the interface.
class TestClass implements In1 {

    // Implementing the capabilities of
    // interface.
    public void display(){
      System.out.println("Geek");
    }

    // Driver Code
    public static void main(String[] args)
    {
        TestClass t = new TestClass();
        t.display();
        System.out.println(a);
    }
}
```

## Output

```
Geek
```

**Real-World Example:** Let's consider the example of vehicles like bicycle, car, bike....., they have common functionalities. So we make an interface and put all these common functionalities. And lets Bicycle, Bike, car ....etc implement all these functionalities in their own class in their own way.

---

## Java

```
// Java program to demonstrate the
// real-world example of Interfaces

import java.io.*;

interface Vehicle {

    // all are the abstract methods.
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}

class Bicycle implements Vehicle{

    int speed;
    int gear;

    // to change gear
    @Override
    public void changeGear(int newGear){

        gear = newGear;
```

```
    // to increase speed
    @Override
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    @Override
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }
}

class Bike implements Vehicle {

    int speed;
    int gear;

    // to change gear
    @Override
    public void changeGear(int newGear){

        gear = newGear;
    }
```

```
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    @Override
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }

}
class GFG {

    public static void main (String[] args) {

        // creating an instance of Bicycle
        // doing some operations
        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);

        System.out.println("Bicycle present state :");
        bicycle.printStates();
```

```
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);

        System.out.println("Bike present state :");
        bike.printStates();
    }
}
```

## Output

```
Bicycle present state :
speed: 2 gear: 2
Bike present state :
speed: 1 gear: 1
```

### Advantages of Interfaces in Java

The advantages of using interfaces in Java are as follows:

1. Without bothering about the implementation part, we can achieve the security of the implementation.
2. In Java, multiple inheritances is not allowed, however, you can use an interface to make use of it as you can implement more than one interface.

## New Features Added in Interfaces in JDK 8

**1.** Prior to JDK 8, the interface could not define the implementation. We can now add default implementation for interface methods. This default implementation has a special use and does not affect the intention behind interfaces.

Suppose we need to add a new function in an existing interface. Obviously, the old code will not work as the classes have not implemented those new functions. So with the help of default implementation, we will give a default body for the newly added functions. Then the old codes will still work.

---

## Java

```
// Java program to show that interfaces can
// have methods from JDK 1.8 onwards

interface In1
{
    final int a = 10;
    default void display()
    {
        System.out.println("hello");
    }
}

// A class that implements the interface.
class TestClass implements In1
{
    // Driver Code
    public static void main (String[] args)
    {
```

```
    }
}
```

### Output

```
hello
```

**2.** Another feature that was added in JDK 8 is that we can now define static methods in interfaces that can be called independently without an object. Note: these methods are not inherited.

---

## Java

```java
// Java Program to show that interfaces can
// have methods from JDK 1.8 onwards

interface In1
{
    final int a = 10;
    static void display()
    {
        System.out.println("hello");
    }
}

// A class that implements the interface.
class TestClass implements In1
{
    // Driver Code
```

```
        In1.display();
    }
}
```

### Output

```
hello
```

### Extending Interfaces

One interface can inherit another by use of keyword extends. When a class implements an interface that inherit another interface, it must provide implementation for all method required by the interface inheritance chain.

---

## Java

```java
interface A {
    void method1();
    void method2();
}
// B now includes method1 and method2
interface B extends A {
    void method3();
}
// the class must implement all method of A and B.
class gfg implements B {
```

```
    }
    public void method2()
    {
        System.out.println("Method 2");
    }
    public void method3()
    {
        System.out.println("Method 3");
    }
}
```

### Example 3:

## Java

```
interface Student
{
    public void data();

}
class avi implements Student
{
    public void data ()
    {
        String name="avinash";
        int rollno=68;
        System.out.println(name);
        System.out.println(rollno);
    }
```

```
{
    public static void main (String args [])
    {
        avi h= new avi();
        h.data();
    }
}
```

### Output

```
avinash
68
```

In Simple way,The interface contains multiple abstract methods, so write the implementation in implementation classes. If the implementation is unable to provide implementation of all abstract methods, then declare the implementation class with abstract modifier, and complete the remaining method implementation in next created child classes. It is possible to declare multiple child classes but at final we have complete the implementation of all abstract methods.

In general the development process is as steps by step:

1. Level 1- interfaces : It contains the service details.
2. Level 2 – abstract classes : It contains partial implementation.
3. Level 3 – implementation classes : It contains all implementation.

E.g.

## Java

```
// Java Program for
// implementation Level wise
import java.io.*;
import java.lang.*;
import java.util.*;

// Level 1
interface Bank {
    void deposit();
    void withdraw();
    void loan();
    void account();
}

// Level 2
abstract class Dev1 implements Bank {
    public void deposit()
    {
        System.out.println("Your deposit Amount :" + 100);
    }
}

abstract class Dev2 extends Dev1 {
    public void withdraw()
    {
        System.out.println("Your withdraw Amount :" + 50);
```

```
// Level 3
class Dev3 extends Dev2 {
    public void loan() {}
    public void account() {}
}

// Level 4
class GFG {
    public static void main(String[] args)
    {
        Dev3 d = new Dev3();
        d.account();
        d.loan();
        d.deposit();
        d.withdraw();
    }
}
```

### Output

```
Your deposit Amount :100
Your withdraw Amount :50
```

### Important Points About Interface or Summary of the Article:

- We can't create an instance(interface can't be instantiated) of the interface but we can make the reference of it that refers to the Object of its implementing class

- An interface can extend to another interface or interface (more than one interface).
- A class that implements the interface must implement all the methods in the interface.
- All the methods are public and abstract. And all the fields are public, static, and final.
- It is used to achieve multiple inheritances.
- It is used to achieve loose coupling.
- Inside the Interface not possible to declare instance variables because by default variables are **public static final.**
- Inside the Interface constructors are not allowed.
- Inside the interface main method is not allowed.
- Inside the interface static ,final,private methods declaration are not possible.

**New Features Added in Interfaces in JDK 9**

From Java 9 onwards, interfaces can contain the following also:

1. Static methods
2. Private methods
3. Private Static methods

**Related Articles:**

- Access Specifier of Methods in Interfaces
- Access Specifiers for Classes or Interfaces in Java
- Abstract Classes in Java
- Comparator Interface in Java
- Java Interface Methods

This article is contributed by **Mehak Kumar** and **Nitsdheerendra.** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 30 Mar, 2023

380

## Similar Reads

1. Access modifiers for classes or interfaces in Java
2. Callback using Interfaces in Java
3. Private Methods in Java 9 Interfaces
4. Why Java Interfaces Cannot Have Constructor But Abstract Classes Can Have?
5. Generic Constructors and Interfaces in Java
6. Interfaces and Polymorphism in Java
7. Which Java Types Can Implement Interfaces?
8. Match Lambdas to Interfaces in Java

10. Interfaces and Inheritance in Java

## Related Tutorials

1. Spring MVC Tutorial
2. Spring Tutorial
3. Spring Boot Tutorial
4. Java 8 Tutorial
5. CBSE Class 11 Syllabus

Previous

**Polymorphism in Java**

Next

**'this' reference in Java**

### Article Contributed By :

GeeksforGeeks

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

**Improved By :** RohitKumar8, RAraghavarora, bipin_kumar, iamthecoder, shvrekhan, simmytarika5, nishkarshgandhi, krnyele, mitalibhola94, rathoadavinash, kunal2144

**Article Tags :** java-interfaces, Java, School Programming

**Practice Tags :** Java

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Backtracking

## Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

## Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

## Competitive Programming

NodeJS

## Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

## DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

## System Design

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

## Interview Corner

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

## Commerce

Accountancy

Business Studies

Microeconomics

Macroeconomics

Statistics for Economics

Indian Economic Development

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

## GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

## UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

## SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

## Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship