<https://www.geeksforgeeks.org/interfaces-in-java/>

# Java implements Keyword

[❮ Java Keywords](https://www.w3schools.com/java/java_ref_keywords.asp)

Examples of interface in Selenium are WebDriver, WebElement

WebDriver is a remote control interface that enables introspection and control of user agents (browsers). **[WebElement](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/WebElement.html" \o "interface in org.openqa.selenium)** Represents an HTML element.

**[TakesScreenshot](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/TakesScreenshot.html" \o "interface in org.openqa.selenium)** Indicates a driver or an HTML element that can capture a screenshot and store it in different ways.

### **Example**

An interface is an abstract "class" that is used to group related methods with "empty" bodies:

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

// interface

interface Animal {

public void animalSound(); // interface method (does not have a body)

public void animalLegs (); // interface method (does not have a body)

}

// Pig "implements" the Animal interface

class Pig **implements** Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The pig says: wee wee");

}

public void animalLegs() {

// The body of animalLegs() is provided here

System.out.println("The pig has four legs");

}

}

class MyMainClass {

public static void main(String[] args) {

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound();

myPig.animalLegs();

}

}

[Run example »](https://www.w3schools.com/java/showjava.asp?filename=demo_interface" \t "_blank)

We use the interface [keyword](https://www.programiz.com/java-programming/keywords-identifiers) to create an interface in Java. For example,

interface Language {

public void getType();

public void getVersion();

}

Here,

Language is an interface.

It includes abstract methods: getType() and getVersion().

**Implementing an Interface**

Like abstract classes, we cannot create objects of interfaces.

To use an interface, other classes must implement it. We use the implements keyword to implement an interface.

**Example 1: Java Interface**

interface Polygon {

void getArea(int length, int breadth);

}

// implement the Polygon interface

class Rectangle implements Polygon {

// implementation of abstract method

public void getArea(int length, int breadth) {

System.out.println("The area of the rectangle is " + (length \* breadth));

}

}

class Main {

public static void main(String[] args) {

Rectangle r1 = new Rectangle();

r1.getArea(5, 6);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler" \t "_blank)

**Output**

The area of the rectangle is 30

In the above example, we have created an interface named Polygon. The interface contains an abstract method getArea().

Here, the Rectangle class implements Polygon. And, provides the implementation of the getArea() method.

## Definition and Usage

The implements keyword is used to implement an [interface](https://www.w3schools.com/java/ref_keyword_interface.asp).

The interface keyword is used to declare a special type of class that only contains abstract methods.

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class.

#### **Notes on Interfaces:**

* It **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
* Interface methods does not have a body - the body is provided by the "implement" class
* On implementation of an interface, you must override all of its methods
* Interface methods are by default abstract and public
* Interface attributes are by default public, static and final
* An interface cannot contain a constructor (as it cannot be used to create objects)

#### **Why And When To Use Interfaces?**

To achieve security - hide certain details and only show the important details of an object (interface).

Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma (see example below).

## Multiple Interfaces

To implement multiple interfaces, separate them with a comma:

### **Example**

interface FirstInterface {

public void myMethod(); // interface method

}

interface SecondInterface {

public void myOtherMethod(); // interface method

}

// DemoClass "implements" FirstInterface and SecondInterface

class DemoClass implements **FirstInterface, SecondInterface** {

public void myMethod() {

System.out.println("Some text..");

}

public void myOtherMethod() {

System.out.println("Some other text...");

}

}

class MyMainClass {

public static void main(String[] args) {

DemoClass myObj = new DemoClass();

myObj.myMethod();

myObj.myOtherMethod();

}

}

Interfaces in Java

Like a class, an interface can have methods and variables, but the methods declared in 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 class.
* 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 class must be declared abstract.
* A Java library example is, [Comparator Interface](https://www.geeksforgeeks.org/comparator-interface-java/). If a class implements this interface, then it can be used to sort a collection.

**Syntax :**

interface <interface\_name> {

// declare constant fields

// declare methods that abstract

// by default.

}

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

**Why do we use interface ?**

* It is used to achieve total abstraction.
* Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
* 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 interface are final, public and static.

|  |
| --- |
| // A simple interface  interface Player  {      final int id = 10;      int move();  } |

To implement an interface we use keyword: implement

|  |
| --- |
| // Java program to demonstrate working of  // interface.  import java.io.\*;    // A simple interface  interface in1  {      // public, static and final      final int a = 10;        // public and abstract      void display();  }    // A class that implements 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

10

**A 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 Bicylce, Bike, car ….etc implement all these functionalities in their own class in their own way.

|  |
| --- |
| 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;      }        // 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 GFG {        public static void main (String[] args) {            // creating an inatance 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();            // creating instance of bike.          Bike bike = new Bike();          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

**New features added in interfaces in JDK 8**

1. Prior to JDK 8, interface could not define implementation. We can now add default implementation for interface methods. This default implementation has 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.

|  |
| --- |
| // An example 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 interface.  class testClass implements in1  {      // Driver Code      public static void main (String[] args)      {          testClass t = new testClass();          t.display();      }  } |

Output :

hello

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

|  |
| --- |
| // An example 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 interface.  class testClass implements in1  {      // Driver Code      public static void main (String[] args)      {          in1.display();      }  } |

Output :

hello

**Important points about interface or summary of article:**

* We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
* A class can implement more than one interface.
* An interface can extends another interface or interfaces (more than one interface) .
* A class that implements interface must implements all the methods in interface.
* All the methods are public and abstract. And all the fields are public, static, and final.
* It is used to achieve multiple inheritance.
* It is used to achieve loose coupling.

**New features added in interfaces in JDK 9**  
From Java 9 onwards, interfaces can contain following also

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

**Related articles:**

* [Access specifier of methods in interfaces](https://www.geeksforgeeks.org/g-fact-73/)
* [Access specifiers for classes or interfaces in Java](https://www.geeksforgeeks.org/g-fact-81/)
* [Abstract Classes in Java](https://www.geeksforgeeks.org/abstract-classes-in-java/)
* [Comparator Interface in Java](https://www.geeksforgeeks.org/comparator-interface-java/" \o "Permalink to Comparator Interface in Java)
* [Java Interface methods](https://www.geeksforgeeks.org/g-fact-47-java-interface-methods/" \o "Permalink to G-Fact 47 | (Java Interface methods))
* [Nested Interface in Java](https://www.geeksforgeeks.org/interface-nested-class-another-interface/" \o "Permalink to Nested Interface in Java)

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

**Recommended Posts:**

* [Interfaces in Java](https://www.geeksforgeeks.org/interfaces-in-java-unpublished/)
* [Interfaces and Inheritance in Java](https://www.geeksforgeeks.org/interfaces-and-inheritance-in-java/)
* [Callback using Interfaces in Java](https://www.geeksforgeeks.org/callback-using-interfaces-java/)
* [Functional Interfaces In Java](https://www.geeksforgeeks.org/functional-interfaces-java/)
* [Private Methods in Java 9 Interfaces](https://www.geeksforgeeks.org/private-methods-java-9-interfaces/)
* [Access specifiers for classes or interfaces in Java](https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/)
* [Access specifier of methods in interfaces](https://www.geeksforgeeks.org/g-fact-73/)
* [Two interfaces with same methods having same signature but different return types](https://www.geeksforgeeks.org/two-interfaces-methods-signature-different-return-types/)
* [Java.util.LinkedList.poll(), pollFirst(), pollLast() with examples in Java](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-examples-java/)
* [Java.lang.Short toString() method in Java with Examples](https://www.geeksforgeeks.org/java-lang-short-tostring-method-in-java-with-examples/)
* [Java.util.Collections.rotate() Method in Java with Examples](https://www.geeksforgeeks.org/java-util-collections-rotate-method-java-examples/)
* [Java lang.Long.byteValue() method in Java with Examples](https://www.geeksforgeeks.org/java-lang-long-bytevalue-method-java-examples/)
* [Java lang.Long.highestOneBit() method in Java with Examples](https://www.geeksforgeeks.org/java-lang-long-highestonebit-method-java-examples/)
* [Java.util.concurrent.Phaser class in Java with Examples](https://www.geeksforgeeks.org/java-util-concurrent-phaser-class-in-java-with-examples/)
* [Java.util.function.BiPredicate interface in Java with Examples](https://www.geeksforgeeks.org/java-util-function-bipredicate-interface-in-java-with-examples/)
* Java does not support multiple inheritances but we can achieve the effect of multiple inheritances using interfaces. In interfaces, a class can implement more than one interface which can’t be done through extends keyword.  
  Please refer [Multiple inheritance in java](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) for more.  
  Let’s say we have two interfaces with same method name (geek) and different return types(int and String)

|  |
| --- |
| public interface InterfaceX  {      public int geek();  }  public interface InterfaceY  {      public String geek();  } |

* Now, Suppose we have a class that implements both those interfaces:

|  |
| --- |
| public class Testing implements InterfaceX, InterfaceY  {  public String geek()      {          return "hello";      }  } |

* The question is: **Can a java class implement Two interfaces with same methods having the same signature but different return types??**  
  **No, its an error**  
  If two interfaces contain a method with the same signature but different return types, then it is impossible to implement both the interface simultaneously.