**What Is An Interface In Java**

***An interface in Java is defined as an abstract type that specifies class behavior.*** An interface is a kind of a protocol that sets up rules regarding how a particular class should behave.

An interface in Java can contain abstract methods and static constants. By default, all the methods in the interface are public and abstract.

**A simple example of an interface in Java is given below.**

interface shape{

public static final String color = “Red”;

public void calculateArea();

}

The above example defines an interface ‘shape’ which has a static variable and an abstract method ‘calculateArea ()’.

An interface is an entity that has only abstract methods as its body. It can also have static final variables in it.

So just like class, an interface can also have methods and variables but note that the methods are abstract (without implementation) and variables are static.

**Enlisted below are some properties that should be kept in mind related to Interfaces:**

* Interfaces are blueprints for a class. They tell the class what to do through their methods.
* An interface specifies abstract methods and classes implementing that interface should also implement those methods.
* If a class implementing the interface does not define all the methods of the interface, then that class becomes an abstract class.

**The general syntax of the interface declaration is given below.**

interface <interface\_name>{

//constant or static fields declaration

//abstract method declaration

//default declarations

}

As shown in the above declaration, we use a Java keyword “interface” which indicates that we are declaring an interface now.

An ‘interface’ keyword is followed by the interface\_name and then the opening curly braces. Then we have various declarations of abstract methods, static fields declaration, etc. Finally, we close the curly braces.

**For example, if we want to declare an interface ‘TestInterface’ with two methods in it i.e. method\_one and method\_two then the declaration of TestInterface will be as below:**

interface TestInterface{

           void method\_one();

           void method\_two();

}

**Uses of the Interface In Java**

* Interfaces in Java provide 100% abstraction as they can have only abstract methods.
* Using interfaces, we can achieve multiple inheritances in Java which is not possible using classes.
* To achieve loose coupling, an interface can be used.

**How To Implement An Interface In Java**

Once the interface is declared, we can use it in a class **using the “implements” keyword** in the class declaration.

**This ‘implements’ keyword appears after the class name as shown below:**

class <class\_name> implements <interface\_name>{

//class body

}

Implementing an interface is the same as signing a contract. Hence a class implementing an interface means that it has signed a contract and has agreed to implement the abstract methods of the interface or in other words perform the behavior specified by the interface.

If the class implementing the interface does not implement the exact behavior specified in the interface then the class needs to be declared as abstract.

Interface Implementation Example

**Given below is a simple example of an interface in Java.**

|  |
| --- |
| //interface declaration  **interface** Polygon\_Shape {  **void** calculateArea(**int** length, **int** breadth);  }    //implement the interface  **class** Rectangle **implements** Polygon\_Shape {      //implement the interface method  **public** **void** calculateArea(**int** length, **int** breadth) {          System.out.println("The area of the rectangle is " + (length \* breadth));      }  }    **class** Main {  **public** **static** **void** main(String[] args) {          Rectangle rect = **new** Rectangle();       //declare a class object          rect.calculateArea(10, 20);             //call the method      }  } |

**Output:**

[Area of rectangle example output](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/Java-interface-implementation-example.png)

The above program demonstrates the simple example of interfaces in Java. Here, we declare an interface named Polygon\_Shape and then the class Rectangle implements it.

**Interface Naming Convention In Java**

Java naming conventions are the naming guidelines that we have to follow as programmers so that we can produce readable consistent code. Java uses “TitleCase” notations for the naming classes and interfaces. It uses “CamelCase” notations for variables, methods, etc.

As far as interface is concerned, the interface name is in the titlecase with the first letter of every word of the interface name capitalized. Interface names are selected such that they are usually adjectives. But when interfaces represent the family of classes like map or list, then they can be named after nouns.

**Some examples of valid interface names are given below:**

public interface Iterable {}

public interface List {}

public interface Serializable {}

public interface Clonable {}

public interface Runnable {}

**Interface Constructor**

***The next question is whether an interface has a constructor?***

We know that we need objects to invoke methods. To create objects we need constructors. But in the case of Interfaces in Java, the methods are not implemented.

The methods of interfaces are all abstract. Hence there is no use in calling these methods from the interface. Secondly, as interfaces are by default abstract, we cannot create objects of the interface. **Thus we do not need constructors for Interface.**

**Interface Methods**

In this section, we will discuss how to declare interface methods. By rule, an interface can have only public methods or by default, interface methods are public. No other access modifier is allowed to be used inside the interface.

So whether we explicitly declare it or not, every method in the interface is by default abstract with public visibility.

Hence if void printMethod() is the prototype that we intend to declare in an interface, then the following declarations are the same.

void printMethod();

public void printMethod();

abstract void printMethod ();

public abstract void printMethod ();

**Note that we cannot use the following modifiers inside the interface for the interface methods.**

* final
* static
* Private
* protected
* synchronized
* native
* strictfp

**Now let’s implement a Java program to demonstrate the interface method visibility.**

|  |
| --- |
| //declare an interface  **interface** TestInterface  {  **void** printMethod(); //default visibility is public.  }  //interface implementation  **class** TestClass **implements** TestInterface  {      //if the access modifier is changed to any other, compiler generates error  **public** **void** printMethod()      {          System.out.println("TestClass::printMethod()");      }  }    **class** Main  {  **public** **static** **void** main(String[] args)      {          TestClass tc = **new** TestClass();     //create an object          tc.printMethod();                   //call concrete method      }  } |

**Output:**

[TestClass printMethod](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/TestClass-printMethod.png)

As already mentioned, by default, the interface methods are public. Hence when we do not specify any access modifier for the interface method, then it is public as in the above program.

**Suppose we change the interface method declaration in the above program as follows:**

*private void printMethod();*

Then this means that we specified the interface method printMethod () as private. When we compile the program, we get the following compiler error.

*error: modifier private not allowed here*

*private void printMethod();*

The second case we can test is by changing the modifier of the implemented method in the class TestClass from public to private. Now the default modifier in the class is private. So we just remove the public keyword from the method prototype in the class as follows:

void printMethod()

{

System.out.println("TestClass::printMethod()");

}

**Now if we compile the program, then we get the following error.**

*error: printMethod() in TestClass cannot implement printMethod() in TestInterface*  
*void printMethod()*  
*^*  
*attempting to assign weaker access privileges; was public*

Hence the point to be noted here is that we cannot change the access modifier of the implemented method of the interface to any other access modifier. As the interface methods are by default public, when they are implemented by classes that implement interfaces, these methods should also be public.

**Interface Fields In Java**

The fields or variables declared in an interface are by default public, static, and final. This means that once declared their value cannot be changed.

Note that if the interface fields are defined without specifying any of these modifiers then Java compilers assume these modifiers. **For example,** if we do not specify a public modifier when declaring the field in the interface, then it is assumed by default.

When an interface is implemented by a class, then it provides an implementation for all the abstract methods of the interface. Similarly, all the fields declared in the interface are also inherited by the class that implements the interface. Thus a copy of the interface field is present in the implementing class.

Now all the fields in the interface are by default static. Hence we can access them by using the interface name directly as the same as we access static fields of the class using the class name and not the object.

**The example Java program below shows how we can access the interface fields.**

|  |
| --- |
| //interface declaration  **interface** TestInterface{  **public** **static** **int** value = 100;       //interface field  **public** **void** display();  }  //Interface implementation  **class** TestClass **implements** TestInterface{  **public** **static** **int** value = 5000;      //class fields  **public** **void** display() {        System.out.println("TestClass::display () method");     }  **public** **void** show() {        System.out.println("TestClass::show () method");     }  }  **public** **class** Main{  **public** **static** **void** main(String args[]) {        TestClass testObj = **new** TestClass();        //print interface and class field values.        System.out.println("Value of the interface variable (value): "+TestInterface.value);        System.out.println("Value of the class variable (value): "+testObj.value);     }  } |

**Output:**

[Interface field example](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/Value-of-the-interface-variable.png)

As shown in the program above, the interface fields can be accessed using an Interface name followed by dot operator (.) and then the actual variable or field name.

**The Generic Interface In Java**

We have discussed Java generics in our earlier tutorials. Apart from having generic classes, methods, etc., we can also have generic interfaces. Generic interfaces can be specified similarly in the way in which we specify generic classes.

Generic interfaces are declared with type parameters which make them independent of a data type.

**The general syntax of the generic interface is as follows:**

interface <interface\_name><type-param-list>{

//interface methods and variables

}

**Now if we want to use the above generic interface in a class, then we can have the class definition as shown below:**

class <class\_name><type-param-list> implements interface\_name <type-param-list>{

//class body

}

Note that we have to specify the same param-list with the class as with the interface.

**The following Java program demonstrates the Generic Interfaces in Java.**

|  |
| --- |
| //generic interface declaration  **interface** MinInterface&lt;T **extends** Comparable&lt;T&gt;&gt;{       T minValue();  }    //implementation for generic interface  **class** MinClassImpl&lt;T **extends** Comparable&lt;T&gt;&gt; **implements** MinInterface&lt;T&gt; {       T[] intArray;       MinClassImpl(T[] o) {           intArray = o;       }    **public** T minValue() {           T v = intArray[0];    **for** (**int** i = 1; i &lt;intArray.length; i++) {  **if** (intArray[i].compareTo(v) &lt; 0) {                    v = intArray[i];               }           }  **return** v;       }  }    **public** **class** Main {  **public** **static** **void** main(String args[]) {           //create int and char type arrays           Integer intArray[] = { 13, 36, 22, 18, 26 };           Character charArray[] = { 'S', 's', 'V', 'w', 'p', 'R'};           //Create objects of type MinClassImpl with interger and character data types           MinClassImpl&lt;Integer&gt; intMinValue = **new** MinClassImpl&lt;Integer&gt;(intArray);           MinClassImpl&lt;Character&gt; charMinValue = **new** MinClassImpl&lt;Character&gt;(charArray);             //call interface method minValue for int type array           System.out.println("Min value in intOfArray: " + intMinValue.minValue());          //call interface method minValue for char type array           System.out.println("Min value in charOfArray: " + charMinValue.minValue());       } |

**Output:**

[Generic interface example output](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/Min-value-in-intOfArray.png)

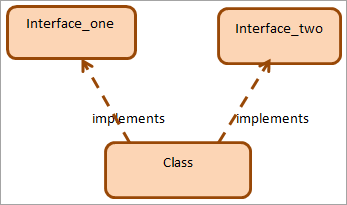
The above program implements an interface containing a method to find the minimum value in the array. This is a generic interface. The class implements this interface and overrides the method. In the main method, we call the interface method to find the minimum value in an integer and a character array.

**Multiple Interfaces In Java**

In our inheritance topic, we have seen that Java does not allow a class to inherit from multiple classes as it results in an ambiguity called the “Diamond Problem”.

However, a class can inherit or implement more than one interface. In this case, it is known as multiple inheritance. So although we are not allowed to implement multiple inheritance in Java through classes, we can do so using interfaces.

**The following diagram shows multiple inheritance using interfaces. Here a class implements two interfaces i.e. Interface\_one and Interface\_two.**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/class-implements-two-interfaces.png)

Note that when a class implements the multiple interfaces, the interface names are comma-separated in the class declaration. We can implement as many interfaces as long as we can handle the complexity.

**The Java program that demonstrates multiple interfaces is shown below.**

|  |
| --- |
| //Interface\_One declaration  **interface** Interface\_One{  **void** print();  }    //Interface\_Two declaration  **interface** Interface\_Two{  **void** show();  }    //multiple inheritance - DemoClass implementing Interface\_One&amp;Interface\_Two  **class** DemoClass **implements** Interface\_One,Interface\_Two{  **public** **void** print(){        //Override Interface\_One print()          System.out.println("Democlass::Interface\_One\_Print ()");      }  **public** **void** show(){         //Override Interface\_Two show()          System.out.println("DemoClass::Interface\_Two\_Show ()");      }  }    **public** **class** Main{    **public** **static** **void** main(String args[]){          DemoClass obj = **new** DemoClass();  //create DemoClass object and call methods          obj.print();          obj.show();      }  } |

**Output:**

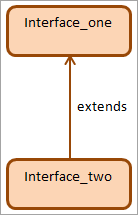
[multiple interfaces example](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/multiple-interfaces.png)

As shown above, we implement two interfaces. Then we override their respective methods and call them in the main method.

Multiple inheritance in Java provides all the benefits that multiple inheritance provide in the C++. But unlike multiple inheritance using classes, multiple inheritance using interfaces is without any ambiguity.

**Interface Inheritance In Java: Interface Extends Interface**

When a class implements an interface, it is done using the ‘**implements**’ keyword. In Java, an interface can inherit another interface. This is done using the ‘**extends**’ keyword. When an interface extends another interface it is called “**Interface inheritance**” in Java.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/interface-extends-interface.png)

**The Java program to implement interface inheritance is shown below.**

|  |
| --- |
| //Interface\_One declaration  **interface** Interface\_One{  **void** print();  }    //Interface\_Two declaration; inherits from Interface\_One  **interface** Interface\_Two **extends** Interface\_One{  **void** show();  }    //multiple inheritance - DemoClass implementing  Interface\_Two  **class** DemoClass **implements** Interface\_Two{  **public** **void** print(){        //Override Interface\_Two print()          System.out.println("Democlass  **public** **class** Main{  **public** **static** **void** main(String args[]){          DemoClass obj = **new** DemoClass();  //create DemoClass object and call methods          obj.print();          obj.show();      }  } |

**Output:**

[demo class example output](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/08/to-implement-interface-inheritance.png)

We have modified the same program that we used for multiple inheritance using interfaces to demonstrate the interface inheritance. Here, we extend Interface\_one in Interface\_two and then go about implementing Interface\_two in a class. As interfaces are inherited, both the methods are available for overriding.

**Frequently Asked Questions**

**Q #1) What is the use of the Interface in Java?**

**Answer:** An interface in Java is an entity that is used to achieve 100% abstraction. It can contain only abstract methods that can be overridden by the class implementing the interface.

The interface in a way acts like a blueprint of the class wherein it provides the class the abstract method prototypes and static constants and then the class has to override those methods by implementing the interface.

**Q #2) What are the advantages of the Interface in Java?**

**Answer:** **Some of the advantages of Interface are as follows:**

1. The interface acts as a blueprint of the class.
2. The interface provides 100% abstraction in Java as it has all the abstract methods.
3. Interfaces can be used to achieve multiple inheritance in Java. Java does not permit to inherit from more than one class but a class can implement multiple interfaces.

**#3) Can an interface have methods?**

**Answer:** Interfaces can have prototypes of methods and static and final constants. But starting from Java 8, interfaces can contain static and default methods.

**Q #4) Can we declare the interface as final?**

**Answer:** No. If we declare an interface as final, then the class will not be able to implement it. Without being implemented by any class, the interface will not serve any purpose.

**More About Interfaces**

Interfaces are blueprints like class, but it will have only the method declaration. It won’t have any method of implementation. All the methods in the interface are public abstract by default. Java 1.8 interface can have static and default methods.

Interfaces are mainly used in APIs.

**For Example:** Consider you are designing a vehicle’s engine.

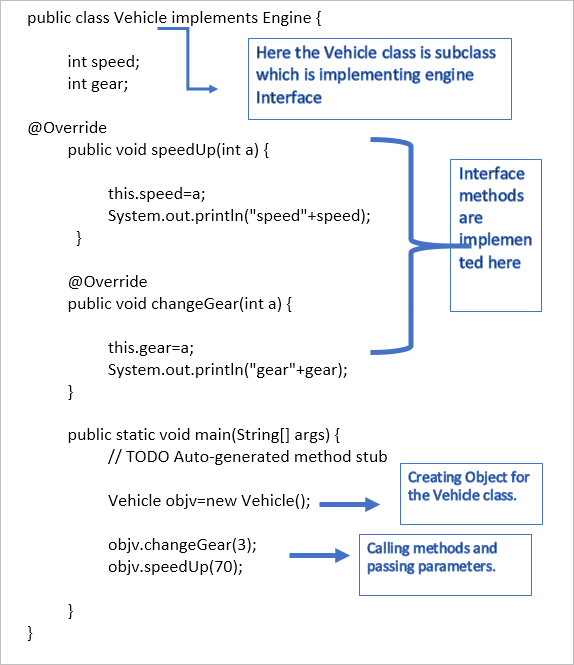
When you are done with the hardware part, you want some of the software functionalities to be implemented by a client who is using your engine. So, in that case, you can define your engine functionalities in an interface.

|  |
| --- |
| Interface Engine  {    **void** changeGear(**int** a);  **void** speedUp(**int** a);  } |

**Rules to be followed for Interface**

* The class which is implementing the interface should implement all the methods in the interface.
* An interface can contain final variables.

|  |
| --- |
| **public** **class** Vehicle **implements** Engine {    **int** speed;  **int** gear;    @Override  **public** **void** speedUp(**int** a) {    **this**.speed=a;    System.out.println("speed"+speed);  }    @Override  **public** **void** changeGear(**int** a) {    **this**.gear=a;  System.out.println("gear"+gear);  }    **public** **static** **void** main(String[] args) {  // TODO Auto-generated method    stub    Vehicle objv=**new** Vehicle();    objv.changeGear(3);  objv.speedUp(70);  }  } |



Here the Vehicle class is the subclass which is implementing the engine interface.

**What are Abstract Classes?**

An abstract class is like a class but it will have abstract methods and concrete methods. Abstract methods don’t have an implementation. It will only have the method declaration.

**Rules to be followed for Abstract Class**

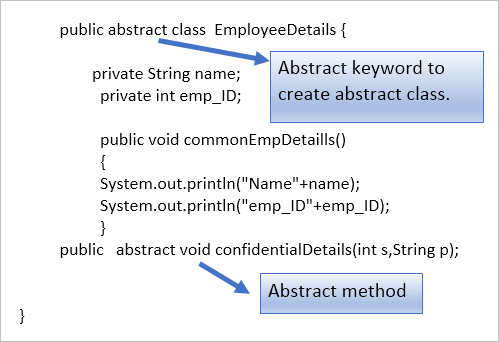
* The abstract class cannot be instantiated.
* Child class which extends the abstract class should implement all the abstract methods of the parent class or the Child class should be declared as an abstract class.

When you want to design partial implementation, you can go for an abstract class.

**Example abstract class program:**

**EmployeeDetails.java**

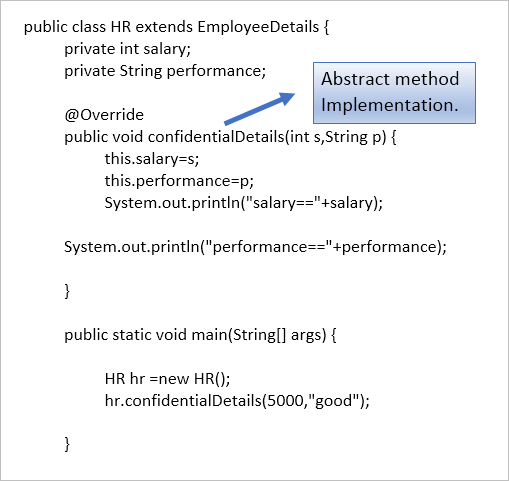
|  |
| --- |
| **public** **abstract** **class** EmployeeDetails {  **private** String name;  **private** **int** emp\_ID;    **public** **void** commonEmpDetaills()  {  System.out.println("Name"+name);  System.out.println("emp\_ID"+emp\_ID);  }  **public** **abstract** **void** confidentialDetails(**int** s,String p);    } |



The class which is going to extend the abstract class.

**HR.java**

|  |
| --- |
| **public** **class** HR **extends** EmployeeDetails {  **private** **int** salary;  **private** String performance;    @Override  **public** **void** confidentialDetails(**int** s,String p) {  **this**.salary=s;  **this**.performance=p;  System.out.println("salary=="+salary);  System.out.println("performance=="+performance);  }    **public** **static** **void** main(String[] args) {  HR hr =**new** HR();  hr.confidentialDetails(5000,"good");  }    } |



**Key points to be noted:**

* In Interfaces, all the methods will not have method implementation.
* The class which is implementing the interface should implement all the methods in that particular interface.
* Abstract classes can have abstract methods as well as normal concrete methods. Abstract methods don’t have an implementation.
* The class which is extending the abstract class should have the implementation for all the abstract methods in the abstract class.
* If the subclass doesn’t have enough information to implement the abstract methods, then the subclass should be declared as an abstract class.

**Conclusion**

In this tutorial, we have presented the basic concepts of interfaces in Java. We have discussed the definition of the interface, along with the need for interfaces. We explored their basic syntax and definition. Then we discussed how to use interfaces for which we use the ‘implements’ keyword.

We learned how to use multiple interfaces and interface inheritance in Java. Using multiple interfaces we can implement multiple inheritance in Java. Interface inheritance is when one interface extends another interface.