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. 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.
```

URL class in Java with Examples
Reading from a URL using URLConnection Class
Networking in Java | Set 1 (Java.net.InetAddress class)
Arrays in Java
Java | How to start learning Java
HashMap in Java
How JVM Works - JVM Architecture?
Stack Class in Java
Arrays.sort() in Java with examples
ArrayList in Java
Scanner Class in Java
Multithreading in Java
Collections in Java
Queue Interface In Java
Ways to read input from console in Java
LinkedList in Java
Inheritance in Java
PriorityQueue in Java
Beginning Java programming with Hello World Example
enum in Java
Classes and Objects in Java
Reverse a string in Java
How to get rid of Java TLE problem
HashSet in Java
Split() String method in Java with examples
String class in Java | Set 1
Binary Tree (Array implementation)
Java Naming Conventions
Collections.sort() in Java with Examples

Most popular in Java

Differences between JDK, JRE and JVM
Data types in Java
Exceptions in Java
Internal Working of HashMap in Java
Abstraction in Java

Most visited in School Programming

Arrays in C/C++
Print a given matrix in spiral form
Program to find GCD or HCF of two numbers
Basic Concepts of Object Oriented Programming using C++
C++ Data Types

```
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) {
```


Most visited in Java

- Java Identifiers
- Encapsulation in Java
- Formatted output in Java
- Java Virtual Machine (JVM) Stack Area
- Commonly Asked Java Programming Interview Questions | Set 2
- Association, Composition and Aggregation in Java
- Abstract Classes in Java
- String vs StringBuilder vs StringBuffer in Java
- Checked vs Unchecked Exceptions in Java
- Serialization and Deserialization in Java with Example
- Fast I/O in Java in Competitive Programming
- How to iterate any Map in Java
- BigInteger Class in Java
- How are Java objects stored in memory?
- Difference between Scanner and BufferReader Class in Java

```
    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
```

2. 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
- Access specifiers for classes or interfaces in Java
- Abstract Classes in Java
- Comparator Interface in Java
- Java Interface methods
- 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
Interfaces and Inheritance in Java
Functional Interfaces In Java
Callback using Interfaces in Java
Private Methods in Java 9 Interfaces
Access specifiers for classes or interfaces in Java
Access specifier of methods in interfaces
Two interfaces with same methods having same signature but different return types
Java.util.LinkedList.poll(), pollFirst(), pollLast() with examples in Java
Java lang.Long.reverse() method in Java with Examples
Java lang.Long.highestOneBit() method in Java with Examples
Java lang.Long.byteValue() method in Java with Examples
Java lang.Long.lowestOneBit() method in Java with Examples
Java lang.Long.numberOfLeadingZeros() method in Java with Examples
Java lang.Long.numberOfTrailingZeros() method in Java with Examples

Improved By : RohitKumar8, RAraghavarora

Article Tags : Java School Programming java-interfaces

Practice Tags : Java

2

1.9

To-do Done

Based on **34** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous
PriorityQueue in Java

Next
Networking in Java | Set 1 (Java.net.InetAddress class)

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Share this post!

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

Abstract class in Java

← prev | next →

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

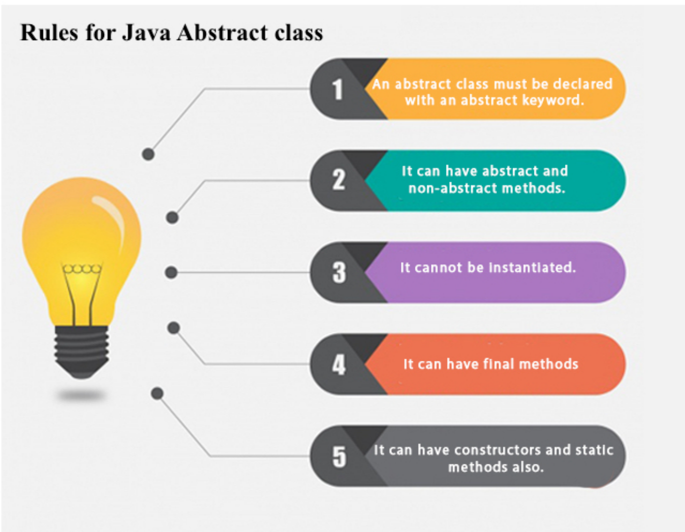
1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.



Example of abstract class

```
abstract class A{}
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

```
abstract void printStatus();//no method body and abstract
```

Example of Abstract class that has an abstract method

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Test it Now

```
running safely
```

Understanding the real scenario of Abstract class

In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.

Mostly, we don't know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the **factory method**.

A **factory method** is a method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

File: TestAbstraction1.java

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
s.draw();
}
}
```

Test it Now

```
drawing circle
```

Another example of Abstract class in java

File: TestBank.java

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
```

```
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Test it Now

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

Abstract class having constructor, data member and methods

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

File: TestAbstraction2.java

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Test it Now

```
bike is created
running safely..
gear changed
```

Rule: If there is an abstract method in a class, that class must be abstract.

```
class Bike12{
abstract void run();
}
```

Test it Now

```
compile time error
```

Rule: If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

Another real scenario of abstract class

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

Note: If you are beginner to java, learn interface first and skip this example.

```
interface A{
void a();
void b();
void c();
void d();
}

abstract class B implements A{
public void c(){System.out.println("I am c");}
}

class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
```

```
public void d(){System.out.println("I am d");}
}

class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Test it Now

```
Output:I am a
       I am b
       I am c
       I am d
```

Next Topic Interface in Java

← prev next →

Learn Java
Learn Data Structures
Learn C Programming
Learn C++ Tutorial
Learn C# Tutorial
Learn PHP Tutorial
Learn HTML Tutorial
Learn JavaScript Tutorial
Learn jQuery Tutorial
Learn Spring Tutorial

javatpoint.com
Hindi100.com
Lyricsia.com
Quoteperson.com
Hindi-typing.com
Shayaree.com
Jobandplacement.com

Website Development
Android Development
Website Designing
Digital Marketing
Summer Training
Industrial Training
College Campus Training

Address: G-13, 2nd Floor, Sec-3
Noida, UP, 201301, India
Contact No: 0120-4256464, 9990449935

Contact Us
Subscribe Us
Privacy Policy
Sitemap

Java Training
- Basics of Java
- Java Object Class
  - Java OOPs Concepts
  - Naming Convention
  - Object and Class
  - Constructor
  - static keyword
  - this keyword
- Java Inheritance
  - Inheritance(IS-A)
  - Aggregation(HAS-A)
- Java Polymorphism
  - Method Overloading
  - Method Overriding
  - Covariant Return Type
  - super keyword
  - Instance Initializer block
  - final keyword
  - Runtime Polymorphism
  - Dynamic Binding
  - instanceof operator
- Java Abstraction
  - Abstract class
  - Interface
  - Abstract vs Interface
- Java Encapsulation
  - Package
  - Access Modifiers
  - Encapsulation
- Java Array
  - Java Array
- Java OOPs Misc
  - Object class
  - Object Cloning
  - Math class
  - Wrapper Class
  - Java Recursion
  - Call By Value
  - strictfp keyword
  - javadoc tool
  - Command Line Arg
  - Object vs Class
  - Overloading vs Overriding
- Java String
- Java Regex
- Exception Handling
- Java Inner classes
- Java Multithreading
- Java I/O
- Java Networking
- Java AWT & Events
- Java Swing
- JavaFX
- Java Applet
- Java Reflection
- Java Date
- Java Conversion
- Java Collection
- Java JDBC
- Java New Features
- RMI
- Internationalization
- Interview Questions

Difference between abstract class and interface

← prev | next →

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance.**	Interface **supports multiple inheritance.**
3) Abstract class **can have final, non-final, static and non-static variables.**	Interface has **only static and final variables.**
4) Abstract class **can provide the implementation of interface.**	Interface **can't provide the implementation of abstract class.**
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class**can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class**can be extended using keyword "extends".	An **interface class**can be implemented using keyword "implements".
8) A Java**abstract class**can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Test it Now

Output:

```
I am a
I am b
I am c
I am d
```

Next Topic Package in Java

← prev next →

Please Share

Learn Latest Tutorials

Programs Selenium Control System Rust Intellij DBMS