

[Home](#)[Java](#)[Programs](#)[OOPs](#)[String](#)[Exception](#)[Multithreading](#)[Collections](#)[↑ SCROLL TO TOP](#)

## Abstract class in Java

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

↑ SCROLL TO TOP



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.

### Rules for Java Abstract class



### 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

↑ SCROLL TO TOP



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();           >>>Here we can hold child's obj. into parent's refrence..
        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
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;
}
}

```



↑ SCROLL TO TOP

```
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 abs

↑ SCROLL TO TOP

```
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();
    }
}
```



↑ SCROLL TO TOP

```
a.b();  
a.c();  
a.d();  
}}
```

**Test it Now**

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

[← Prev](#)[Next →](#)

For Videos Join Our Youtube Channel: [Join Now](#)

**Feedback**

- Send your Feedback to [feedback@javatpoint.com](mailto:feedback@javatpoint.com)

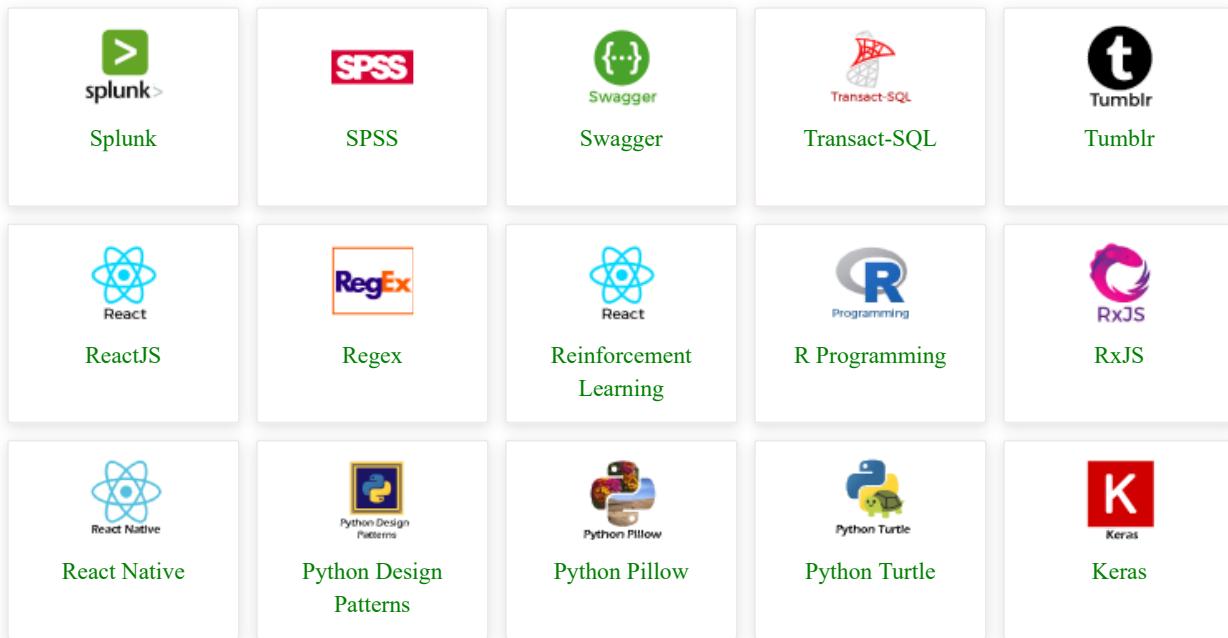
**Help Others, Please Share**[↑ SCROLL TO TOP](#)

## Identity Management Capability

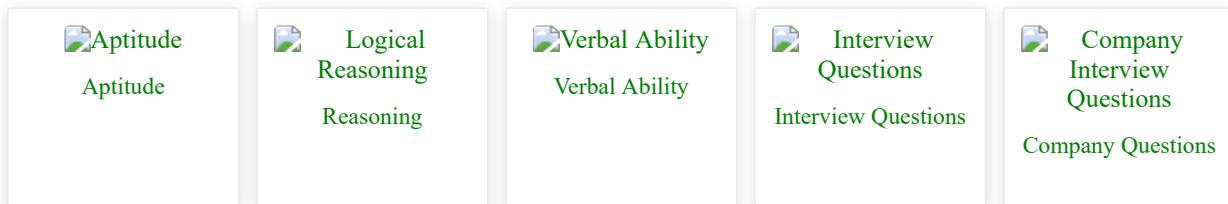
Embed Powerful Analytics inside Your Applications

Bold BI by Syncfusion

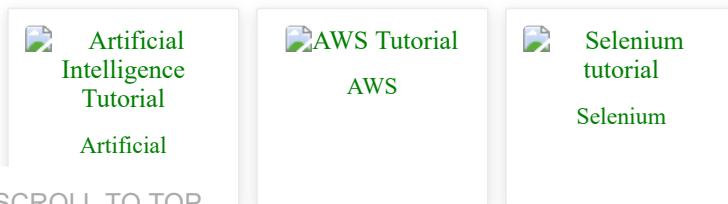
### Learn Latest Tutorials



### Preparation



### Trending Technologies



SCROLL TO TOP





ReactJS Tutorial

ReactJS



Data Science Tutorial

Data Science



Angular 7 Tutorial

Angular 7

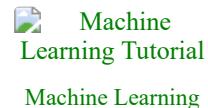


Blockchain Tutorial

Blockchain



Git



Machine Learning Tutorial

Machine Learning



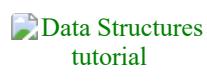
DevOps Tutorial

DevOps

## B.Tech / MCA



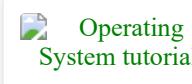
DBMS



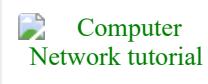
Data Structures



DAA



Operating System



Computer Network

Computer Network

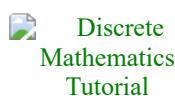


Compiler Design



Computer Organization and Architecture

Computer Organization



Discrete Mathematics

Discrete Mathematics



Ethical Hacking

Ethical Hacking



Computer Graphics

Computer Graphics



Software Engineering

Engineering



Web Technology



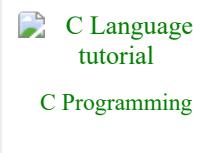
Cyber Security

Cyber Security



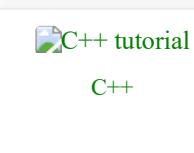
Automata

Automata



C Language

Programming



C++



Java



.Net Framework

.Net



Database Management Systems

Database Management Systems

X



Control



Data Mining

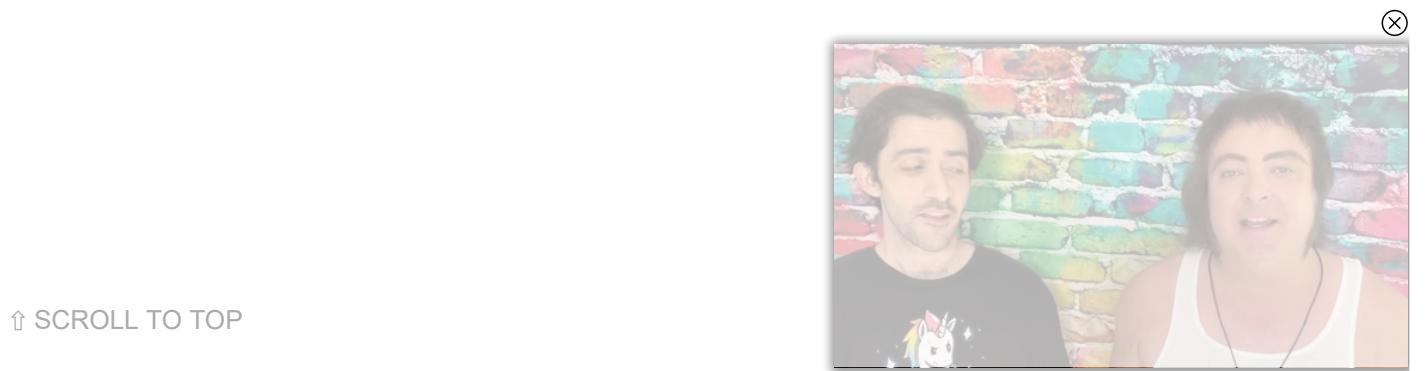
Data Mining



Data Warehouse

Tutorial

↑ SCROLL TO TOP



↑ SCROLL TO TOP