Abstract class in Java

abstract is a keyword

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

Example of abstract method:

abstract public void fun();

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](https://www.javatpoint.com/object-and-class-in-java) 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](https://www.javatpoint.com/java-constructor) 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**

1. **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**

1. **abstract** **void** printStatus();//no method body in 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.

1. **abstract** **class** Bike{
2. **abstract** **void** run();
3. }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println("running safely");}
6. **public** **static** **void** main(String args[]){
7. Bike obj = **new** Honda4();
8. obj.run();
9. }
10. }

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*

1. **abstract** **class** Shape{
2. **abstract** **void** draw();
3. }
4. //In real scenario, implementation is provided by others i.e. unknown by end user
5. **class** Rectangle **extends** Shape{
6. **void** draw(){System.out.println("drawing rectangle");}
7. }
8. **class** Circle1 **extends** Shape{
9. **void** draw(){System.out.println("drawing circle");}
10. }
11. //In real scenario, method is called by programmer or user
12. **class** TestAbstraction1{
13. **public** **static** **void** main(String args[]){
14. Shape s=**new** Circle1();
15. s.draw();
16. }
17. }

drawing circle

### **Another example of Abstract class in java**

*Home assignment*

*Create method simpleInterest*

*File: TestBank.java*

1. **abstract** **class** Bank{
2. **abstract** **int** getRateOfInterest();
3. }
4. **class** SBI **extends** Bank{
5. **int** getRateOfInterest(){**return** 7;}
6. }
7. **class** PNB **extends** Bank{
8. **int** getRateOfInterest(){**return** 8;}
9. }
11. **class** TestBank{
12. **public** **static** **void** main(String args[]){
13. Bank b;
14. b=**new** SBI();
15. System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
16. b=**new** PNB();
17. System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
18. }}

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.

Class Parent

{

public void five5000()

}

Class Child extends Parent

{

Public void five5000()

}

Class Test

{

p.s.v.m(String a[])

{

Parent c1=new Child();

C1.5000(); // child will always use its own 5000()

}

}

*File: TestAbstraction2.java*

1. //Example of an abstract class that has abstract and non-abstract methods
2. **abstract** **class** Bike{
3. Bike(){System.out.println("bike is created");}
4. **abstract** **void** run();
5. **void** changeGear(){System.out.println("gear changed");}
6. }
7. //Creating a Child class which inherits Abstract class
8. **class** Honda **extends** Bike{
9. **void** run(){System.out.println("running safely..");}
10. }
11. //Creating a Test class which calls abstract and non-abstract methods
12. **class** TestAbstraction2{
13. **public** **static** **void** main(String args[]){
14. Bike obj = **new** Honda();
15. obj.run();
16. obj.changeGear();
17. }
18. }

bike is created

running safely..

gear changed

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

1. **class** Bike12{
2. **abstract** **void** run();
3. }

compile time error

### **Another real scenario of abstract class**

The abstract class can also be used to provide some implementation of the [interface](https://www.javatpoint.com/interface-in-java). 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.**

1. **interface** A{
2. **void** a();
3. **void** b();
4. **void** c();
5. **void** d();
6. }
8. **abstract** **class** B **implements** A{
9. **public** **void** c(){System.out.println("I am c");}
10. }
12. **class** M **extends** B{
13. **public** **void** a(){System.out.println("I am a");}
14. **public** **void** b(){System.out.println("I am b");}
15. **public** **void** d(){System.out.println("I am d");}
16. }
18. **class** Test5{
19. **public** **static** **void** main(String args[]){
20. A a=**new** M();
21. a.a();
22. a.b();
23. a.c();
24. a.d();
25. }}

Output:I am a

I am b

I am c

I am d

/\*

\* if a class has more than 1 abstract method child class has to override all the abstract methods

\* or if you don't want to override then make child class also an abstract class

\*/

**package** Lecture16;

**abstract** **class** C

{

**abstract** **void** m1();

**abstract** **void** m2();

}

**class** D **extends** C

{

**public** **void** m1()

{

System.***out***.println("M!");

}

**public** **void** m2()

{

System.***out***.println("M2");

}

}

**abstract** **class** E **extends** C

{

**public** **void** m1()

{

System.***out***.println("Class E m1");

}

**abstract** **void** m3();

}

**public** **class** TestAbstractionWithmorethanOneAbstractMethod {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

}

}