Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Before learning 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 important things to the user and hides the internal details for example sending sms, you just 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 Abstaction

There are two ways to achieve abstraction in java

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

### Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

### Example abstract class

1. **abstract** **class** A{}

### abstract method

|  |
| --- |
| A method that is declared as abstract and **does not have implementation** is known as abstract method. |

### Example abstract method

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

### Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It 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, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the factory method.

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();//In real scenario, object is provided through method e.g. getShape() method
15. s.draw();
16. }
17. }

drawing circle

### Another example of abstract class in java

*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, methods etc.

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

*File: TestAbstraction2.java*

1. //example of abstract class that have method body
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. }
8. **class** Honda **extends** Bike{
9. **void** run(){System.out.println("running safely..");}
10. }
11. **class** TestAbstraction2{
12. **public** **static** **void** main(String args[]){
13. Bike obj = **new** Honda();
14. obj.run();
15. obj.changeGear();
16. }
17. }

bike is created

running safely..

gear changed

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

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

compile time error

#### Rule: If you are extending any abstract class that have 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.

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