**Abstract class in Java**

A class that is declared with abstract keyword, is known as abstract class. Before learning abstract class, let's understand the abstraction first.

**Abstraction**

**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**

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.**

**Syntax to declare the abstract class**

1. abstract class <class\_name>{}

**abstract method**

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

**Syntax to define the abstract method**

1. abstract return\_type <method\_name>();//no braces{}

**Example of abstract class that have 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.
5. class Honda extends Bike{
6. void run(){System.out.println("running safely..");}
7.
8. public static void main(String args[]){
9. Bike obj = new Honda();
10. obj.run();
11. }
12. }

Output: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**.

A **factory method** is the 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.

1. abstract class Shape{
2. abstract void draw();
3. }
4.
5. class Rectangle extends Shape{
6. void draw(){System.out.println("drawing rectangle");}
7. }
8.
9. class Circle extends Shape{
10. void draw(){System.out.println("drawing circle");}
11. }
12.
13. class Test{
14. public static void main(String args[]){
15. Shape s=new Circle();
16. //In real scenario, Object is provided through factory method
17. s.draw();
18. }
19. }

Output:drawing circle

**Abstract class having constructor, data member, methods etc.**

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

1. //example of abstract class that have method body
2. abstract class Bike{
3. abstract void run();
4. void changeGear(){System.out.println("gear changed");}
5. }
6.
7. class Honda extends Bike{
8. void run(){System.out.println("running safely..");}
9.
10. public static void main(String args[]){
11. Bike obj = new Honda();
12. obj.run();
13. obj.changeGear();
14. }
15. }

Output:running safely..

 gear changed

1. //example of abstract class having constructor, field and method
2. abstract class Bike
3. {
4. int limit=30;
5. Bike(){System.out.println("constructor is invoked");}
6. void getDetails(){System.out.println("it has two wheels");}
7. abstract void run();
8. }
9.
10. class Honda extends Bike{
11. void run(){System.out.println("running safely..");}
12.
13. public static void main(String args[]){
14. Bike obj = new Honda();
15. obj.run();
16. obj.getDetails();
17. System.out.println(obj.limit);
18. }
19. }

Output:constructor is invoked

running safely..

it has two wheels

30

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

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

Output:compile time error

**Rule: If you are extending any abstact class that have abstract method, you must either provide the implementation of the method or make this class abstract.**

Abstraction refers to the ability to make a class abstract in OOP. An abstract class is one that cannot be instantiated. All other functionality of the class still exists, and its fields, methods, and constructors are all accessed in the same manner. You just cannot create an instance of the abstract class.

If a class is abstract and cannot be instantiated, the class does not have much use unless it is subclass. This is typically how abstract classes come about during the design phase. A parent class contains the common functionality of a collection of child classes, but the parent class itself is too abstract to be used on its own.

## Abstract Class:

Use the **abstract** keyword to declare a class abstract. The keyword appears in the class declaration somewhere before the class keyword.

/\* File name : Employee.java \*/

public abstract class Employee

{

 private String name;

 private String address;

 private int number;

 public Employee(String name, String address, int number)

 {

 System.out.println("Constructing an Employee");

 this.name = name;

 this.address = address;

 this.number = number;

 }

 public double computePay()

 {

 System.out.println("Inside Employee computePay");

 return 0.0;

 }

 public void mailCheck()

 {

 System.out.println("Mailing a check to " + this.name

 + " " + this.address);

 }

 public String toString()

 {

 return name + " " + address + " " + number;

 }

 public String getName()

 {

 return name;

 }

 public String getAddress()

 {

 return address;

 }

 public void setAddress(String newAddress)

 {

 address = newAddress;

 }

 public int getNumber()

 {

 return number;

 }

}

Notice that nothing is different in this Employee class. The class is now abstract, but it still has three fields, seven methods, and one constructor.

Now if you would try as follows:

/\* File name : AbstractDemo.java \*/

public class AbstractDemo

{

 public static void main(String [] args)

 {

 /\* Following is not allowed and would raise error \*/

 Employee e = new Employee("George W.", "Houston, TX", 43);

 System.out.println("\n Call mailCheck using Employee reference--");

 e.mailCheck();

 }

}

When you would compile above class then you would get the following error:

Employee.java:46: Employee is abstract; cannot be instantiated

 Employee e = new Employee("George W.", "Houston, TX", 43);

 ^

1 error

## Extending Abstract Class:

We can extend Employee class in normal way as follows:

/\* File name : Salary.java \*/

public class Salary extends Employee

{

 private double salary; //Annual salary

 public Salary(String name, String address, int number, double

 salary)

 {

 super(name, address, number);

 setSalary(salary);

 }

 public void mailCheck()

 {

 System.out.println("Within mailCheck of Salary class ");

 System.out.println("Mailing check to " + getName()

 + " with salary " + salary);

 }

 public double getSalary()

 {

 return salary;

 }

 public void setSalary(double newSalary)

 {

 if(newSalary >= 0.0)

 {

 salary = newSalary;

 }

 }

 public double computePay()

 {

 System.out.println("Computing salary pay for " + getName());

 return salary/52;

 }

}

Here, we cannot instantiate a new Employee, but if we instantiate a new Salary object, the Salary object will inherit the three fields and seven methods from Employee.

/\* File name : AbstractDemo.java \*/

public class AbstractDemo

{

 public static void main(String [] args)

 {

 Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

 Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

 System.out.println("Call mailCheck using Salary reference --");

 s.mailCheck();

 System.out.println("\n Call mailCheck using Employee reference--");

 e.mailCheck();

 }

}

This would produce the following result:

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

Mailing check to John Adams with salary 2400.

## Abstract Methods:

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.

The abstract keyword is also used to declare a method as abstract. An abstract method consists of a method signature, but no method body.

Abstract method would have no definition, and its signature is followed by a semicolon, not curly braces as follows:

public abstract class Employee

{

 private String name;

 private String address;

 private int number;

 public abstract double computePay();

 //Remainder of class definition

}

Declaring a method as abstract has two results:

* The class must also be declared abstract. If a class contains an abstract method, the class must be abstract as well.
* Any child class must either override the abstract method or declare itself abstract.

A child class that inherits an abstract method must override it. If they do not, they must be abstract and any of their children must override it.

Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

If Salary is extending Employee class, then it is required to implement computePay() method as follows:

/\* File name : Salary.java \*/

public class Salary extends Employee

{

 private double salary; // Annual salary

 public double computePay()

 {

 System.out.println("Computing salary pay for " + getName());

 return salary/52;

 }

 //Remainder of class definition

}

**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.**

1. interface A{
2. void a();
3. void b();
4. void c();
5. void d();
6. }
7.
8. abstract class B implements A{
9. public void c(){System.out.println("I am C");}
10. }
11.
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. }
17.
18. class Test{
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

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.
5. **class** Honda4 **extends** Bike{
6. **void** run(){System.out.println("running safely..");}
7.
8. **public** **static** **void** main(String args[]){
9. Bike obj = **new** Honda4();
10. obj.run();
11. }
12. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Honda4)

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**.

A **factory method** is the 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.
9. **class** Circle1 **extends** Shape{
10. **void** draw(){System.out.println("drawing circle");}
11. }
12.
13. //In real scenario, method is called by programmer or user
14. **class** TestAbstraction1{
15. **public** **static** **void** main(String args[]){
16. Shape s=**new** Circle1();//In real scenario, object is provided through method e.g. getShape() method
17. s.draw();
18. }
19. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction1)

drawing circle

Another example of abstract class in java

*File: TestBank.java*

1. **abstract** **class** Bank{
2. **abstract** **int** getRateOfInterest();
3. }
4.
5. **class** SBI **extends** Bank{
6. **int** getRateOfInterest(){**return** 7;}
7. }
8. **class** PNB **extends** Bank{
9. **int** getRateOfInterest(){**return** 7;}
10. }
11.
12. **class** TestBank{
13. **public** **static** **void** main(String args[]){
14. Bank b=**new** SBI();//if object is PNB, method of PNB will be invoked
15. **int** interest=b.getRateOfInterest();
16. System.out.println("Rate of Interest is: "+interest+" %");
17. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestBank)

Rate of Interest is: 7 %

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. }
7.
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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction2)

 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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Bike12)

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.

***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. }
7.
8. **abstract** **class** B **implements** A{
9. **public** **void** c(){System.out.println("I am C");}
10. }
11.
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. }
17.
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. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Test5)

Output:I am a

 I am b

 I am c

 I am d

**Next Topic**[Interface in Java](http://www.javatpoint.com/interface-in-java)