

Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either **abstract classes** or interfaces.

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal {  
  
    public abstract void animalSound();  
  
    public void sleep() {  
  
        System.out.println("Zzz");  
  
    }  
}
```

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();
}
}
output:
running safely
```

some 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.
- the abstract methods of an abstract class must be defined in its subclass.
- we cannot declare abstract constructors or abstract static methods.

INTERFACES IN JAVA

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

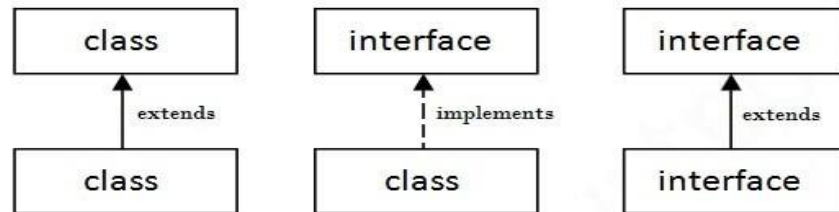
```
interface <interface_name>{
```

```
// declare constant fields
```

```
// declare methods that abstract
```

```
// by default.
```

```
}
```



A class that implement interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

example:

```
interface printable{
```

```
void print();
```

```
}
```

```
class A6 implements printable{
```

```
public void print()
{
System.out.println("Hello");
}
```

```
public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

Output:Hello

some points to remember:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final
- An interface cannot contain a constructor (as it cannot be used to create objects)

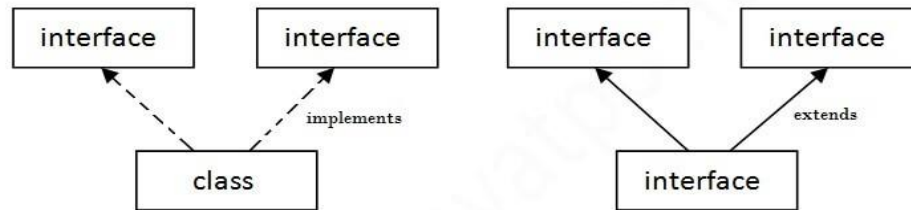
Why And When To Use Interfaces?

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. To implement multiple interfaces, separate them with a comma (see example below).

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

example of multiple inheritance:

```
interface Printable{  
    void print();  
}  
interface Showable{  
    void show();  
}  
class A7 implements Printable,Showable{  
    public void print(){System.out.println("Hello");}  
    public void show(){System.out.println("Welcome");}  
  
    public static void main(String args[]){  
        A7 obj = new A7();
```

```
obj.print();
obj.show();
}
}
Output:Hello
       Welcome
```

Interface inheritance

A class implements an interface, but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
}
}
```

Output:

Hello
Welcome

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 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();  
}
```