

# **Object Oriented Programming with Java**

## **(Subject Code: BCS-403)**

### **Unit 1**

### **Lecture 10**

# Lecture 10

- **Abstraction**
- **Interfaces**
- **Abstract Class**

# Abstraction in Java

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

## Example to understand Abstraction:

Television remote control is an excellent example of abstraction. It simplifies the interaction with a TV by hiding the complexity behind simple buttons and symbols, making it easy without needing to understand the technical details of how the TV functions.

## 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).

## Ways to achieve Abstaction

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- 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

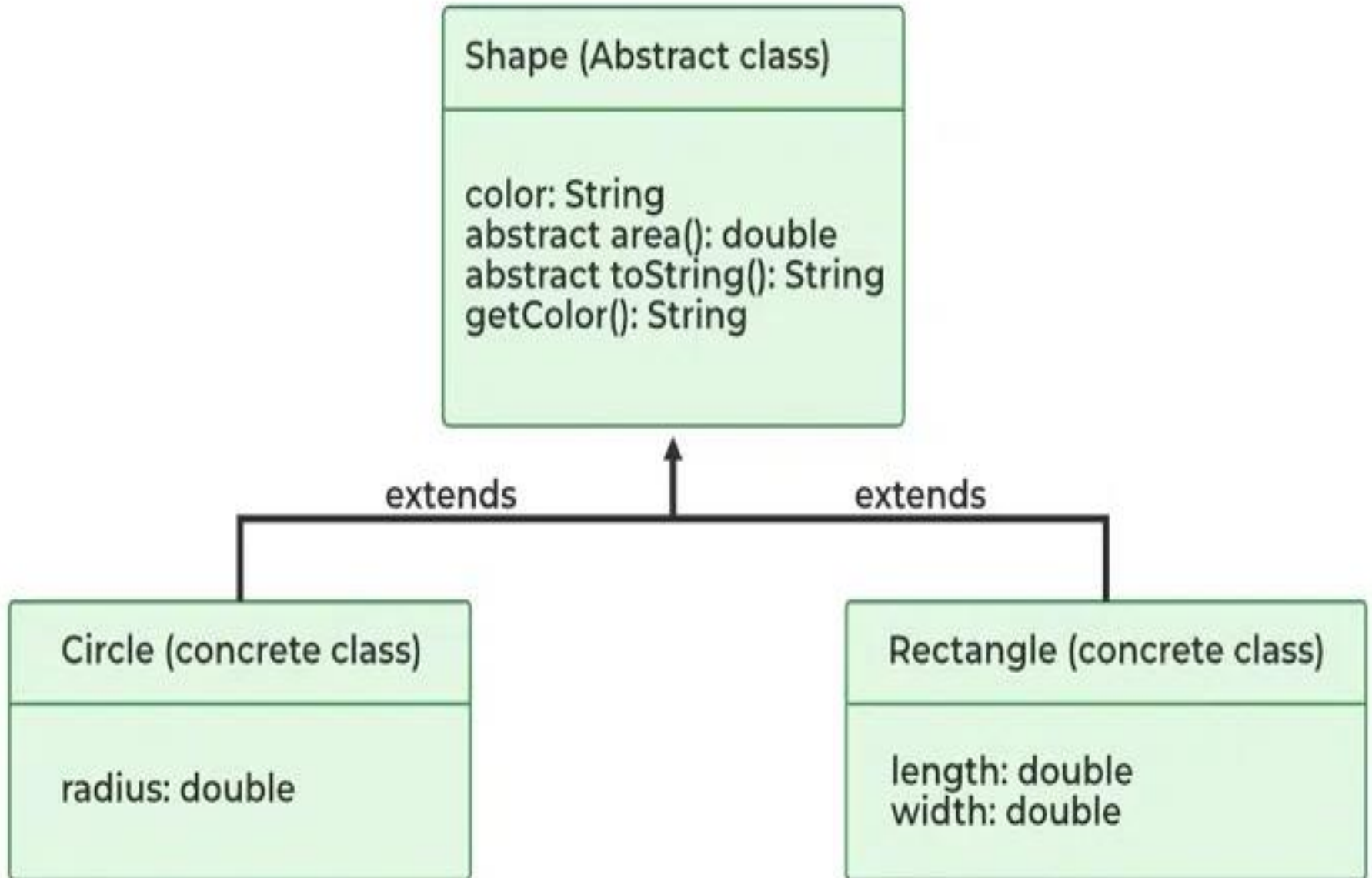
```
abstract class A{}
```

### Abstract method

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

### Example

```
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. 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[]){  
    Honda4 obj = new Honda4();  
    obj.run();  
}  
}
```

```
abstract class Bank{  
abstract double getRateOfInterest();  
}
```

```
class SBI extends Bank{  
double getRateOfInterest(){return 7.5;}  
}
```

```
class PNB extends Bank{  
double getRateOfInterest(){return 7;}  
}
```

```
class TestBank{  
public static void main(String args[]){  
SBI b=new SBI();//if object is PNB, method of PNB will be invoked  
int interest=b.getRateOfInterest();  
System.out.println("Rate of Interest is: "+interest+" %");  
}}
```

- Rate of Interest is: 7.5 %



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

*File: TestAbstraction2.java*

//example of abstract class that have method body

```
abstract class Bike{  
    Bike(){System.out.println("bike is created");}  
    abstract void run();  
    void changeGear(){System.out.println("gear changed");}  
}  
class Honda extends Bike{  
    void run(){System.out.println("running safely..");}  
}  
class TestAbstraction2{  
    public static void main(String args[]){  
        Bike obj = new Honda();  
        obj.run();  
        obj.changeGear();  
    } }  
}
```

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

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.

```
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[]){  
        M a=new M();  
        a.a();  
        a.b();  
        a.c();  
        a.d();  
    }  
}
```

# Interface in Java

- An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.
- The interface in java is a **mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.

# Why use Java interface?

There are mainly three reasons to use interface.  
They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

**The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.**

In other words, Interface fields are public, static and final by default, and methods are public and abstract.

```
interface Printable{  
    int MIN=5;  
    void print();  
}
```

Printable.java

compiler

```
interface Printable{  
    public static final int MIN=5;  
    public abstract void print();  
}
```

Printable.class

```
interface printable{  
void print();  
}
```

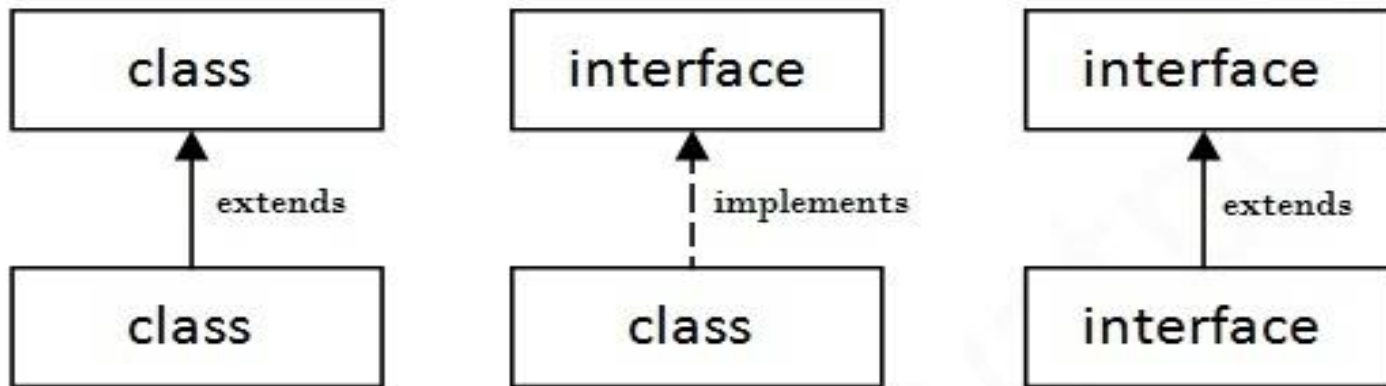
```
class A6 implements printable{  
public void print(){System.out.println("Hello");}
```

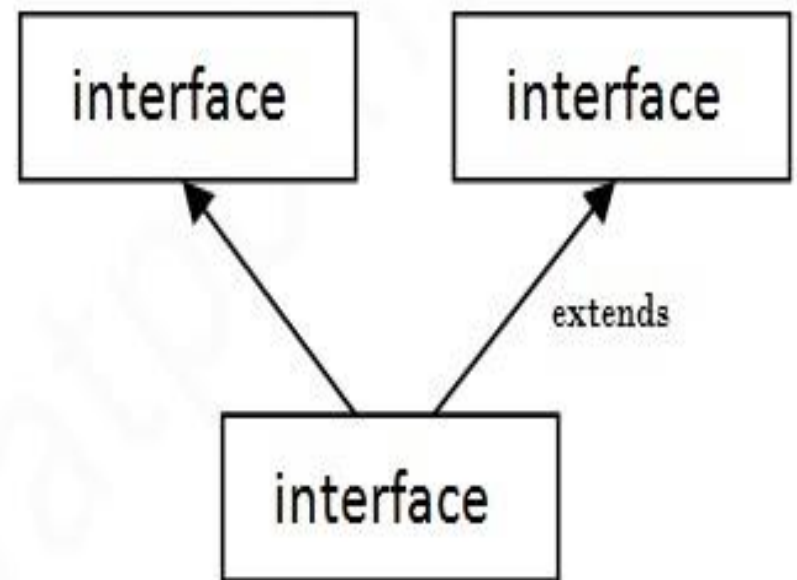
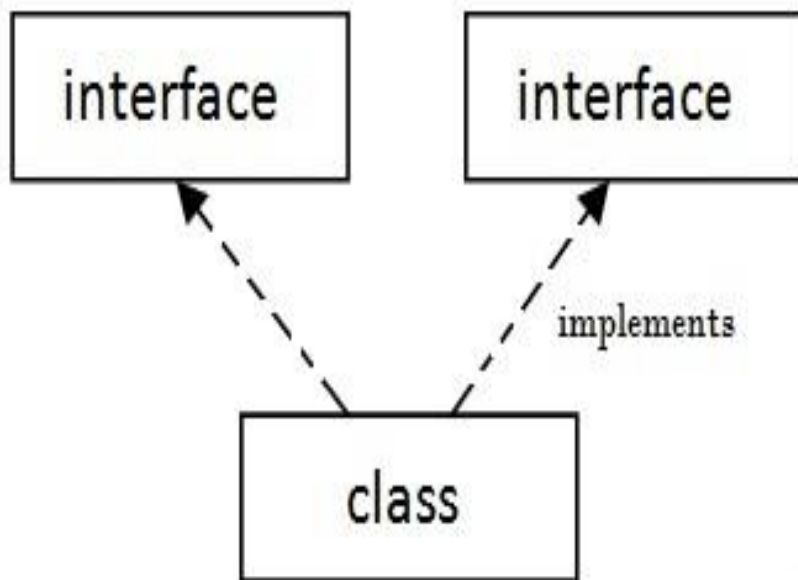
```
public static void main(String args[]){  
    A6 obj = new A6();  
    obj.print();  
}  
}
```



## Multiple inheritance in Java by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.





## Multiple Inheritance in Java

```
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();
```

```
}
```

```
}
```

## Interface inheritance

A class implements interface but one interface extends another interface .

```
interface Printable{
```

```
void print();
```

```
}
```

```
interface Showable extends Printable{
```

```
void show();
```

```
}
```

```
class Testinterface2 implements Showable{
```

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

```
public void show(){System.out.println("Welcome");}
```

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

```
Testinterface2 obj = new Testinterface2();
```

```
obj.print();
```

```
obj.show();
```

```
} }
```

Abstract class	Interface
1) Abstract class can <b>have abstract and non-abstract</b> methods.	Interface can have <b>only abstract</b> methods.
2) Abstract class <b>doesn't support multiple inheritance</b> .	Interface <b>supports multiple inheritance</b> .
3) Abstract class <b>can have final, non-final, static and non-static variables</b> .	Interface has <b>only static and final variables</b> .
4) Abstract class <b>can have static methods, main method and constructor</b> .	Interface <b>can't have static methods, main method or constructor</b> .
5) Abstract class <b>can provide the implementation of interface</b> .	Interface <b>can't provide the implementation of abstract class</b> .
6) The <b>abstract</b> keyword is used to declare abstract class.	The <b>interface</b> keyword is used to declare interface.
<b>) Example:</b> <pre>public abstract class Shape{ public abstract void draw(); }</pre>	<b>Example:</b> <pre>public interface Drawable{ void draw(); }</pre>