Khulna Khan Bahadur Ahsanullah University

Object-oriented programming

CSE 1203

Lecture -5

Abstraction

- Hiding internal details and **showing functionality** is known as abstraction.
- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- A class which is declared with the abstract keyword is known as an abstract class
- we use abstract class and interface to achieve abstraction.
- example: **phone call**, we do not know the internal processing.

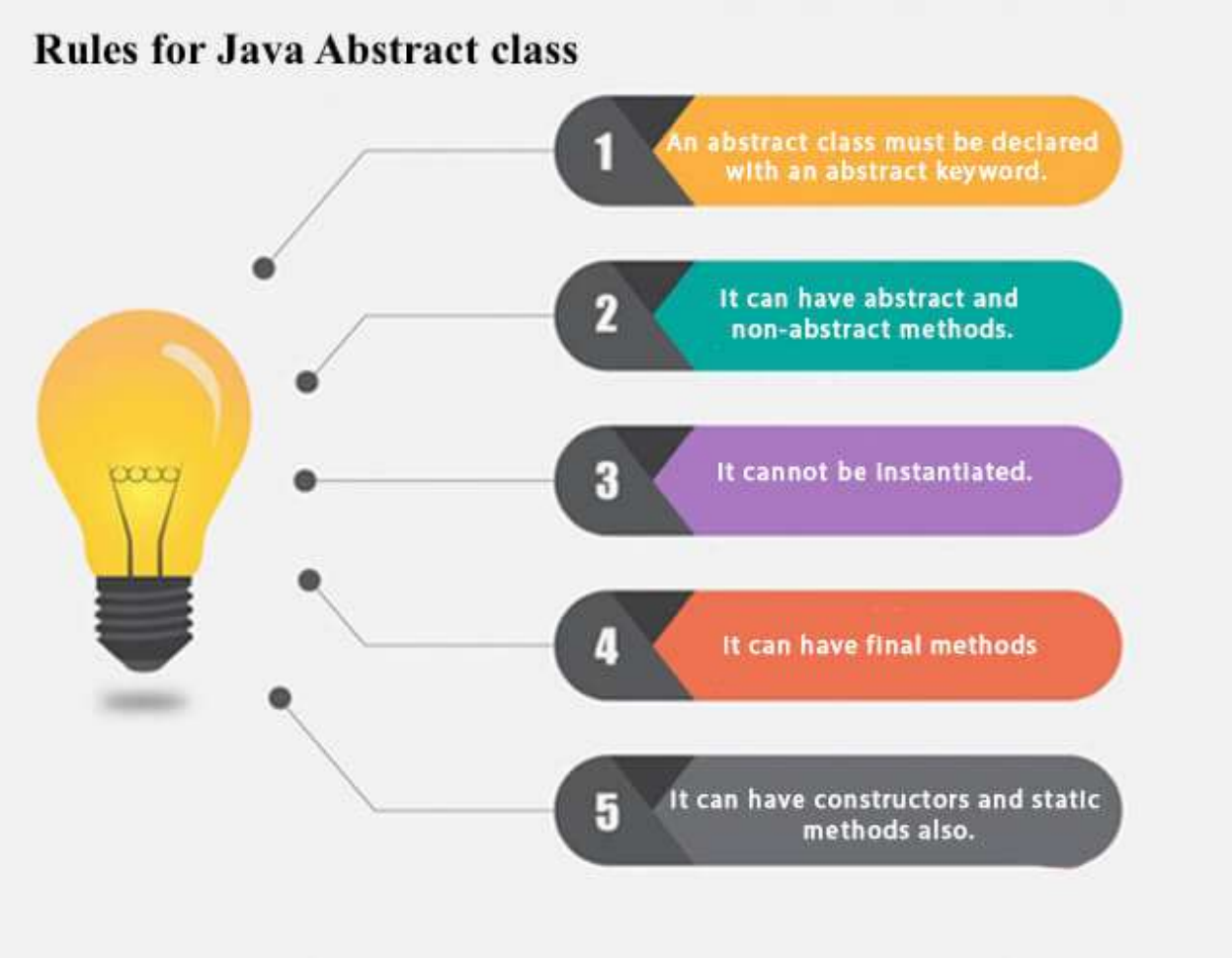
Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

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.
- It can have final methods which will force the subclass not to change the body of the method.



Example of abstract class

```
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

```
abstract void printStatus();//no method body and 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. class Rectangle extends Shape{
4. void draw(){System.out.println("drawing rectangle");}
5. }
6. class Circle1 extends Shape{
7. void draw(){System.out.println("drawing circle");}  }
8. class TestAbstraction1{
9. public static void main(String args[]){
10.Shape s=new Circle1();//In a real scenario, object is provided through metho
    d, e.g., getShape() method
11.s.draw();
12.}  }
```

```
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. }
10.
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 %
```

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

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

Rule: If you are extending an abstract class that has an 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. }
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.}}
```

```
Output:I am a
       I am b
       I am c
       I am d
```

Interface in Java

- An interface is a collection of abstract methods
- An **interface in Java** is a blueprint of a class.
- 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.
- Java Interface also **represents the IS-A relationship**.
- It cannot be instantiated just like the abstract class. Since Java 8, we can have **default and static methods** in an interface. Since Java 9, we can have **private methods** in an interface.

Why use Java interface?

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

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

How to declare an interface?

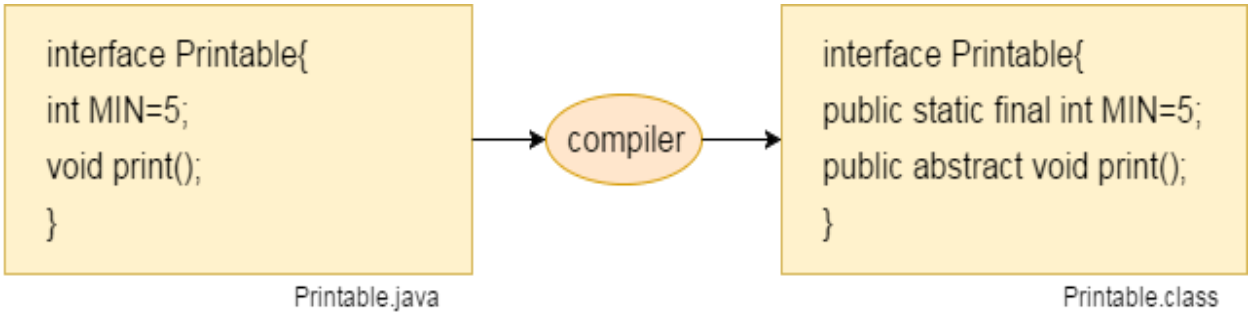
- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared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

Syntax:

```
1. interface <interface_name>{
2.
3.     // declare constant fields
4.     // declare methods that abstract
5.     // by default.
6. }
```

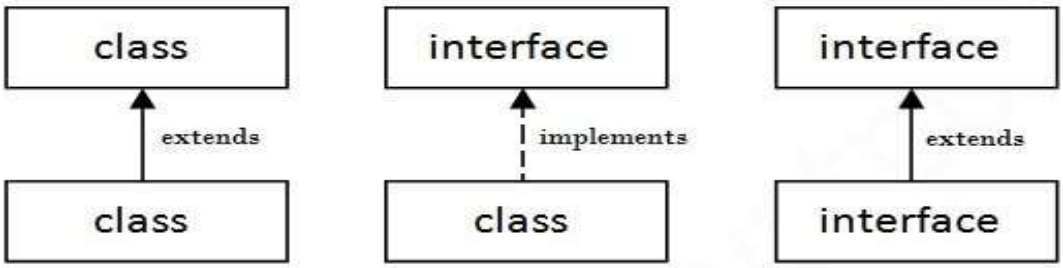
Internal addition by the compiler

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



The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
1. interface printable{
2. void print();  }
3. class A6 implements printable{
4. public void print(){System.out.println("Hello");}
5. public static void main(String args[]){
6. A6 obj = new A6();
7. obj.print();
8.  }  }
```

Output:

```
Hello
```

Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

File: TestInterface1.java

```
1. //Interface declaration: by first user
2. interface Drawable{
3. void draw();  }
4. class Rectangle implements Drawable{
5. public void draw(){System.out.println("drawing rectangle");}
6. }
7. class Circle implements Drawable{
8. public void draw(){System.out.println("drawing circle");}
9. }
10.class TestInterface1{
11.public static void main(String args[]){
12.Drawable d=new Circle();//In real scenario, object is provided by method e.
   g. getDrawable()
13.d.draw();
14.}}
```

Output:

```
drawing circle
```

Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

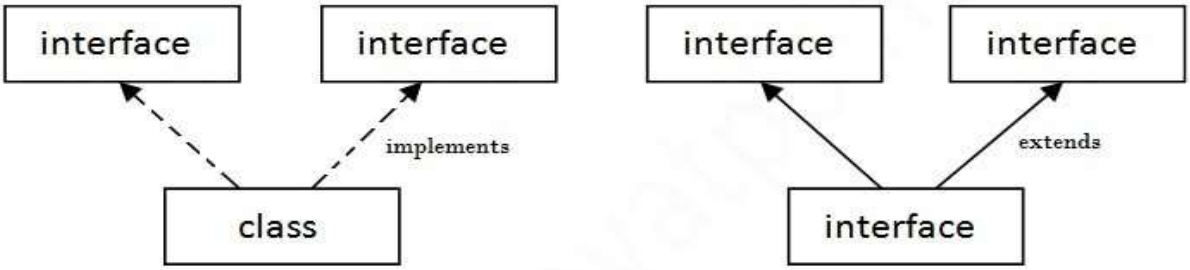
```
1. interface Bank{
2. float rateOfInterest();
3. }
4. class SBI implements Bank{
5. public float rateOfInterest(){return 9.15f;}
6. }
7. class PNB implements Bank{
8. public float rateOfInterest(){return 9.7f;}
9. }
10.class TestInterface2{
11.public static void main(String[] args){
12.Bank b=new SBI();
13.System.out.println("ROI: "+b.rateOfInterest());
14.}}
```

Output:

```
ROI: 9.15
```

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

```
1. interface Printable{
2. void print();  }
3. interface Showable{
4. void show();  }
5. class A7 implements Printable,Showable{
6. public void print(){System.out.println("Hello");}
7. public void show(){System.out.println("Welcome");}
8.
9. public static void main(String args[]){
10.A7 obj = new A7();
11.obj.print();
12.obj.show();
13. }  }
```

```
Output: Hello
        Welcome
```

Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class. For example:

```
1. interface Printable{
2. void print();  }
3. interface Showable{
4. void print();  }
5. class TestInterface3 implements Printable, Showable{
6. public void print(){System.out.println("Hello");}
7. public static void main(String args[]){
8. TestInterface3 obj = new TestInterface3();
9. obj.print();
10. }  }
```

Output:

```
Hello
```

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

Interface inheritance

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

```
1. interface Printable{
2. void print();
3. }
4. interface Showable extends Printable{
5. void show();
6. }
7. class TestInterface4 implements Showable{
8. public void print(){System.out.println("Hello");}
9. public void show(){System.out.println("Welcome");}
10.
11.public static void main(String args[]){
12.TestInterface4 obj = new TestInterface4();
13.obj.print();
14.obj.show();
15. }
16.}
```

Output:

```
Hello
Welcome
```

Q) What is marker or tagged interface?

An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
1. //How Serializable interface is written?
2. public interface Serializable{
3. }
```

Nested Interface in Java

Note: An interface can have another interface which is known as a nested interface. For example:

```
1. interface printable{
2.  void print();
3.  interface MessagePrintable{
4.    void msg();
5.  }
6. }
```

Difference between abstract class and interface

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods.
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();}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

```
1. //Creating interface that has 4 methods
2. interface A{
3. void a();//by default, public and abstract
4. void b();
5. void c();
6. void d();
7. }
8. //Creating abstract class that provides the implementation of one method of
   A interface
9. abstract class B implements A{
10.public void c(){System.out.println("I am C");}
11.}
12.//Creating subclass of abstract class, now we need to provide the implementa
   tion of rest of the methods
13.class M extends B{
14.public void a(){System.out.println("I am a");}
15.public void b(){System.out.println("I am b");}
16.public void d(){System.out.println("I am d");}
17.}
18.//Creating a test class that calls the methods of A interface
19.class Test5{
20.public static void main(String args[]){
21.A a=new M();
22.a.a();
```

```
23.a.b();
24.a.c();
25.a.d();
26.}}
```

Output:

```
I am a
I am b
I am c
I am d
```