R-Creation

track of IT

215/318, Hasina Monjil (1st Floor), Muradpur, Panchlaish, Chittagong.
Website: www.rcreation-bd.com, Contact: +8801722-964303

Lecture – 11 (Abstraction)

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 with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
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

```
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.

```
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();
}
}
```

Abstract class having constructor, member data, methods etc.

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();
 }
}
```

Interface 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.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

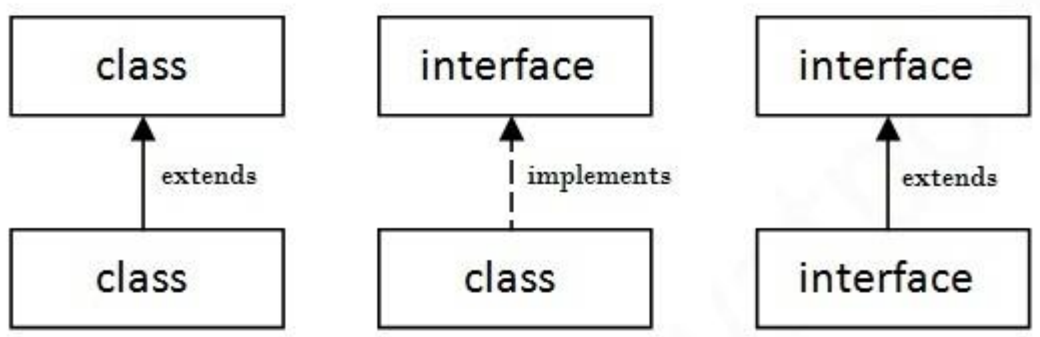
Why use Java interface?

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

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

Understanding 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, Printable interface has only one method, its implementation is provided in the A 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();
 }
}
```

Java Interface Example: Bank

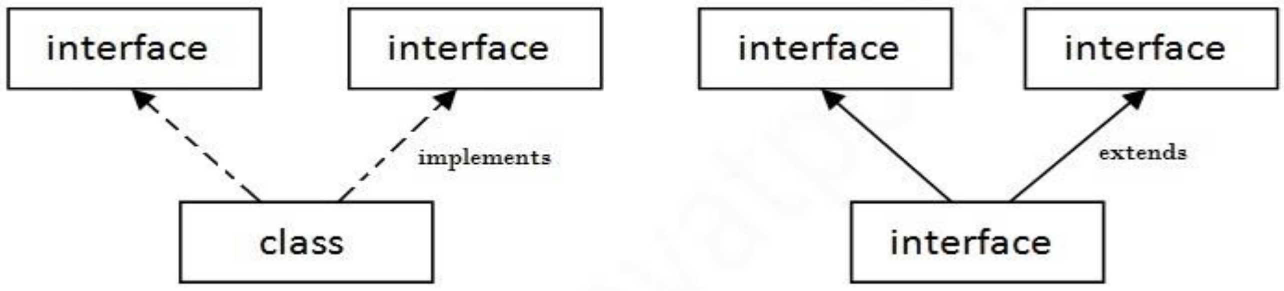
Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

```
interface Bank{
float rateOfInterest();
}
class DBBL  implements Bank{
public float rateOfInterest(){
return 9.15f;
}
}
class BrackBank implements Bank{
public float rateOfInterest(){
return 9.7f;
}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new DBBL ();
System.out.println("Interest Rate: "+b.rateOfInterest());
}
}
```

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();
 }
}
```

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

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) **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.

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}
```

```
//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){
System.out.println("I am C");
}
}
//Creating subclass of abstract class, now we need to provide the implementation of rest of
the methods
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");
}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}
}
```