Java 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 essential things to the user and hides the internal details,
- Example: sending SMS where you 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 Abstraction
  - **Abstract class (0 to 100%)**
  - **Interface (100%)**

Abstract Classes

Abstract methods

- You can *declare* an object without *defining* it:

```
Person p;
```

- Similarly, you can declare a *method* without defining it:

```
public abstract void draw(int size);
```

  - Notice that the body of the method is missing

- A method that has been declared but not defined is an **abstract method**

Example:

```
abstract class A {
abstract void callme();
// concrete methods are still allowed in
abstract classes
void callmetoo() {
System.out.println("This is a concrete
method.");
}
}
class B extends A {
void callme() {
System.out.println("B's implementation of
callme.");
}
}
}
```

```
class AbstractDemo {
public static void main(String args[])
{
B b = new B();
b.callme();
b.callmetoo();
}
```

O/P:

```
B's implementation of callme.
This is a concrete method.
```

Abstract classes

- Any class containing an abstract method is an **abstract class**
- You must declare the class with the keyword abstract:

  abstract class MyClass {…}

- You cannot **instantiate** (create a new instance of) an abstract class
- It can have abstract and non-abstract methods.
- It can have constructors and static methods also.

Abstract classes (contd..)

- You can extend (subclass) an abstract class
  - If the subclass defines all the inherited abstract methods, it is "complete"and **can be instantiated**
  - If the subclass does *not* define all the inherited abstract methods, it too must be abstract
- You can declare a class to be abstract even if it does not contain any abstract methods
  - This prevents the class from being instantiated

Example: 1

```
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
```

Example:2

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by user
class Rectangle extends Shape
{
void draw(){
System.out.println("drawing rectangle");
}
}
class Circle1 extends Shape
{
void draw(){
System.out.println("drawing circle");
}
}
```

```
//In real scenario, method is called by programmer or user
class TestAbstraction1
{
public static void main(String args[])
{
Shape s=new Circle1();
s.draw();
}
}
```

Output:

```
drawing circle
```

Example:3 abstract class with constructor

```
abstract class Base
{
 Base()
 {
System.out.println("Base Constructor
Called"); }
 abstract void fun();
}
class Derived extends Base
 {
  Derived()
 {
 System.out.println("Derived Constructor
Called"); }
void fun() { System.out.println("Derived
fun() called"); }
}

class Main
 {
   public static void main(String args[])
 {
     Derived d = new Derived();
   }
}
```

Output:

```
Base Constructor Called
Derived Constructor Called
```

Example: 4 An abstract class without any abstract method

```
abstract class Base
{
 void fun()
 { System.out.println("Base fun() called"); }
}

class Derived extends Base { }

class Main
 {
   public static void main(String args[]) {
   Derived d = new Derived();
   d.fun();
   }
}
```

Output:

```
Base fun() called
```

Example 5: An abstract class with a final method

```
abstract class Base
 {
final void fun()
 { System.out.println("Derived fun() called"); }
}

class Derived extends Base {}

class Main
 {
 public static void main(String args[])
 {
   Base b = new Derived();
    b.fun();
    }
}
```

Output:

```
Derived fun() called
```

Interfaces

What is an Interface?

- An interface is similar to an abstract class with the following exceptions:
  - All methods defined in an interface are abstract. Interfaces contain no implementation
  - Interfaces cannot contain instance variables.

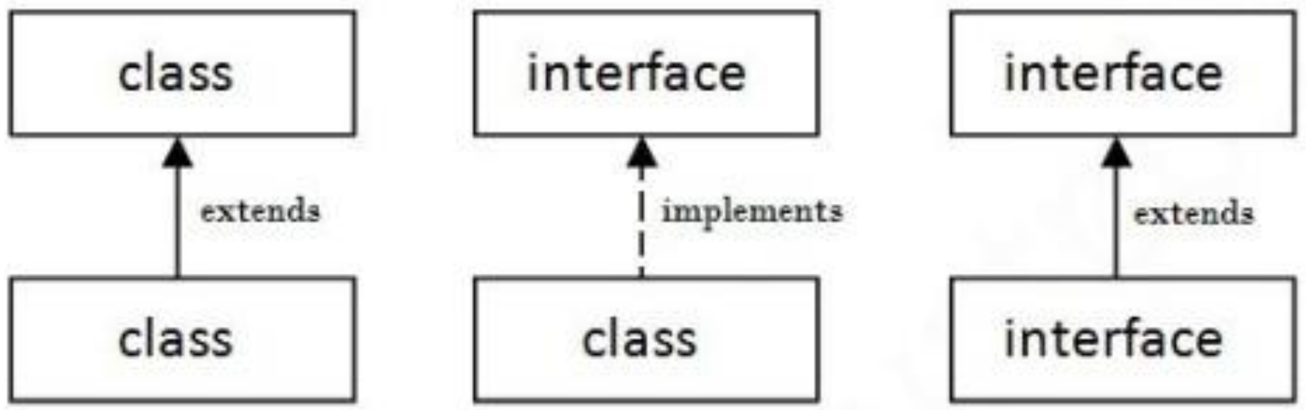
    However, they can contain public, static or final variables (ie. constant class variables)

- Interfaces are declared using the "interface" keyword

- Interfaces are more abstract than abstract classes

- Interfaces are implemented by classes using the "implements" keyword.

Relationship between classes and interfaces



Declaring an Interface

In Steerable.java:

```
public interface Steerable
{
    public void turnLeft(int degrees);
    public void turnRight(int degrees);
}
```

In Car.java:

```
public class Car extends Vehicle implements Steerable
{
    public int turnLeft(int degrees)
    {
            [...]
    }

    public int turnRight(int degrees)
    {
            [...]
    }
```

When a class "implements" an interface, the compiler ensures that it provides an implementation for all methods defined within the interface.

Why do we use interface ?

- It is used to achieve total abstraction.
- Since java **does not support multiple inheritance** in case of class, but by using interface it can achieve multiple inheritance
- So the question arises why use interfaces when we have abstract classes?

abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

Implementing Interfaces

- A Class can only inherit from one superclass. However, a class may implement several Interfaces
  - The interfaces that a class implements are separated by commas

- Any class which implements an interface must provide an implementation for all methods defined within the interface.

NOTE: if an **abstract class** implements an interface, it NEED NOT implement all methods defined in the interface. HOWEVER, each concrete subclass MUST implement the methods defined in the interface.

Example:1

```
import java.io.*;
interface in1
{
    // public, static and final
    final int a = 10;
   // public and abstract
    void display();
}

class testClass implements in1
{
    // Implementing the capabilities of
   // interface.
  public void display()
   {
      System.out.println(“abc");
   }


// Driver Code
   public static void main (String[] args)
   {
      testClass t = new testClass();
      t.display();
      System.out.println(a);
   }
}
```

Output:

```
abc
10
```

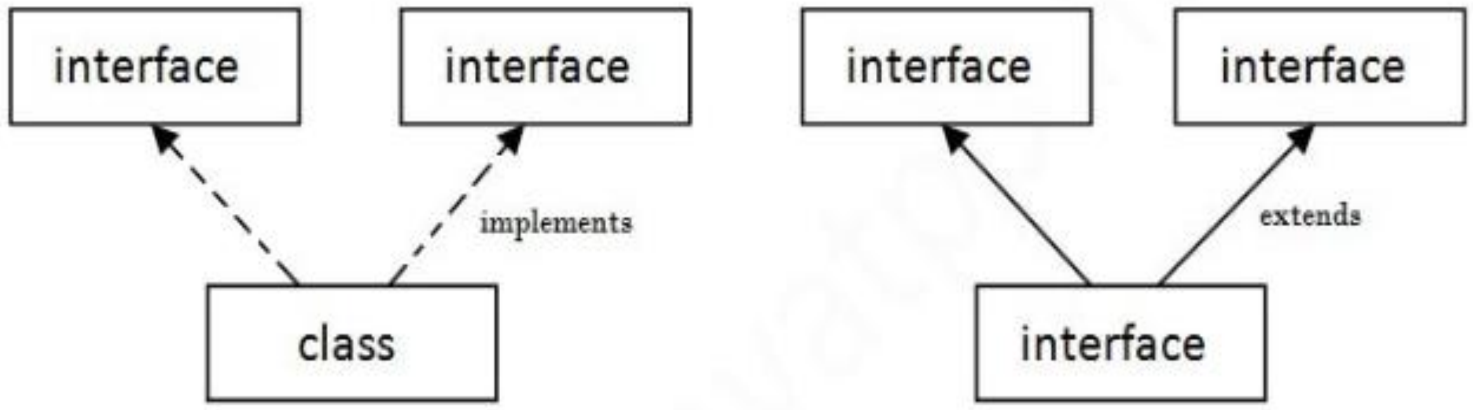
Example:2

```
interface MyInterface
{ /* compiler will treat them as: public abstract
methods */
 public void method1();
 public void method2();
}
class Demo implements
MyInterface
{/* This class must have to implement both
the abstract methods else you will get
compilation error*/
public void method1()
{
System.out.println("implementation of
method1");
}
public void method2()
{
System.out.println("implementation of
method2");
}

public static void main(String arg[])
  {
           MyInterface obj = new Demo();
           obj.method1();
  }
}
```

O/P: implementation of method1

Multiple inheritance in Java by interface



Multiple Inheritance in Java

Example:

```
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
```

Extending an Interface: Interface Inheritance

- One interface can inherit another by use of the keyword **extends**.
- The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.

Example: Interface Inheritance

```
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
```

Difference between

	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();
		}

Assignment: Abstract Class

- We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B.
- Create an abstract class 'Marks' with an abstract method 'getPercentage'.
- It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students.
- The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B.
- Create an object for eac of the two classes and print the percentage of marks for both the students.

Assignment: Interface

- You are given an interface *Arithmetic* which contains a method signature *int sum(int n), int minus(int n), int multiply(int x,int y),int division(int x, int y*.
- You need to write a class called MyCalculator which implements the interface.