Abstract Class

Abstract Class

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- Shows only important things to the user and hides the internal details
- Abstraction lets you focus on what the object does instead of how it does it.

…Contd

- A class that is declared with abstract keyword, is known as abstract class in java.
- It can have abstract and non-abstract methods

Ways to achieve Abstaction

- There are two ways to achieve abstraction in java
  – Abstract class (0 to 100%)
  – Interface (100%)

Abstract Class Syntax

```
abstract class ClassName
{
        ...
        ...
        abstract Type MethodName1();
        ...
        ...
        Type Method2()
        {
            // method body
        }
}
```

- When a class contains one or more abstract methods, it should be declared as abstract class.
- The abstract methods of an abstract class must be defined in its subclass.
- We cannot declare abstract constructors or abstract static methods.

...contd

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

  - **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

Abstract Class - Example

```
abstract class Bank
{
    abstract int getRateOfInterest();
}
class SBI extends Bank
{
    int getRateOfInterest(){return 7;}
}
class PNB extends Bank
{
    int getRateOfInterest(){return 9;}
}
class TestBank
{
    public static void main(String args[]){
    Bank 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+" %");
}
}
```

The Shape Abstract Class

```
public abstract class Shape {
        public abstract double area();
        public void move() { // non-abstract method
            // implementation
        }
}
```

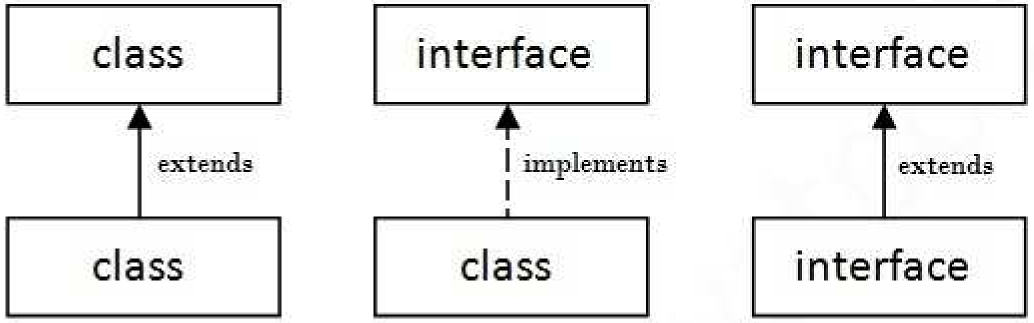
- Is the following statement valid?
  - Shape s = new Shape();
- No. It is illegal because the Shape class is an abstract class, which cannot be instantiated to create its objects.

Abstract Classes Properties

- A class with one or more abstract methods is automatically abstract and it cannot be instantiated.
- A class declared abstract, even with no abstract methods can not be instantiated.
- A subclass of an abstract class can be instantiated if it overrides all abstract methods by implementation them.
- A subclass that does not implement all of the superclass abstract methods is itself abstract; and it cannot be instantiated.

...Contd

- Writing an interface is similar to writing a class.
- A class describes the attributes and behaviours of an object.
- An interface contains behaviours that a class implements.



What is interface?

- An interface is similar to class
- It is a collection of abstract methods.
- A class implements an interface, inherits the a
- Along with abstract methods an interface may also contain constants, default methods and static methods.
- Method bodies exist only for default methods and static methods.

How it is different from class?

- All of the methods in an interface are abstract.
- You cannot instantiate an interface.
- An interface does not contain any constructors.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both **static and final**.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

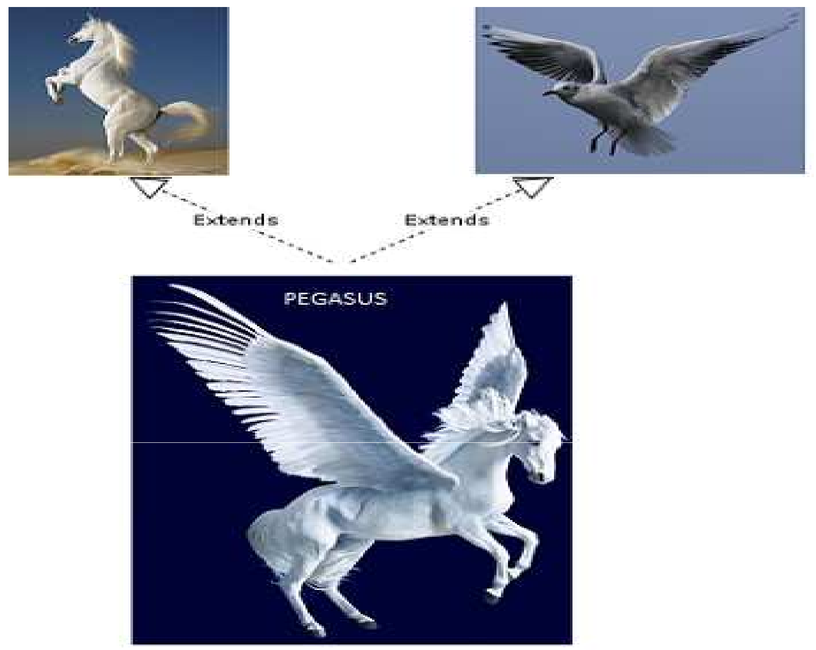
Why use Java interface?

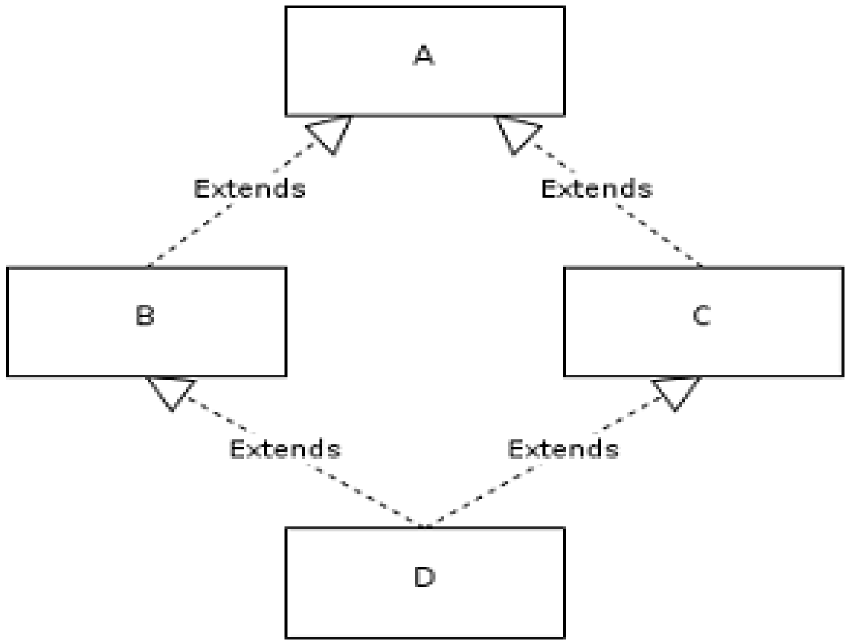
- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Simple 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();
 }
}
```

Why not Multiple Inheritance?





We have two classes B and C inheriting from A.
Assume that B and C are overriding an inherited method and they provide their own implementation.
Now D inherits from both B and C.
D should inherit that overridden method, which overridden method will be used?
Will it be from B or C? Here we have an ambiguity.

Multiple Inheritance – Example1

```
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();
 }
}
```

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

Class implementing multiple interfaces

Multiple Inheritance – Example2

```
interface Printable
{
    void print();
}
interface Showable
{
    void print();
}
class TestTnterface1 implements Printable,Showable
{
public void print(){System.out.println("Hello");}
public static void main(String args[])
{
    TestTnterface1 obj = new TestTnterface1();
    obj.print();
 }
}
```

Printable and Showable interface have same methods, but its implementation is provided by class TestTnterface1, so there is no ambiguity.