Basic Java

Unit 4 – Abstract Classes and Interfaces

Pratian Technologies (India) Pvt. Ltd.
www.pratian.com

PRATIAN
TECHNOLOGIES

PRATIAN TECHNOLOGIES

Topics

- Abstract classes & Abstract methods
- Abstract methods as a contract
- Interfaces
- Interface as a contract
- Programming to the interface vs. programming to the implementation
- Interface and Multiple Inheritance
- Inheritance among interfaces
- Polymorphism revisited

PRATIAN TECHNOLOGIES

Abstract Classes

- An abstract class is a class that exists only as a notion, and has no real world mapping.
- An abstract class cannot be instantiated.
- An abstract type always exists as one or more sub types.
- An abstract class serves as a placeholder for common structure and behavior, which all it's sub classes can reuse.

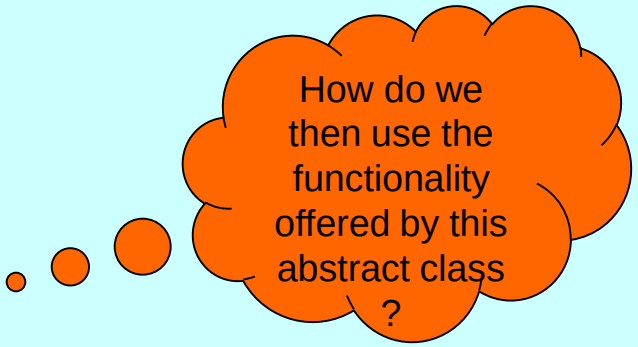
PRATIAN TECHNOLOGIES

Abstract classes in Java

- An abstract class can be defined by use of the keyword 'abstract' in the class definition

```
abstract class Vehicle
{
    Engine e;
    FuelTank tank;
    void pullFuelFromTank()
        {.....}

     void regulateEngineTemp()
         {.....}
     void start() { }
     void stop() { }
}
```

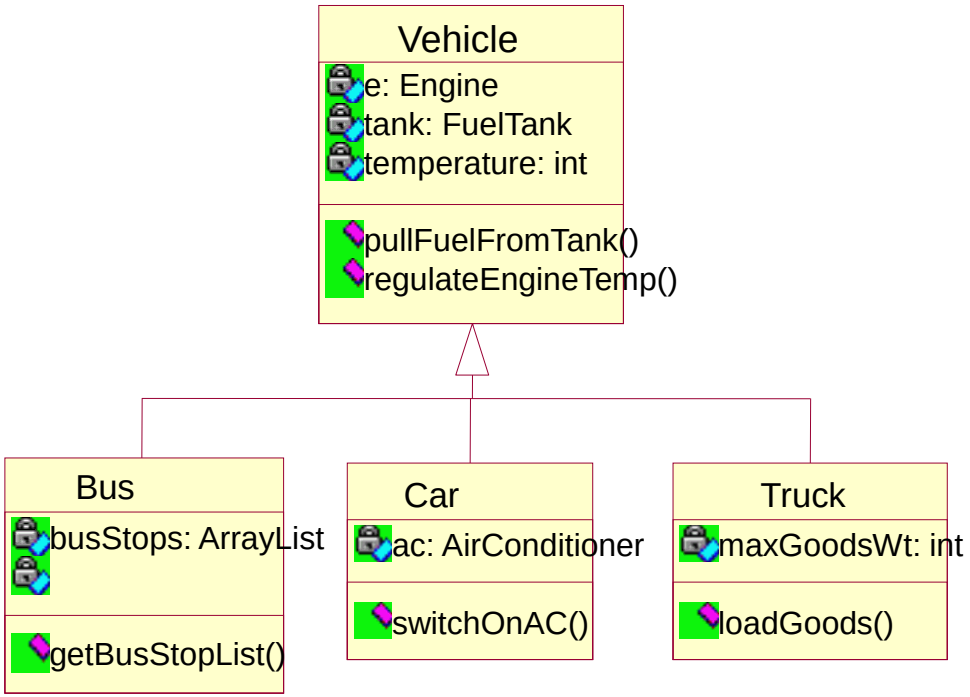
```
class AbstractDemo
{
    public static void main(String[] args)
    {
        Vehicle v = new Vehicle();
        Error – Abstract class cannot
        be instantiated
     }
 }
```



Sample Code : AbstractClassDemo.java

Abstract classes in Java

- In real world, Vehicle exists in more than one forms as Bus, Car, Truck and so on



- Classes Car, Bus, Truck all extend the abstract class Vehicle and reuse the structure and behavior defined.
- Vehicle, which has no real world mapping is not represented, not allowed to be instantiated in our application

PRATIAN TECHNOLOGIES

Abstract Classes

```
public abstract class Vehicle
{
      Engine e;
      FuelTank tank;
      void pullFuelFromTank(){.....}
      void regulateEngineTemperature(){.....}
      void start(){.....}
      void stop(){.....}
}
```

```
public class Car extends Vehicle
{
            switchOnAC(){…….}
}
```

```
public class Truck  extends Vehicle
{
            loadGoods(){…….}
            unloadGoods(){…….}
}
```

Why Abstract Classes?

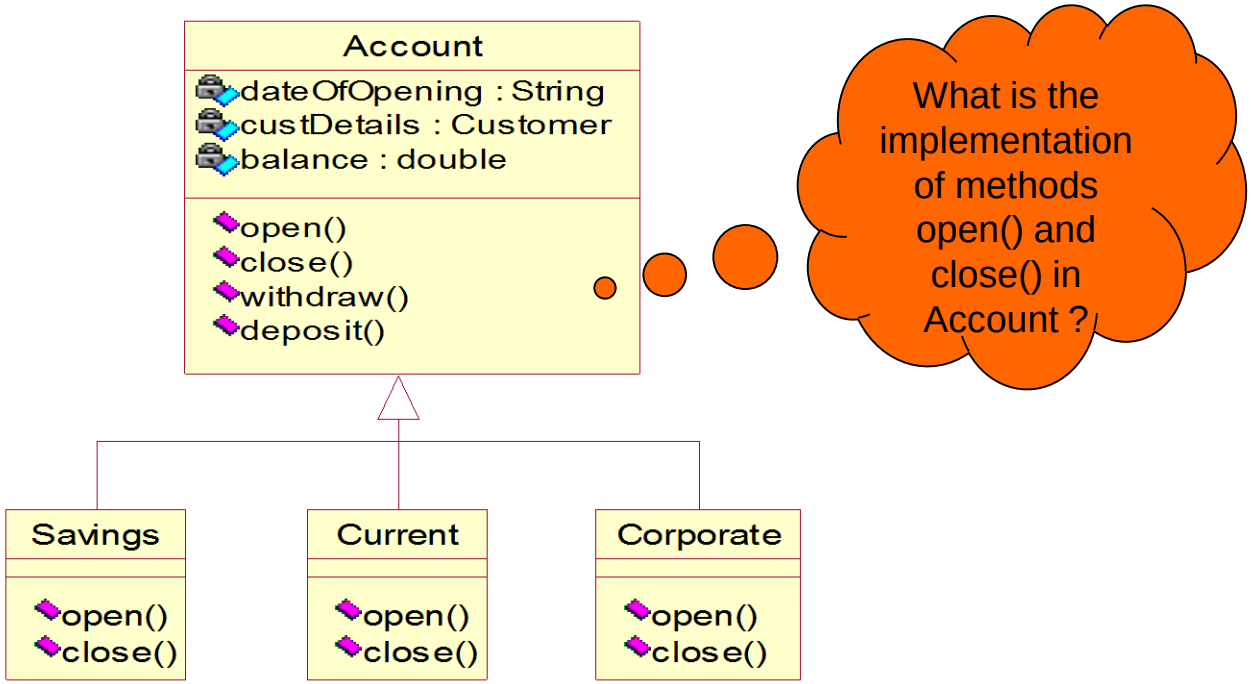
- To not allow instantiation of a class, since it has no real world representation.

- To capture all common structure and behavior in the base class, so that it can be reused by all sub types.

PRATIAN TECHNOLOGIES

Abstract Methods

- **An abstract method is a method with no implementation / body.**
- **Must be members of either abstract classes or interfaces.**



PRATIAN TECHNOLOGIES

Abstract Methods

```
abstract class Account
{
      private String dateOfOpening;
      private Customer custDetails;
      private double balance;

      public void withdraw(double amt)
      {
            // Implementation
       }
      public void deposit(double amt)
      {
            // Implementation
       }

      public abstract void open();
      public abstract void close();
}
```

A class with one or more abstract methods has to be marked as abstract.

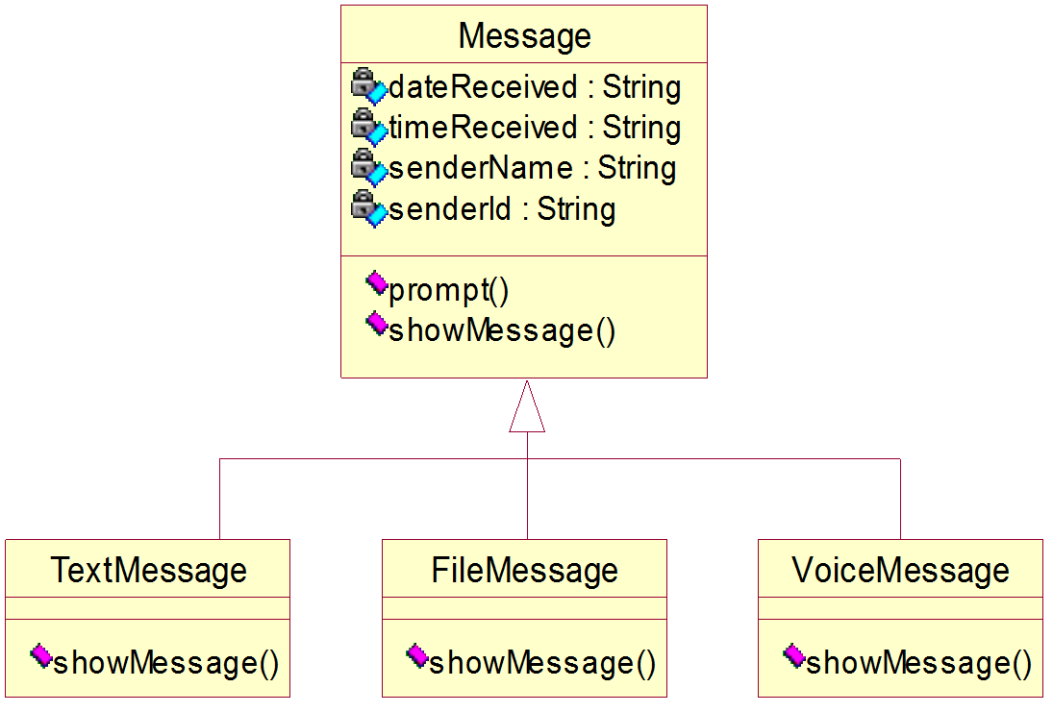
```
class Savings extends Account
{
      public void open()
      {       // Implementation
      }
      public void open()
      {       // Implementation
      }
}
```

```
class Current extends Account
{
      public void open()
      {       // Implementation
      }
      public void open()
      {       // Implementation
      }
}
```

Abstract Methods as a contract

- Abstract methods serve as a contract between design and implementation

- During design phase, it is convenient to identify and design abstract methods and defer the implementation details to the development phase.



Abstract Methods as a contract

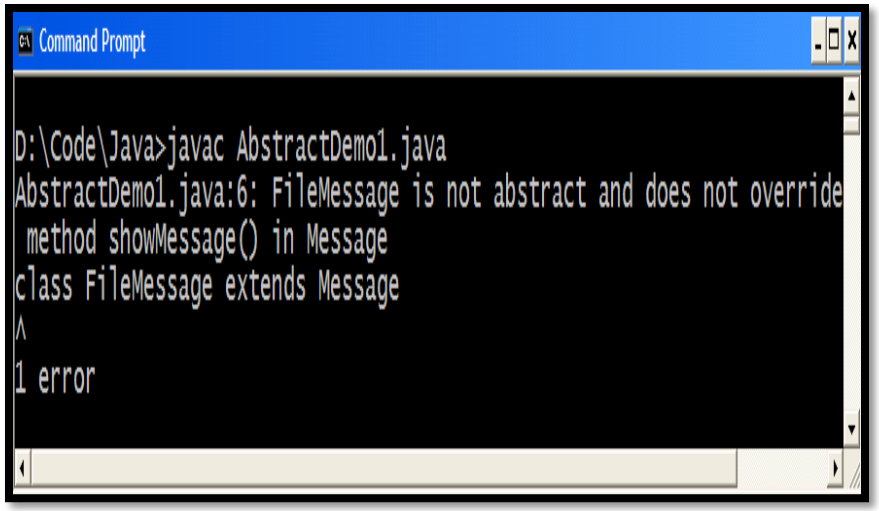
```
abstract class Message
{
      private String dateReceived;
      private String timeReceived;
      private SenderDetails sender;

      public String getDateReceived()
      {
            return dateReceived
       }
      public void setDateReceived(String dt)
      {
            dateReceived = dt;
       }

       public void prompt()
       {
              // logic of the method …
       }

      public abstract void showMessage();
}
```

PRATIAN TECHNOLOGIES

Abstract Methods as a contract

```
class TextMessage extends Message
{
    public void showMessage()
    {
        // Implementation
    }
}
```

```
class FileMessage extends Message
{
    public void getFileFormat()
    {
        // Implementation
    }
    public void launchApplication()
    {
        // Implementation
    }
}
```



FileMessage is required to override the method showMessage()

Polymorphism revisited…

- All sub classes of Message, namely,TextMessage, FileMessage and VoiceMessage are required to **override** the method - public void showMessage().
- A generic method can be designed which takes the **abstract type reference** as argument, to which any **derived type object** can be passed.

```
class MessageReceiver
{
    public void onMessage(Message msg )
    {
            msg.prompt();
            msg.showMessage();
    }
}
```

Abstract type reference

```
class MessageDemo
{
   private MessageReceiver mr = new
     MessageReceiver();
   public static void main(String[] s)
   {
       TextMessage t = new
                          TextMessage();
       FileMessage f = new
                           FileMessage();
        mr.onMessage(t);
        mr.onMessage(f);
    }
}
```

Derived type object

PRATIAN TECHNOLOGIES

Exercise

- Write an abstract class Shape with method double area(). Different sub classes of Shape namely Circle, Rectangle, Triangle override the area() method.
- Write a class AreaFinder that has a method showArea, that takes any Shape object and displays its area.
- Write a Demo class and test the code

Summarizing…

- An abstract class cannot be instantiated.
- An abstract class can contain both concrete methods as well as abstract methods.
- An abstract method has no body.
- Abstract methods in an abstract class need to be overridden by it's sub class

PRATIAN TECHNOLOGIES

Interfaces

- Interface is a java construct with method declarations only.
- Interface is a group of related methods with no implementation.

Defining an Interface

- The keyword ‘interface’ is used to define an interface.

- All methods declared within an interface are implicitly **public and abstract**

```
public interface CharSequence
{
    char charAt(int index);
    int length();
}
```

Abstract Methods

Implementing an interface

- A class then is said to ‘implement’ an interface, making use of the keyword ‘implements’.
- The class that implements the interface has to provide the logic for all methods defined in the interface.

```
public class String implements CharSequence
{
        public char charAt(int index)
        {
                // The implementation of this method
        }

        public int length()
        {
                 // The implementation of this method
        }
}
```

Interface as a 'type'

- When we define a new interface, we are defining a new reference data type
  - Very similar to defining a class, that is a reference data type

- To a reference variable whose type is an interface, an object of the class that implements the interface can be assigned.
  - For Example
    - **CharSequence cs = new String();**

PRATIAN TECHNOLOGIES

Interface as a contract

- Imagine an interface with some method signatures, and a class that will implement the interface.

Interface : I have 5 method signatures.
Class : I want to implement them.
Interface : Okay. But then you have to implement all of them. You are not allowed to say that you implement me without implementing every single one of my methods.
Class : It's a deal.

- An interface is a contract. It is a binding between the interface and the class that implements the interface.

PRATIAN TECHNOLOGIES

Interface as a contract

- Interface serves as a contract between design and implementation.
- At design time, it is convenient to discover what functionality needs to be achieved and specify as an interface.
- Based on varied implementation, different classes will implement the interface in different ways.
- Nevertheless, all classes need to conform to the contract.

Why implement an Interface?

- Implementing an interface allows a class to become more formal about the behavior it promises to provide.
- Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler.
- If a class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

Interface and constants

- An interface can also have data members defined in it, that serve as global constants.

- All data members defined in an interface are **implicitly public static final.**

```
interface OlympicMedal
{
        String GOLD = "Gold";
        String SILVER = "Silver";
        String BRONZE = "Bronze";
}
```

Constants being defined

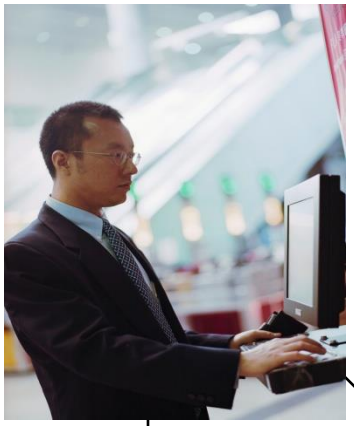
Interfaces and abstraction

- There are a number of situations in software engineering when it is important for disparate groups to work in parallel.
- Each group should be able to write their code without any knowledge of how the other group's code is written.
- Interfaces help in bringing about such an abstraction for the class users.
- Both the class creators and the class users heavily depend on the contract defined.
  - While the class creators actually ‘implement’ the contract, the class users simply ‘use’ the interface

PRATIAN TECHNOLOGIES

Interfaces and abstraction

Class Creator



defines

```
public interface ItemList
{
    void addItem (Object obj);
    Object[] getItems();
}
```

uses

Class User

Class Users are abstracted from implementation details

```
public class SequentialItemList
implements ItemList
{
    public void addItem (Object obj)  { …. }
    public Object[] getItems()  { …. }
}
```

```
public class RandomItemList
implements ItemList
{
    public void addItem (Object obj)  { …. }
    public Object [] getItems()  { …. }
}
```

```
public class Client {
    ItemList i1 = new SequentialItemList();
    i1.addItem(new Sale());
    Sale[] sales = (Sale[]) i1.getItems();

    ItemList i2 = new RandomItemList();
    i2.addItem(new Book());
    Book[] books = (Book[])
            i2.getItems();
}
```

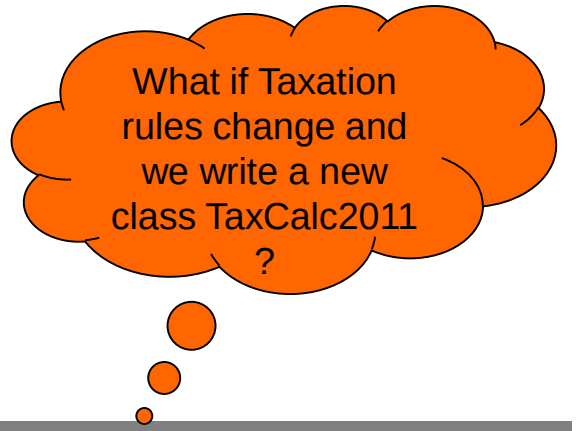
Programming to Interface

- It is a good programming practice to program to an interface and not to the implementation
- It is a good design to have methods take the generic interface type as arguments.
  - As discussed earlier, an interface reference can be used anywhere a type can be used.
- The advantage being, a change in the implementation would not have an impact on the client code.

PRATIAN TECHNOLOGIES

Programming to interface

```
public interface StdTaxCalc
{
    double getIncomeTax(salaryDetails details);
    double getFBT(InvestmentDetails details);
}
```




```
public class TaxCalc2006 implements StdTaxCalc
{
    double getIncomeTax(salaryDetails details)
    { …. }
    double getFBT(InvestmentDetails details)
    { …. }
}
```

```
public class SalaryCalculator
{
    void computeSalary()
    {       …..
        TaxCalc2006 tc = new TaxCalc2006();
        tc.getIncomeTax(sd);
        tc.getFBT(id);
    }
}
```

Programming to the Interface

- Always program to the interface type

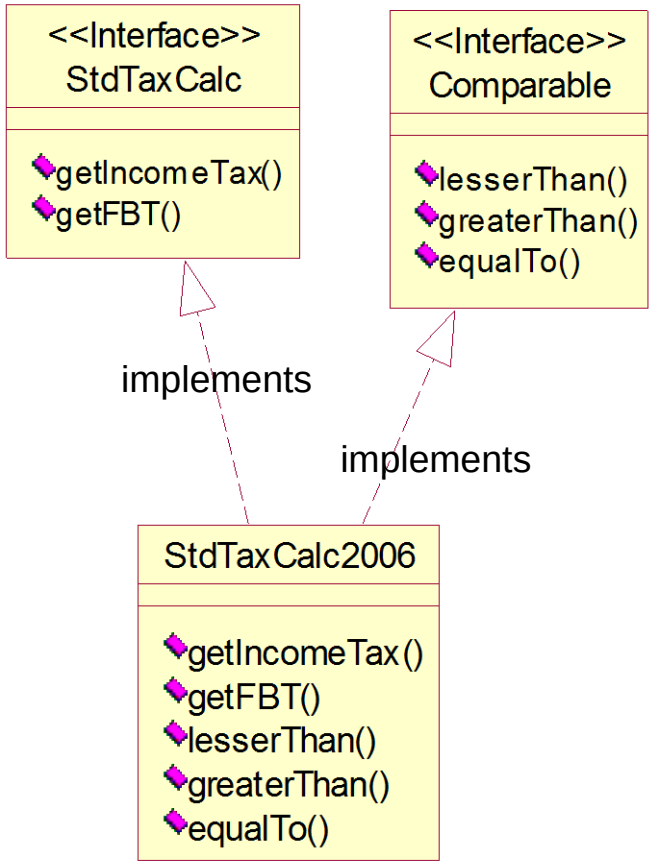
```
public class SalaryCalculator
{
    void computeSalary()
    {
        SalaryDetails sd;
        InvestmentDetails id;
        …..
        StdTaxCalc tc = TaxCalcCreator.getObject();
        tc.getIncomeTax(sd);
        tc.getFBT(id);
    }
}
```

Interface and Multiple Inheritance

- A class can implement multiple interfaces
  - Although the class can inherit only one other class.
- The class that implements multiple interfaces has to conform to multiple contracts
  - The class is required to provide implementation for all the methods defined in all the interfaces that it implements.
- Interface implementation is essentially behavioral reuse, while inheritance is structural reuse as well.
  - Hence, Java allows a class to implement multiple interfaces
- This feature is many times seen as Java's way of substituting multiple inheritance

PRATIAN TECHNOLOGIES

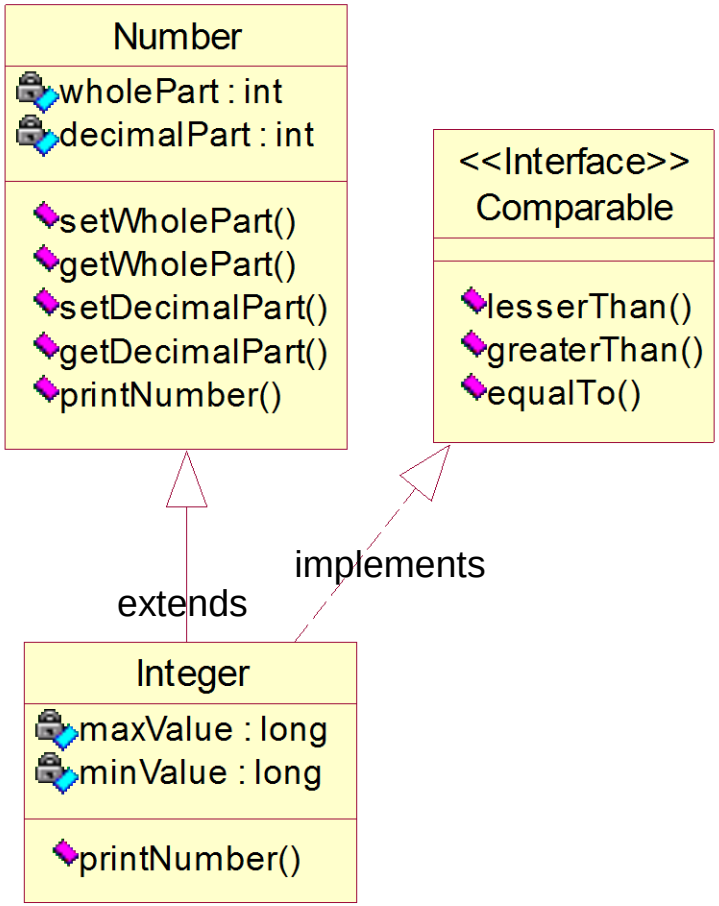
Interface and Multiple Inheritance



- TaxCalc2006 is one particular implementation of the StdTaxCalc interface.
- Let us suppose that it also needs to be Comparable
- Since we already have an interface Comparable defined, TaxCalc2006 implements two interfaces and conforms to both contracts

PRATIAN TECHNOLOGIES

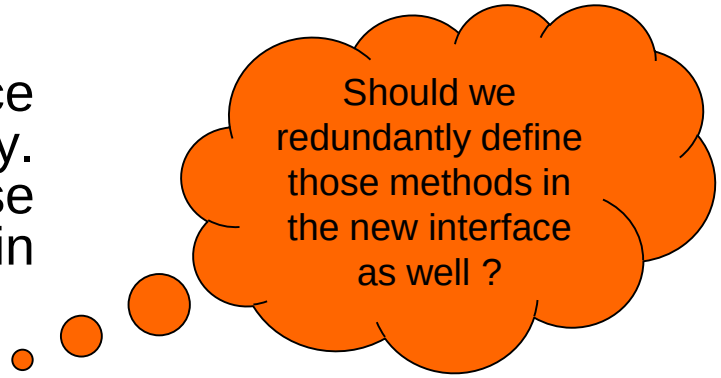
Interface and Multiple Inheritance



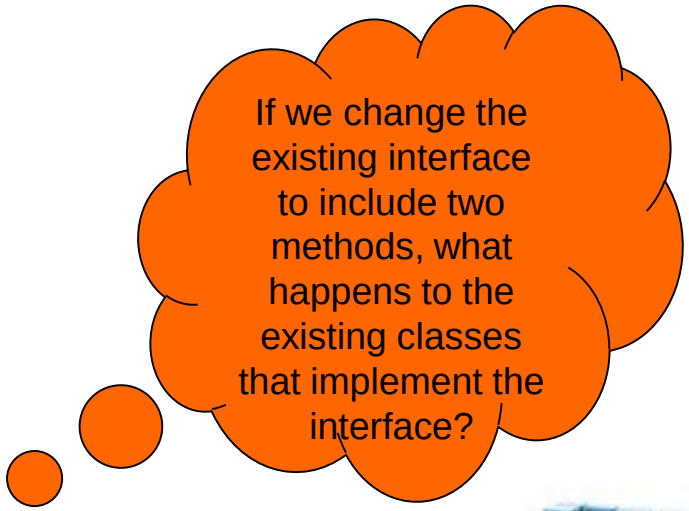
- Class Integer needs to conform to both contracts, namely, being a Number and being Comparable
- Class Integer therefore extends the abstract class Number and implements the Comparable interface

Inheritance among Interfaces

- Scenario 1
  - We have to define an interface that exposes some functionality. We realize that a few of those methods are already defined in another interface.



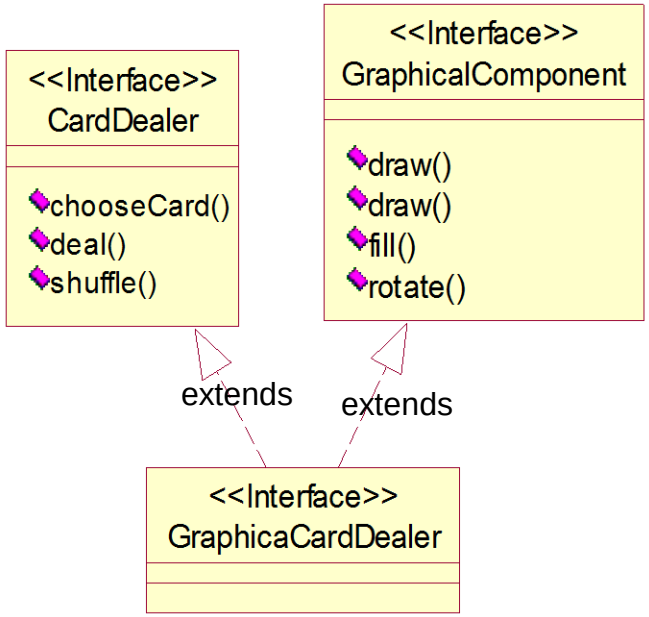
- Scenario 2
  - We have defined an interface with five method declarations. A few classes implement this interface and provide implementation for all the methods defined.
  - Let us suppose that we have received new requirements, because of which we are required to include two additional methods to the interface.



PRATIAN TECHNOLOGIES

Inheritance among interfaces

- As a solution to both problems, we have inheritance among interfaces.
- An interface can ‘extend’ one or more interfaces.



Summarizing…

- An interface defines a protocol of communication (contract) between two objects.
- An interface declaration contains method signatures, but no implementations, and might also contain constant definitions.
- A class that implements an interface must implement all the methods declared in the interface.
- Multiple interfaces can be implemented by a class.
- Interfaces help in bringing about abstractions
- **Always remember to program to the interface and not to the implementation**

PRATIAN TECHNOLOGIES

Question time

Please try to limit the questions to the topics discussed during the session. Thank you.