Abstract Classes and Interfaces

Unit 4

Pradeep LN

Topics

- Abstract classes and abstract methods
- Abstract methods as a contract between design and implementation
- Polymorphism revisited
- Interfaces
- Interface as a contract between design and implementation
- Class user vs. Class Creator revisited.
- Abstraction revisited.
- Programming to the interface vs. programming to the implementation
- Inheritance among interfaces

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.

Abstract classes in Java

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

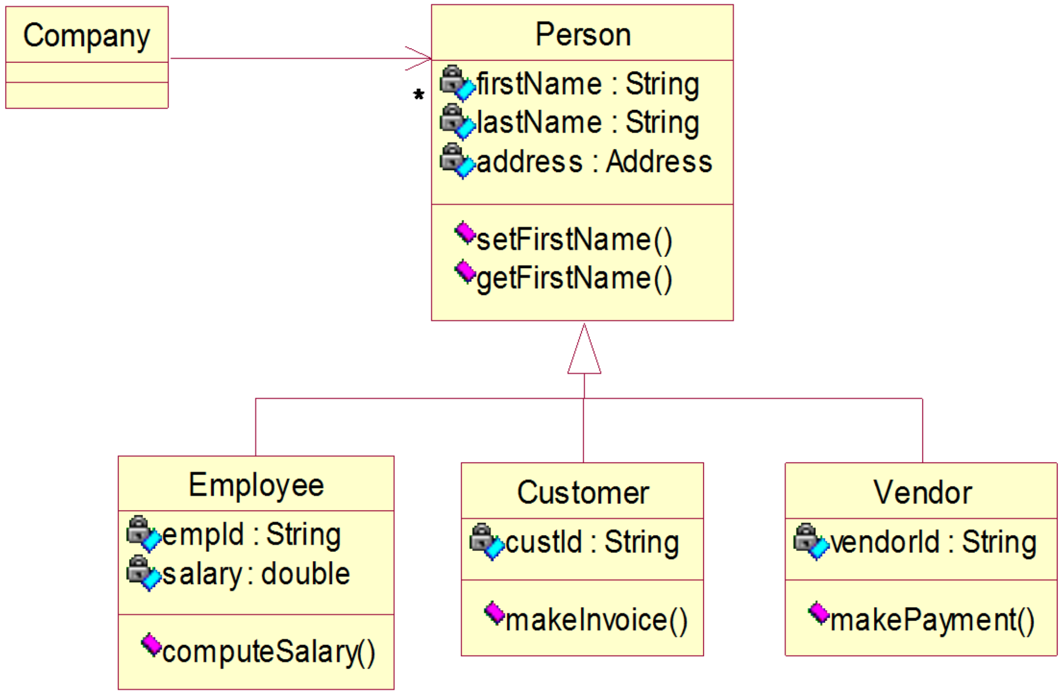
```
abstract class Person
{
     String firstName;
     String lastName;
     Address address;
     public void setFirstName
               (String name)
     {
          firstName = name;
     }
     // Other methods defined
}
```

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

How do we then use the functionality offered by this abstract class ?

Abstract classes in Java

- A company is associated with many persons, like employee, customer, vendor.



- Person exists as one or more of its sub types – Employee, Customer, Vendor
- Classes Employee, Customer, Vendor all extend the abstract class Person and reuse the structure and behavior defined.

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.

Why Abstract classes ?

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

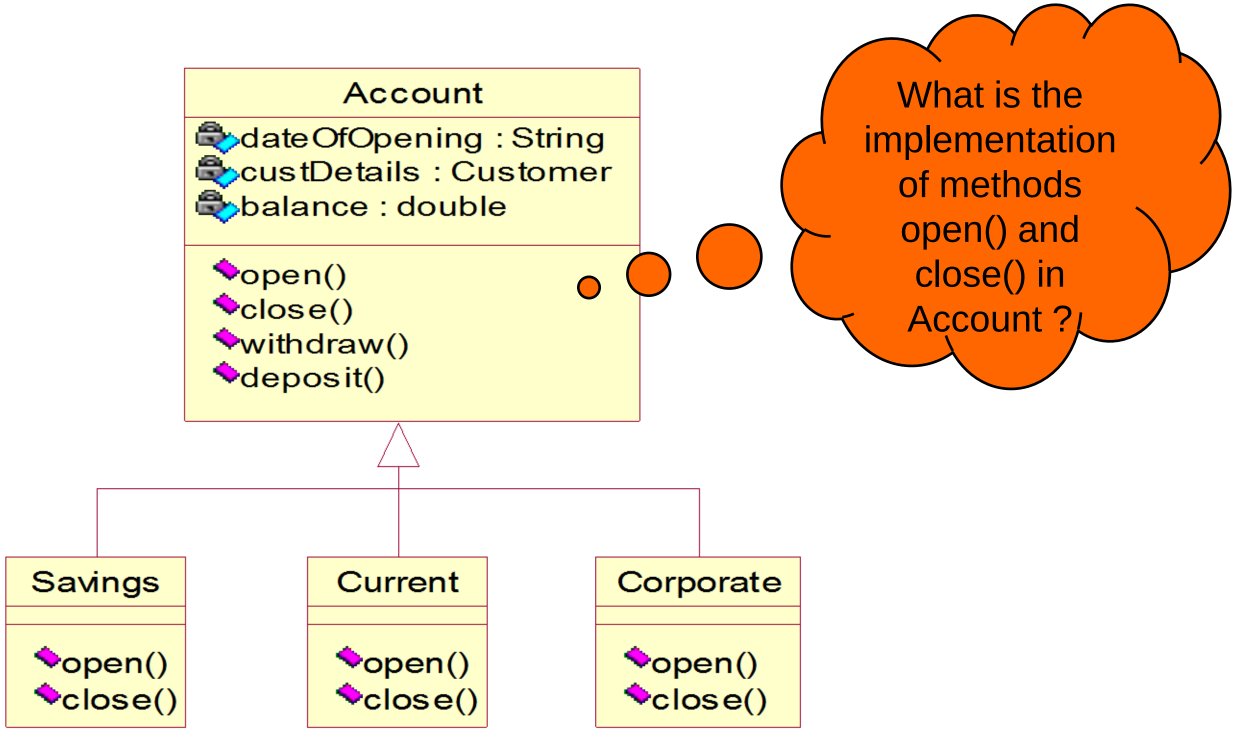
```
public class Car extends Vehicle
{
            switchOnAC(){…….}
}
```

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

See Listing : AbstractClassDemo.java

Abstract Methods

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



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 close()
      {         // Implementation
      }
}
```

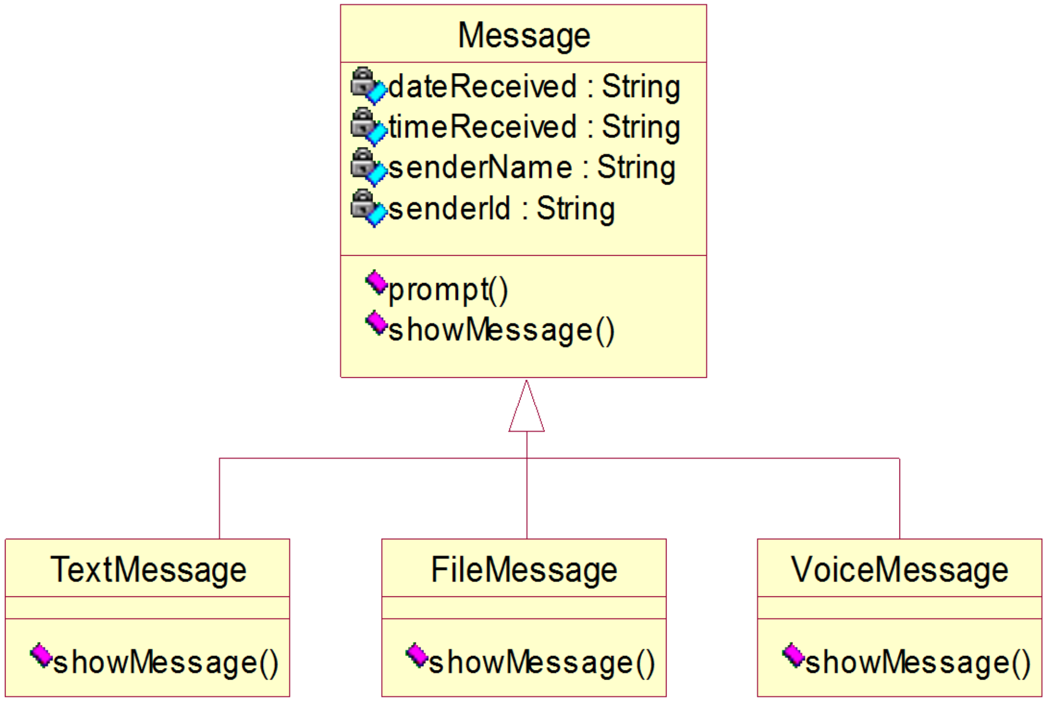
```
class Current extends Account
{
      public void open()
      {         // Implementation
      }
      public void close()
      {         // Implementation
      }
}
```

- See Listing : **AbstractMethodDemo.java**

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

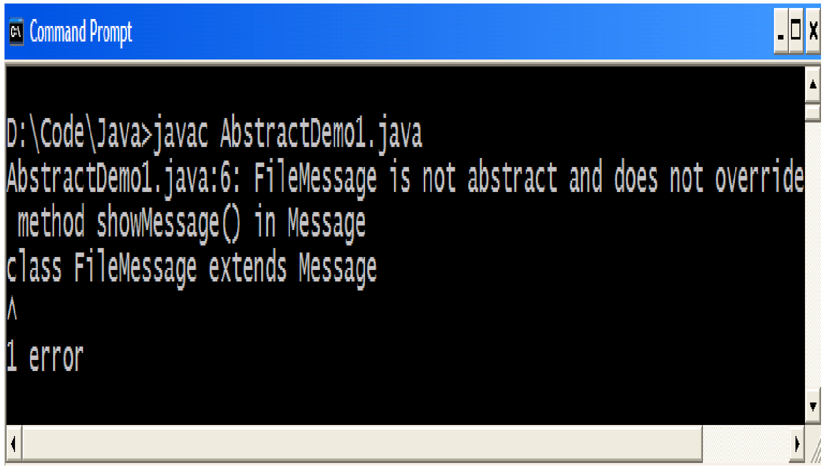
```
abstract class Message
{
      private String dateReceived;
      private String timeReceived;
      private SenderDetails sender;

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

      // Other methods defined

      public abstract void showMessage();
}
```

Abstract Methods as a contract

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

TextMessage is required to override the method showMessage( )

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



```
D:\Code\Java>javac AbstractDemo1.java
AbstractDemo1.java:6: FileMessage is not abstract and does not override
 method showMessage() in Message
class FileMessage extends Message
^
1 error
```

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 referenc e

```
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

Exercise

- Write an abstract class Shape with method double area(). Different sub classes of shape 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

Interfaces

- Interface is a java construct with method declarations only.
- Interface is a group of related methods with no implementation.
- An abstract class with only abstract methods can be conveniently defined as an interface

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();**

See Listing : **InterfaceDemo.java**

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.

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.

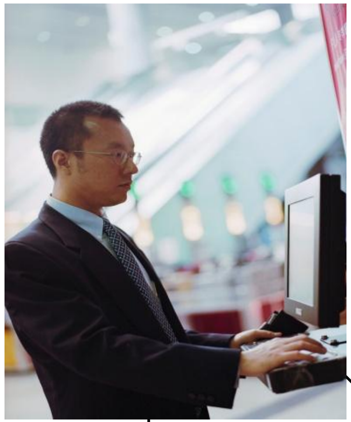
```
interface OlympicMedal
{
        String GOLD = "Gold";
        String SILVER = "Silver";          Constants being defined
        String BRONZE = "Bronze";
}
```

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

Interfaces and abstraction

Class Creator



defines

```
public interface Stack
{
    void push (Object obj);
    Object pop();
}
```

uses

Class User

Class Users are abstracted from implementation details

```
public class FixedStack implements Stack
{
  public void push (Object obj)
{ …. }
public Object pop()
{ …. }
}
```

```
public class DynStack implements Stack
{
  public void push (Object obj)
  { …. }
public Object pop()
  { …. }
}
```

```
pu  c class Client {
   Stack s1 = new FixedStack();
   s1.push(“Java”);
   String str = (String) s1.pop();

   Stack s2 = new DynStack();
   s2.push(new Customer());
   Customer cust = (Customer) s2.pop;
}
```

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.

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)
      { …. }
}
```

What if Taxation rules change and we write a new class TaxCalc2007 ?

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

Programming to the Interface

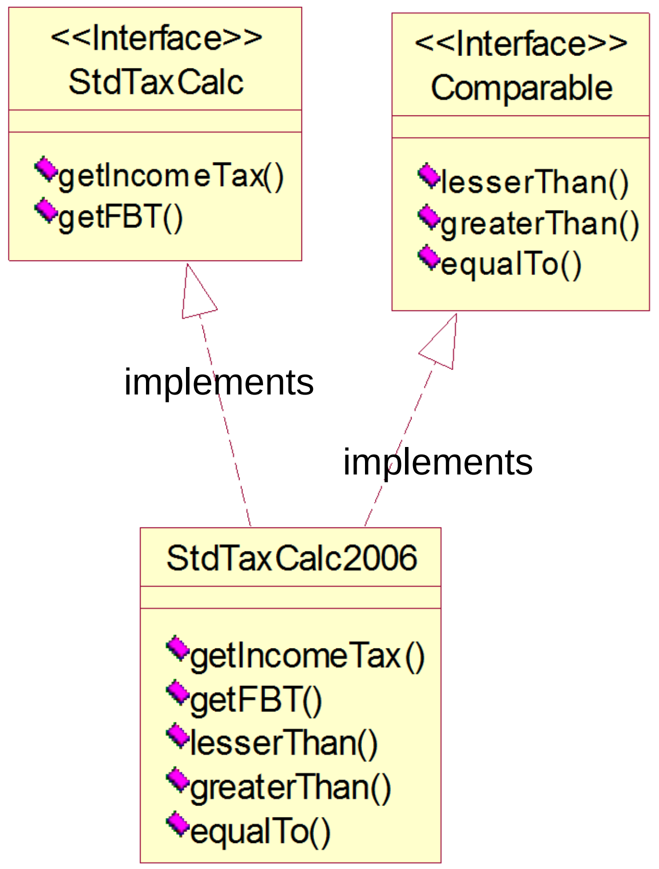
- Always program to the interface type

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

Interface and Multiple Inheritance

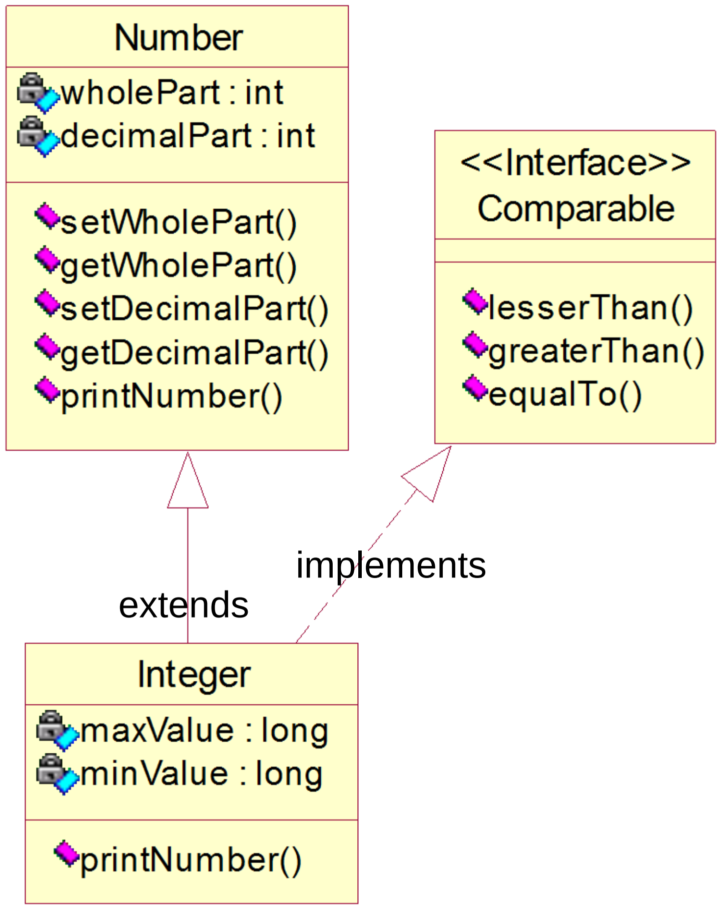
- A class can implement multiple interfaces
  - Although the class can inherit only one other class.
- The class that implements multiple interfaces has to conform to multiple contracts
  - The class is required to provide implementation for all the methods defined in all the interfaces that it implements.
- 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 simulating multiple inheritance

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

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.

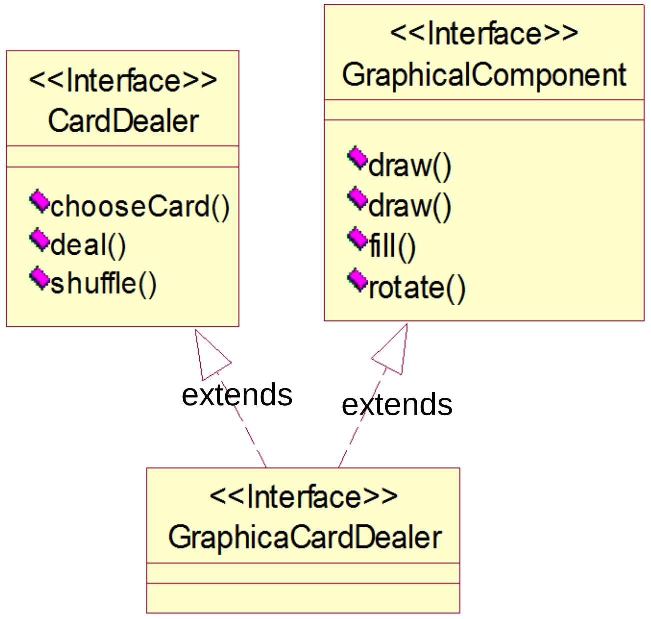
Should we redundantly define those methods in the new interface as well ?

- 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.

If we change the existing interface to include two methods, what happens to the existing classes that implement the interface?

Inheritance among interfaces

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



- See Listing : InterfaceInheritance.java

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**

Question time