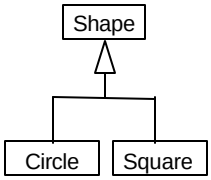
Abstract Classes a and Interfaces

Abstract Classes

Class Shape Hierarchy

- Consider the following class hierarchy



Problem AND Requirements

- Suppose that in order to exploit polymorphism, we specify that 2-D objects must be able to compute their area.
  - All 2-D classes must respond to area() message.
- How do we ensure that?
  - Define area method in class Shape
  - Force the subclasses of Shape to respond area() message
- Java's Solutions
  - Abstract Classes
  - Interfaces

Abstract Classes

- Idea
  - To define only part of an implementation
    - Can contain instance variables & methods that are fully implemented
  - Leaving the subclasses to provide the details
- Any class with an abstract method must be declared abstract
  - However you can declare a class abstract that has no abstract method.
  - An abstract method has no implementation (known in C++ as a pure virtual function)

Abstract Classes

- If subclass overrides all abstract methods of the superclass, than it becomes a concrete class otherwise we have to declare it as abstract or we can not compile it
- Any subclass can override a concrete method inherited from the superclass and declare them abstract
- An abstract class cannot be instantiated
- However references to an abstract class can be declared
  - Can point to the objects of concrete subclasses

Example of abstract class Shape.java

```
/* This is an example of abstract class. Note that
this class contains an abstract method with no
definition.
*/
public abstract class Shape {
    public abstract void calculateArea();
}
```

Circle.java

```
/* This class extends from abstract Shape class. Therefore to
become concrete class it must provides the definition of
calculateArea method.
*/
public class Circle extends Shape {
   private int x, y;
   private int radius;
   public Circle() {
      x = 5;
      y = 5;
      radius = 10;
   }
// continue
```

Circle.java

```
  // providing definition of abstract method
  public void calculateArea () {
     double area = 3.14 * (radius * radius);
     System.out.println("Area: " + area);
  }

}//end of class
```

Test.java (Driver class)

```
public class Test  {
  public static void main (String args[ ]){
    //can only create references of abstract class
    Shape s = null;
     // Shape s1 = new Shape();  //cannot instantiate abstract class

     //can point to the concrete subclass
     s = new Circle();
     s.calculateArea();
  }
}
```

Compile & Execute

```
C:\WINDOWS\system32\cmd.exe
D:\examples\abstract>javac Shape.java
D:\examples\abstract>javac Circle.java
D:\examples\abstract>javac Test.java
D:\examples\abstract>java Test
Area:314.0
D:\examples\abstract>
```

Interfaces

Interfaces

– A special java type which

- Defines a set of method prototypes, but does not provide the implementation for the prototypes
- Essentially all the methods inside an interface are Abstract Methods or we can say that an interface is like a pure abstract class (Zero Implementation)
- Can also define static final constants

Interfaces Definition Example

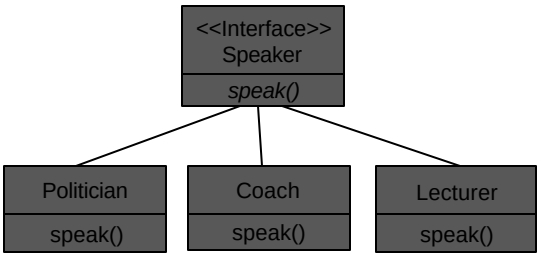
- Syntax (appears like abstract class):
- All methods are abstract and public by default
- All constants are static and final by default

```
public interface Speaker {

    public void speak( );

}
```

Implementing (Using) Interfaces

- Classes Implement interfaces
  - Implementing an interface is like signing a contract.
  - A class that implements an interface will have to provide the definition of all the methods that are present inside an interface"
  - If the class does not provide definitions of all methods, the class would not compile. We have to declare it as an abstract class in order to get it compiled.
- "Responds to" relationship
  - Relationship between a class and interface

Interface - Example



Implementing Interfaces Example

```
class  Politician implements Speaker {
        public void speak(){
            System.out.println("Talk politics");
        }
}
```

```
class  Coach implements Speaker {
        public void speak(){
            System.out.println("Sports Talks");
        }
}
```

```
class  Lecturer implements Speaker {
        public void speak(){
            System.out.println("Web Desing and Development Talks");
        }
}
```

Example Code

Defining Interface

```
public interface Printable {
  public void print();
}
```

Implementing Interface

```
public class Student implements Printable{
  private String name;
  private String address;

  public String toString () {
    return "name:"+name +" address:"+address;
  }
  // NOT providing implementation of print method
}
```

Compile

```
C:\WINDOWS\system32\cmd.exe
D:\examples\interface>javac Student.java
Student.java:1: Student is not abstract and does not override abstract
 method print() in Printable
public class Student implements Printable{
       ^
1 error

D:\examples\interface>_
```

Example Code (cont.)

Implementing Interface (Modification)

```
public class Student implements Printable{
  private String name;
  private String address;

  public String toString () {
    return "name:"+name +" address:"+address;
  }
  public void print() {
     System.out.println("Name:" +name+" address"+address);
  }

}
```

Compile

```
C:\WINDOWS\system32\cmd.exe
D:\examples\interface>javac Student.java
D:\examples\interface>_
```

More on Interfaces

- Interface imposes a design structure on any class that uses the interface
- Leaves the implementation details to the implementing class and hides that implementation from the client.
- A class can implement more than one interfaces. Java's way of multiple inheritance

```
class Circle implements Drawable, Printable {
  //additional constants and abstract methods
}
```

More on Interfaces (cont.)

- Classes inherit from classes (Single), interfaces inherit from interfaces (Can be multiple) and classes implement interfaces (Can be multiple)

```
public interface Displayable extends Drawable, Printable {
        //additional constants and abstract methods
}
```

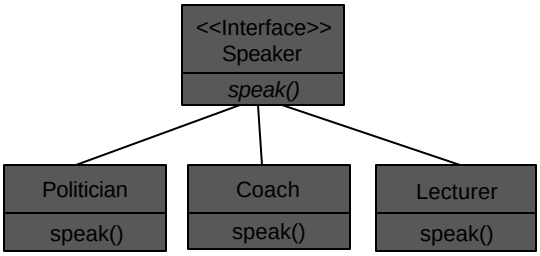
- Objects of interfaces cannot be instantiated.

```
Speaker sp = new Speaker();   // not comaile
```

- However a reference of interface can be created to point to any of its implementation class (Interface based polymorphism).

Interface based Polymorphism

Review again – Interface Example



Example: Interface based Polymorphism

```
/* Speaker interface is implemented by the Politician, Coach and
Lecturer class.    */

public class Test{
   public static void main (String args[ ]) {

      Speaker sp = null;

      System.out.println("sp pointing to Politician");
      sp = new Politician();
      sp.speak();

      System.out.println("sp pointing to Coach");
      sp = new Coach();
      sp.speak();

      System.out.println("sp pointing to Lecturer");
      sp = new Lecturer();
      sp.speak();
   }
}
```

Interface based Polymorphism Compile & Execute

```
C:\WINDOWS\system32\cmd.exe

D:\examples\interface\polymorphism>javac Speaker.java

D:\examples\interface\polymorphism>javac Politician.java

D:\examples\interface\polymorphism>javac Coach.java

D:\examples\interface\polymorphism>javac Lecturer.java

D:\examples\interface\polymorphism>javac Test.java

D:\examples\interface\polymorphism>java Test
sp pointing to Politician
Politics Talk
sp pointing to Coach
Sports Talk
sp pointing to Lecturer
Web design and Development Talks
```

Interfaces vs. Abstract classes

- Fairly similar uses
  - designed to group behavior, allow upcasting, exploit polymorphism
- Rules of thumb
  - Choose abstract class if we have shared code and logical “is a” relationship
  - Choose interface if only want to ensure design structure (method signatures) and/or it is not logical to use “is a “ relationship.