Interface Class

Lecture 22

Based on Slides of Dr. Norazah Yusof

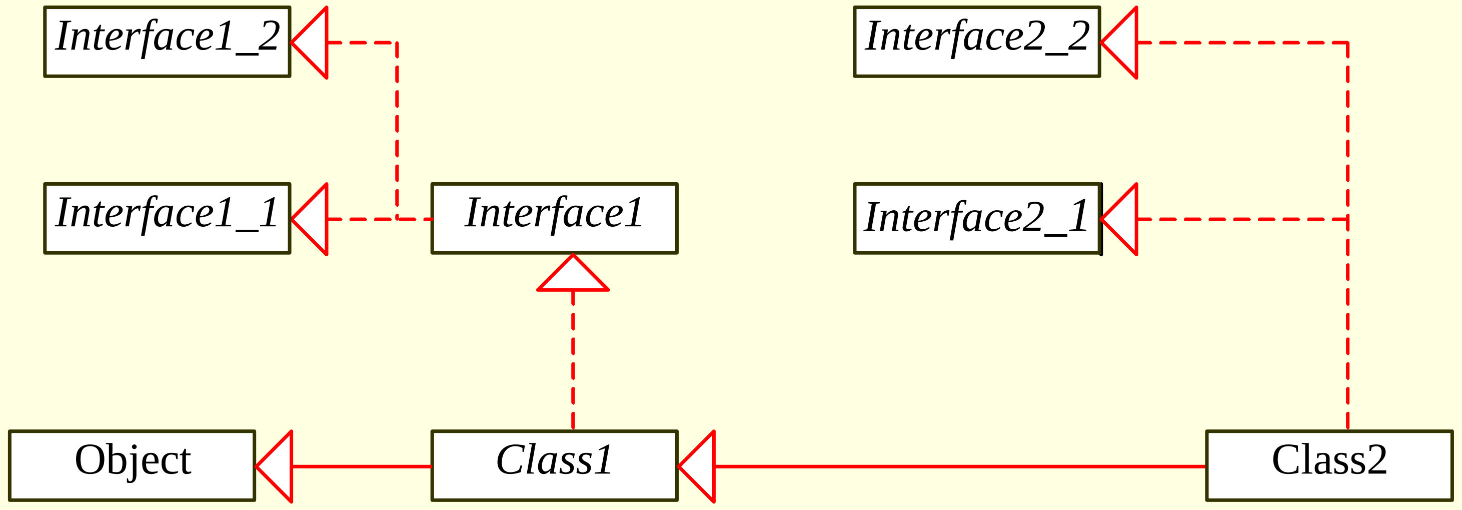
Interface & Implements

- **An interface is a classlike construct that contains only constants variables and abstract methods definition.**
- **An interface cannot contain instance variable and solid methods.**
- **An interface specifies what methods will be implemented by classes that *implements* an interface.**
- **This is another way to design polymorphic methods.**

Interface

- **All classes share a single root, the Object class, but there is no single root for interfaces.**
- **Like a class, an interface also defines a type.**
- **A variable of an interface type can reference any instance of the class that implements the interface.**
- **If a class extends an interface, this interface plays the same role as a superclass. You can use an interface as a data type and cast a variable of an interface type to its subclass, and vice versa.**

Interface



Suppose that c is an instance of Class2. c is also an instance of Object, Class1, Interface1, Interface1_1, Interface1_2, Interface2_1, and Interface2_2.

Interface declaration

- **Declaration:**

```
public interface InterfaceName
{
   /* constant declaration */
   /* method declarations (without
      implementations.) */
}
```

  - **Constant declaration→public, static and final**
  - **Method→abstract, public**

Interface declaration

In an interface, by default all data fields are constant variables and defined as *public final static* and all methods are *public abstract*.

For this reason, these modifiers can be omitted, as shown below:

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

A constant defined in an interface can be accessed using syntax InterfaceName.CONSTANT_NAME (e.g., T1.K).

Ex: Interface declaration (constant variables)

```
public interface Conversionfactors
{
    double inToMm = 25.3;
    double ounToGram = 28.34952;
    double poundToGram = 453.5924 ;
}
```

- **Save as ConversionFactors.java**

Ex: Interface declaration (Abstract Methods)

```
public interface Conversions {
  public double InToMm();
  public double OunceToGram();
  public double PoundToGram();
}
```

- **Save as Conversions.java**

Ex: Interface declaration (variables and Methods)

```
public interface MyInterface     {
    public final int aConstant = 32;
    public final double pi = 3.14159;
    public void methodA(int x);
    double methodB();
}
```

Implementing an Interface

- A class may extends one parent (superclass), but it can implement none or more interfaces
- A class that implements an interface:
  - Can access directly all the variables declared in the interface
  - Have to redefined all the methods declared in the interface
  - Declaration:

```
class SomeClass extends Parent implements SomeInterface

      { ordinary class definition body     }
```

- **The body of the class definition is the same as always.**
- **If any of the abstract methods (in the interface) are not defined in the class that implements the interface, the class will become an abstract class**

Example 1

```
public interface Speakable {
  public String speak();
}

public class Animal {
  protected String kind;
  public Animal() { };
  public String toString(){
     return "I am a " + kind + " and I go " +
     ((Speakable)this).speak();
 }
}
```

Example 1

```
public class Cat extends Animal implements Speakable {
  public Cat() {
    kind ="cat"; }
  public String speak() {
    return "meow"; }
}

public class Cow extends Animal implements Speakable {
  public Cow() {
    kind ="cow"; }
  public String speak(){
    return "moo"; }
}
```

Interface vs Abstract

	Variables	Constructors	Methods
Abstract class	**No restrictions**	**Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.**	**No restrictions.**
Interface	**All variables must be <u>public static final</u>**	**No constructors. An interface cannot be instantiated using the new operator.**	**All methods must be public abstract instance methods**

Comparable Interface

```
// This interface is defined in
// java.lang package
package java.lang;

public interface Comparable {
  public int compareTo(Object o);
}
```

Determine the order of given object with the specified object o,
and return a negative integer if this object is less than,
    return a zero if this object is equal,
    return a positive integer if this object is greater than,
the specified object o.

String and Date Classes

Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects. If you examine the source code of these classes, you will see the keyword implements used in the classes, as shown below:

```
public class String extends Object
     implements Comparable {
   // class body omitted
 }
```

```
public class Date extends Object
     implements Comparable {
   // class body omitted
 }
```

```
new String() instanceof String
new String() instanceof Comparable
new java.util.Date() instanceof java.util.Date
new java.util.Date() instanceof Comparable
```

Generic `max` Method

```
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum of two objects */
  public static Comparable max
      (Comparable o1, Comparable o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

(a)

```
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum of two objects */
  public static Object max
      (Object o1, Object o2) {
    if (((Comparable)o1).compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

(b)

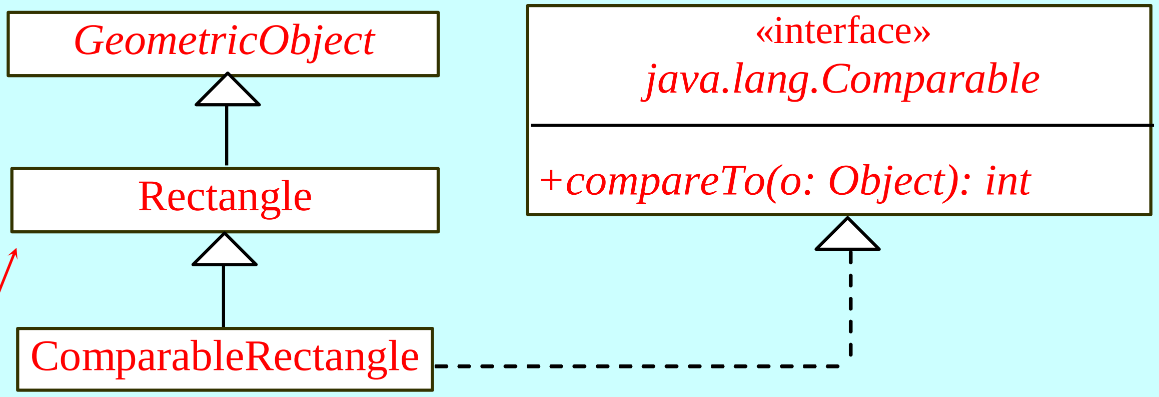
```
String s1 = "abcdef";
String s2 = "abcdee";
String s3 = (String)Max.max(s1, s2);
```

```
Date d1 = new Date();
Date d2 = new Date();
Date d3 = (Date)Max.max(d1, d2);
```

The return value from the max method is of the Comparable type. So, you need to cast it to String or Date explicitly.

Declaring Classes to Implement Comparable

Notation:
The interface name and the method names are italicized. The dashed lines and hollow triangles are used to point to the interface.



You cannot use the max method to find the larger of two instances of Rectangle, because Rectangle does not implement Comparable. However, you can declare a new rectangle class that implements Comparable. The instances of this new class are comparable. Let this new class be named ComparableRectangle.

```
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```

Exercises (Lab 10)

- Self tests
  - Question 3, page 194
- Lab Exercises:
  - Question 2, page 192
  - Question 4, page 194
  - Question 1, page 196