26. Interfaces

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Definition
- The Comparable Interface
- Interfaces vs. Abstract Classes
- Creating Custom Interfaces
- References

Definition

Definition

- **Single Inheritance**
  - A Java class may inherit directly from only one superclass.
  - This restriction is known as **single inheritance**.
- **Multiple Inheritance**
  - Sometimes it is necessary to derive a subclass from several classes.
  - This capability is known as **multiple inheritance**.
  - Java, however, does not allow multiple inheritance.

Definition

- If you use the extends keyword to define a subclass, it allows only one parent class.
- **Interfaces**
  - With **interfaces**, you can obtain the effect of multiple inheritance.
  - An interface is similar to an abstract class, but
    - An **interface** contains only **constants and abstract methods.**
    - An **abstract class** can contain variables and concrete methods as well as constants and abstract methods.

Definition

- To distinguish an interface from a class, Java uses the following syntax to declare an interface:

```
modifier interface InterfaceName
{
        /** Constant declarations */
        /** Method signatures */
}
```

Definition

- An interface is treated like a special class in Java.
- Each interface is compiled into a separate bytecode file, just like a regular class.
- As with an abstract class, you cannot create an instance from an interface using the new operator

The Comparable Interface

The Comparable Interface

- Suppose you want to design **a generic method** to find the larger of two objects.
- The objects can be students, circles, or rectangles.
- Since compare methods are different for different types of objects, you need to define a generic compare method to determine the order of the two objects.
- For example, you can use
  - **student ID** as the key for comparing students,
  - **radius** as the key for comparing circles, and
  - **area** as the key for comparing rectangles.

The Comparable Interface

- You can use an interface to define a generic compareTo method, as follows:

```
// Interface for comparing objects, defined in java.lang
package java.lang;
public interface Comparable
{
        public int compareTo(Object o);
}
```

- The compareTo method determines the order of this object with the specified object o, and returns a negative integer, zero, or a positive integer if this object is less than, equal to, or greater than the specified object o.

The Comparable Interface

- Many classes in the Java library (e.g., String and Date) implement Comparable to define a natural order for the objects.

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

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

- Thus strings are comparable, and so are dates. Let s be a String object and d be a Date object. All the following expressions are all true:

```
s instanceof String
s instanceof Object
s instanceof Comparable
```

```
d instanceof java.util.Date
d instanceof Object
d instanceof Comparable
```

Interfaces vs. Abstract Classes

Interfaces vs. Abstract Classes

- In an interface, the data must be constants; an abstract class can have all types of data.
- Each method in an interface has only a signature without implementation; an abstract class can have concrete methods.

	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 public static final	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

Interfaces vs. Abstract Classes

- Since all data fields are public static final and all methods are public abstract in an interface, Java allows these modifiers to be omitted.
- Therefore the following declarations are equivalent:

```
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 the syntax InterfaceName.CONSTANT_NAME (e.g., T1.K).

Interfaces vs. Abstract Classes

- Java allows only single inheritance for class extension, but multiple extensions for interfaces.
- For example,

```
public class NewClass extends BaseClass implements
Interface1, ..., InterfaceN
{
        ...
}
```

Interfaces vs. Abstract Classes

- An interface can inherit other interfaces using the extends keyword.
- Such an interface is called a **subinterface**.
- For example, NewInterface in the following code is a subinterface of Interface1, ..., and InterfaceN:

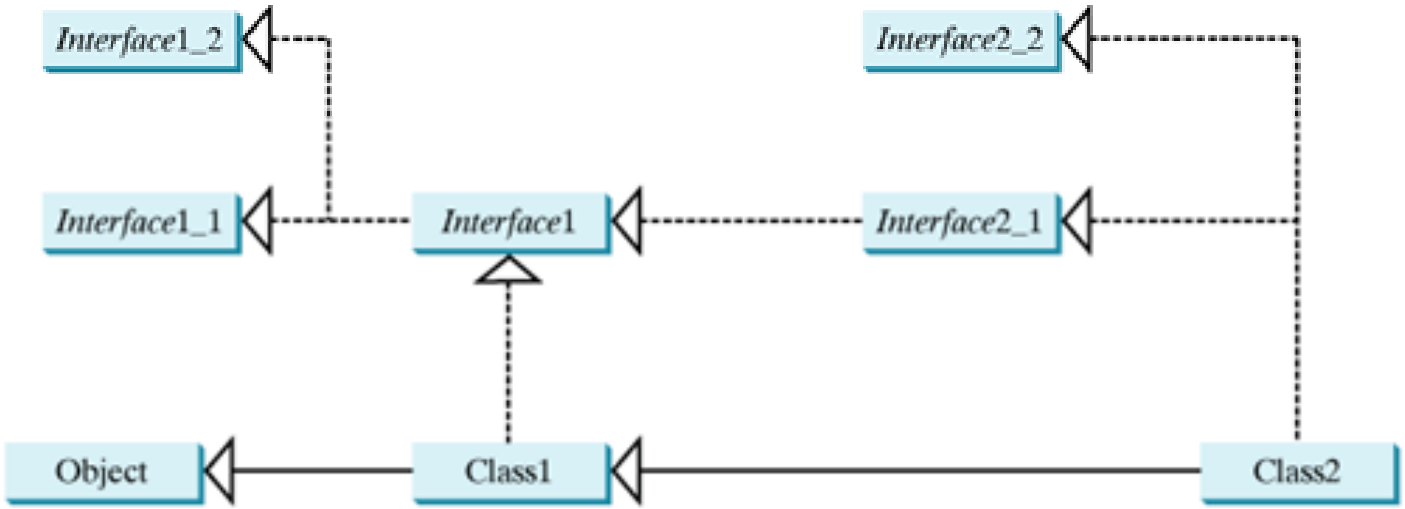
```
public interface NewInterface extends Interface1, ..., InterfaceN
{
        // constants and abstract methods
}
```

Interfaces vs. Abstract Classes

- 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.

Interfaces vs. Abstract Classes

- Abstract class Class1 implements Interface1, Interface1 extends Interface1_1 and Interface1_2.
- Class2 extends Class1 and implements Interface2_1 and Interface2_2.



- 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.

Interfaces vs. Abstract Classes

- Class names are **nouns**.
- Interface names may be **adjectives** or **nouns**.
- For example, both java.lang.Comparable and java.awt.event.ActionListener are interfaces.
- Comparable is an adjective, and ActionListener is a noun.

Creating Custom Interfaces

Creating Custom Interfaces

- Suppose you want to describe whether an object is edible.
- You can declare the Edible interface.
- To denote that an object is edible, the class for the object must implement Edible.
- Create a class named Animal and its subclasses Tiger, Chicken, and Elephant.
- Create a class named Fruit and its subclasses Apple and Orange.

Creating Custom Interfaces

- The programs:
  - Edible.java
  - Fruit.java
  - Animal.java
  - TestEdible.java

Creating Custom Interfaces

- Since chicken is edible, implement the Edible interface for the Chicken class.
- The Chicken class also implements the Comparable interface to compare two chickens
- The Fruit class is abstract, because you cannot implement the howToEat method without knowing exactly what the fruit is.

References

References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 10)

The End