

Interface

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, April 29, 2013

Chapter 13: Interface and Inner Classes

- will teach inner classes later (with Swing)
- let's learn interface first

what devices come with headset jacks?

Interface: “Headset Jack” of Instances (1)

```
1 interface ListenableWithHeadset{
2     void playto(Headset h);
3 }
4 abstract class Player implements ListenablewithHeadset{
5     abstract void playto(Headset h);
6 }
7 class MP3Player extends Player{ //implements ListenablewithHeadset
8     void playto(Headset h){
9         // ...
10    }
11 }
12 class Laptop extends Computer implements ListenablewithHeadset{
13     void playto(Headset h){
14         // ...
15    }
16 }
17 class iPhone extends MobilePhone implements ListenablewithHeadset{
18     void playto(Headset h){
19         // ...
20    }
21 }
```

Interface: “Headset Jack” of Instances (2)

```
1 interface ListenableWithHeadset{
2     void playto(Headset h);
3 }
4 class Headset{
5     void listen(ListenableWithHeadset device){ device.playto(this); }
6 }
7 class MP3Player extends Player{ //implements ListenablewithHeadset
8     void playto(Headset h){ }
9 }
10 class Laptop extends Computer implements ListenablewithHeadset{
11     void playto(Headset h){ }
12 }
13 class iPhone extends MobilePhone implements ListenablewithHeadset{
14     void playto(Headset h){ }
15 }
```

- anything that implements `ListenablewithHeadset` can be coupled with the `Headset`
- `ListenablewithHeadset` is a **trait** (type)
- interface relationship: not by blood (inheritance), but by **ability**

Interface: Like a Miniature Abstract Class (1)

```
1 interface ListenableWithHeadset{
2     void playto(Headset h);
3 }
4 abstract class Player{
5     public abstract void playto(Headset h);
6 }
7 class Headset{
8     void listen1(ListenableWithHeadset device){
9         device.playto(this);
10    }
11    void listen2(Player device){ device.playto(this); }
12 }
```

- multiple inheritance with abstract/concrete classes: no
- approximate-multiple inheritance with interfaces and one ancestor class: **yes**

Interface: Like a Miniature Abstract Class (2)

```
1 interface ListenableWithHeadset{
2     //no instance variables
3     //no static methods
4     //only static final variables (constants)
5     /* public abstract */ void playto(Headset h);
6 }
7 abstract class Player{
8     String name;
9     public abstract void playto(Headset h);
10 }
11 class Headset{
12     void listen1(ListenableWithHeadset device){
13         device.playto(this);
14     }
15     void listen2(Player device){ device.playto(this); }
16 }
```

Semantic of Interfaces

```
interface ListenableWithHeadset{ }  
abstract class Player implements ListenableWithHeadset{ }
```

- any Player **can** do the tasks in ListenableWithHeadset

```
class MP3Player extends Player{ }
```

- any MP3Player **is a** Player (and hence **can do** the tasks in ListenableWithHeadset)

```
interface ListenableWithBoth extends ListenableWithSpeaker ,  
    ListenableWithHeadset{ }  
class iPhone implements ListenableWithBoth{ }
```

- ListenableWithBoth **contains** the tasks in ListenableWithSpeaker and ListenableWithHeadset **and possibly more**
- any iPhone **can** do all the tasks in ListenableWithBoth

interface Comparable

- Java version of:

```
1 void qsort (void* base, size_t num, size_t size,  
2 int (*compar)(const void*,const void*));
```

- Arrays.sort()

Interface Serializable

```
1 public interface Serializable {  
2 }
```

- empty interface, no method contract?
- a **marker** that allows special handling

```
1 public class ObjectOutputStream extends OutputStream{  
2     // ...  
3     if (obj instanceof Serializable) {  
4         writeOrdinaryObject(obj, desc, unshared);  
5     } else {  
6         if (extendedDebugInfo) {  
7             throw new NotSerializableException(  
8                 cl.getName() + "\n" + debugInfoStack.toString());  
9         } else {  
10            throw new NotSerializableException(cl.getName());  
11        }  
12    }  
13 }
```

Interface Cloneable

```
1 public interface Cloneable {  
2 }
```

- empty interface (note: doesn't define `clone()`), another marker

```
1 /**  
2  * Invoking Object's clone method on an instance that does not  
3  * implement the  
4  * Cloneable interface results in the exception  
5  * CloneNotSupportedException being thrown.  
6  */
```

- Cloneable: can use the Object's `clone()` (bit by bit copy)