Interface

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, May 24-25, 2010

Chapter 11: Recursion

- assume you've learned it in the C class
- have covered it from Java's view in method invocation
- have given a difficult problem in midterm

very important for computer programmers/scientists

Chapter 12: UML and Patterns

- assume you are learning them in the UML class
- some design patterns: learn as you encounter, or in a more advanced class
  - container-iterator: partially covered in **foreach**
  - decoration: partially covered in **Java IO**
  - model-view-controller: partially covered in your **homework/project** (e.g. how to separate the Computer-Casino-Player?)

encourage you to read more!

Chapter 13: Interface and Inner Classes

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

iPod
PC
TV

MP3 player
mobile phone
speaker

radio
casette player
elec. dict.

what devices come with headset jacks?

cdrom
統聯機
PSP

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 }
```

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){ device.playto(this);
         }
13   void listen2(Player device){ device.playto(this); }
14 }
```

Semantic of Interfaces

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

- any Player can do the tasks in ListenableWithHeadset

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

class A extends B implements C, D, E, F, G, S

- 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

Interfaces in `OutputStream`

```
1  public abstract class OutputStream
2    implements Closeable, Flushable { ... }
3  public class Formatter{//java.util.Formatter
4      public void flush() {
5          ensureOpen();
6          if (a instanceof Flushable) {
7              try {
8                  ((Flushable)a).flush();
9              } catch (IOException ioe) {
10                 lastException = ioe;
11             }
12         }
13     }
14 }
```

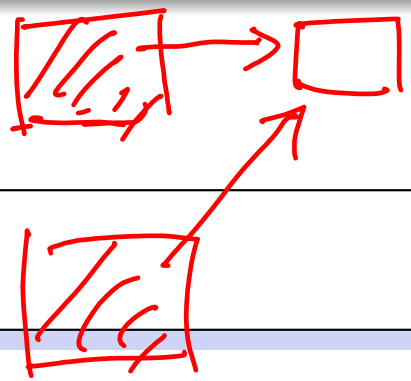
Interface `Serializable`

```
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
      implement the
3  * <code>Cloneable</code> interface results in the exception
4  * <code>CloneNotSupportedException</code> being thrown.
5  */
```

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