Javaのインタフェースについての補足

2006年5月17日

海谷 治彦

Interface

- あるクラスの多面性(ま，別に一面でもいいけど)を明示的に記述したもの.
- あるクラスでinterfaceを実装することで，そのクラスはinterfaceで定義されたメソッドを提供することを保障する.
- interfaceの意味通り，(interfaceを実装した)クラスを使う(他の)クラスのための接点を提供するもの.

例 (というか比喩)

サッカー選手であって,
サッカーする.

大学生であって,
勉強する.
実験する.

家庭教師であって,
教える.

interfaceの記述

```
public interface FootballPlayer{
  public void select(Game g);
}
```

```
public interface Student{
  public void teach(Field f);
}
```

```
public interface PrivateTeacher{
  public void learn();
  public void invite();
  public void emply(Pupil p);
}
```

interfaceの実装

```
class Gakusei implements FootballPlayer, Student, PrivateTeacher{
  public void select(Game g){
      // 実際あるゲームgへの参加選抜をされる際の処理をかく
  }
  public void teach(Field f){
     // 実際にある分野fを教わる場合の処理を書く
  }
  public void employ(Pupil p){
    // 実際にある生徒pに雇われる際の処理を書く
  }
  public void learn(){
    // 実際に雇われている生徒が学ぶ際の処理を書く
  }
}
```

interfaceの効用

- (前述のように)あるクラスの持っている側面に明示的に名をつけて区別することができる.
- クラスを使う側からすれば，クラスではなくインタフェースを指定することで，クラス間の関連を低くすることができる.

  ⇒ あるインタフェースを実装したクラスなら，なんでも使えるような汎用性の高いクラスを書ける.

Interfaceで指定

家庭教師を雇う方にすれば,
その機能を提供するものなら
誰でも良い(はず).

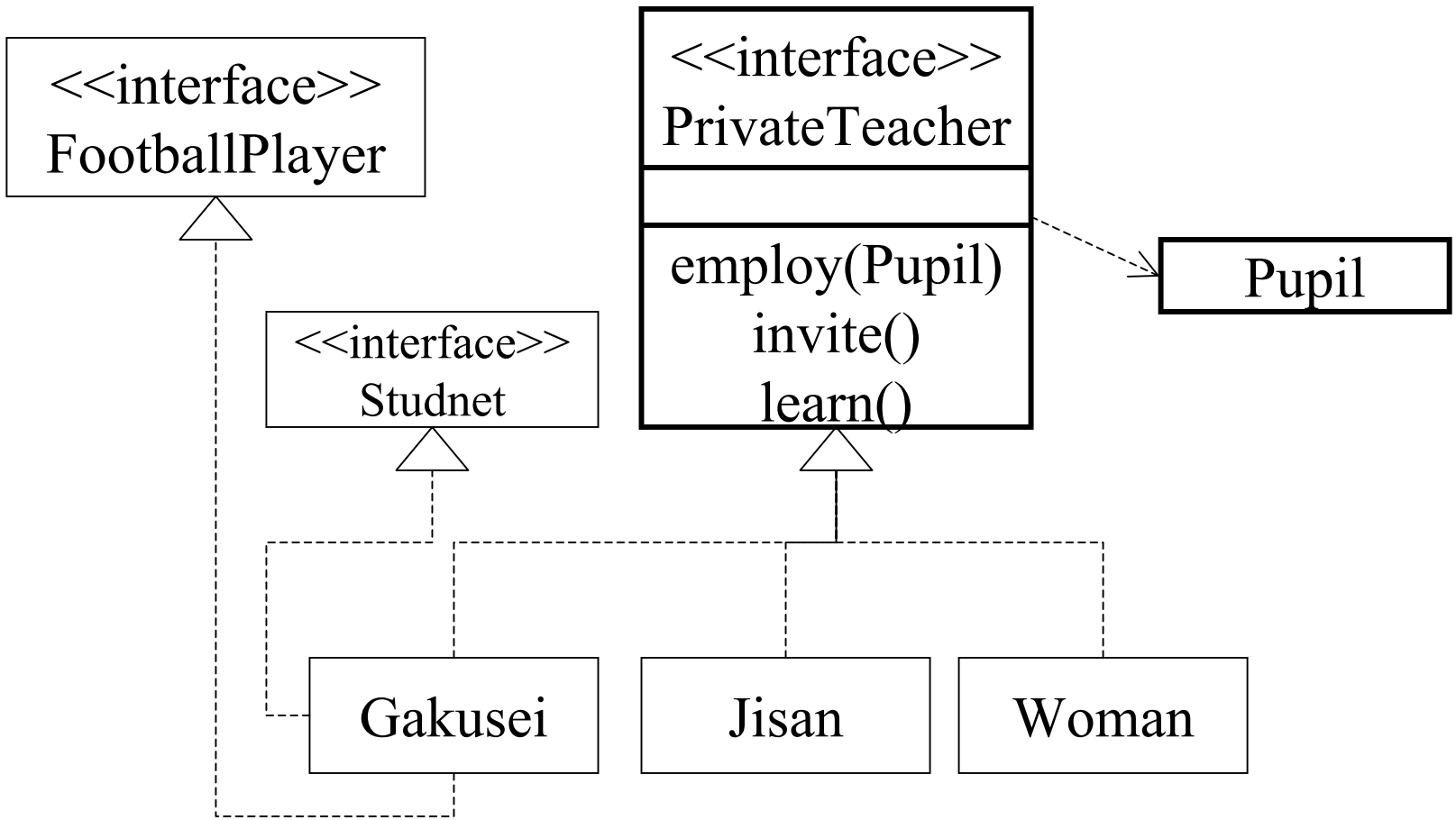
```
class Pupil {
   .... educate(PrivateTeacher pt){
        pt.employ(this);
        pt.invite();
        pt.lean();
   }
}
```

```
class Woman implements
   PrivateTeacher, HouseWife{
      ...
}
```

```
class Jisan implements
     PrivateTeacher, Niwashi{
      ...
}
```

```
class Gakusei implements
    FootballPlayer, Student, PrivateTeacher{ ...
}
```

クラス図で書いてみると



現実的なinterface

- Javaの標準API (Application Programming Interface, 標準的に利用できるクラスライブラリのこと)には，多数のinterfaceと多数のinterfaceを実装したクラスがある.

Interface Serializable

- java.io パッケージ内に定義
- このinterfaceが実装されているクラスのインスタンスは，ファイルにしまったり，ネットワーク上にデータとして転送できたりする.
  - 逆にこれが実装されていないクラスのインスタンスはファイル保存等ができない.
- String, Vector, Integer等，データ指向のクラスでは大抵実装されている.

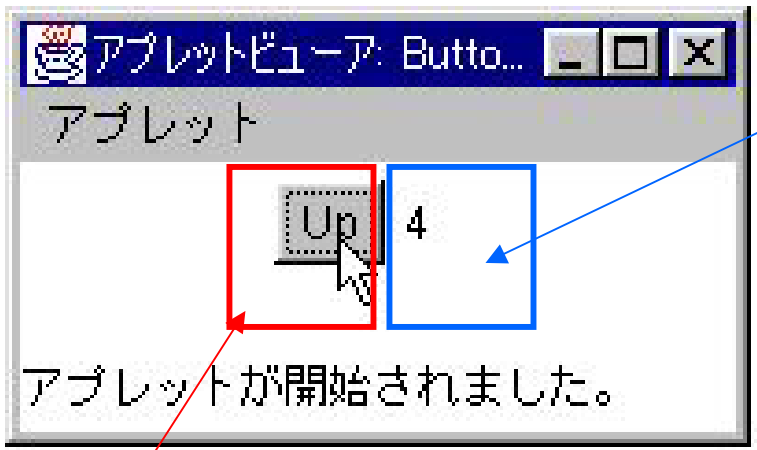
interface Runnable

- run + able すなわち「実行可能」を示す interface.
- スレッド(プログラム内の並行処理の1つ)を実現するためには，このインタフェースに，処理ループを書くのが普通.
- public void run() メソッドの実装を指示.

MouseListener

- GUIにおいてマウスの動作に伴い発生するデータ(event)を拾い，それに反応するためのクラスは大抵，コレを実装している.
- ボタン等は，通常，特定のMouseListenerを実装したクラスが結びついているが，その結びつきを変えることで，簡単にボタンを押した際の振る舞いを変えることができる.

例: リスナを使ったイベント駆動



イベントリスナ:
発生したイベントに対応してある処理をする部品

イベントソース:
イベントを発生する部品

この例では，ボタンを押すとラベルの数値が増える，という単純なもの．

例: ソースコード

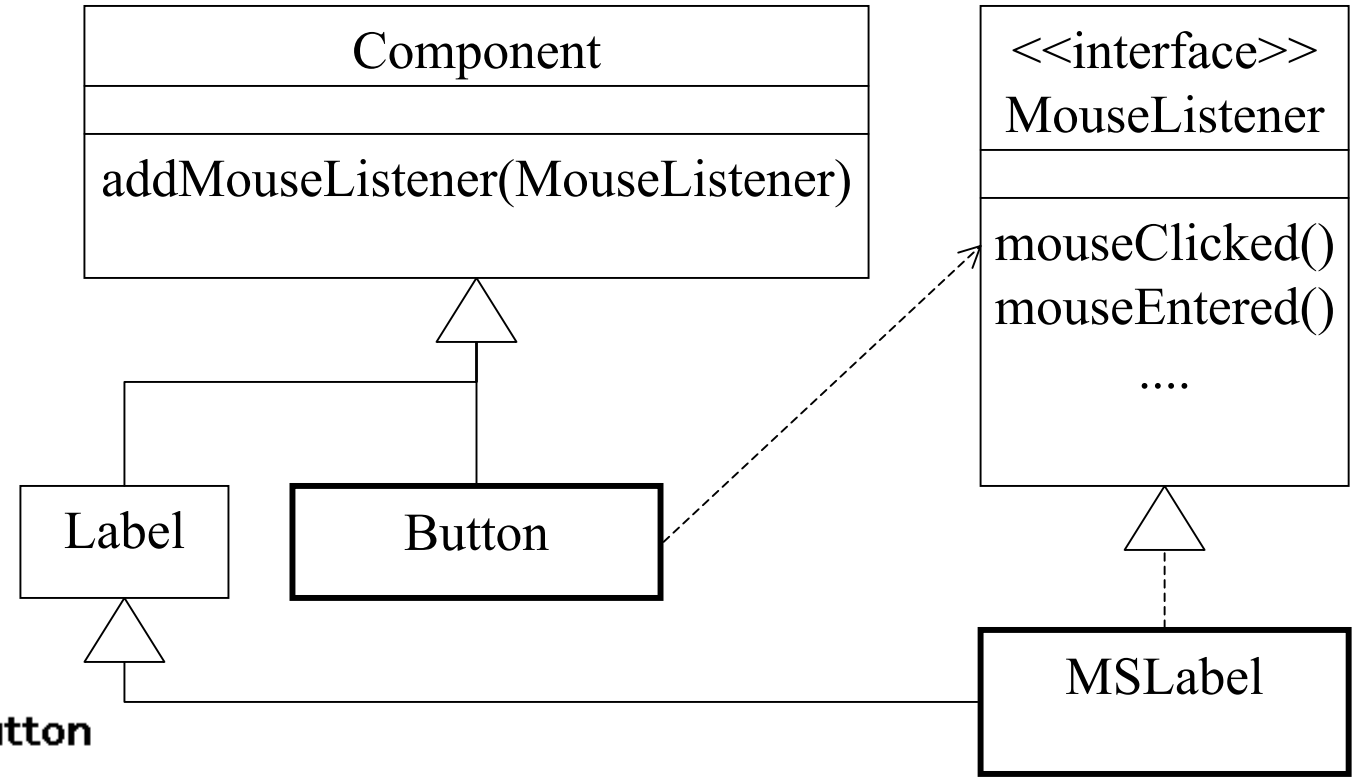
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class ButtonListen extends Applet{

        public void init(){
         Button b=new Button("Up");
         MSLabel ms=new MSLabel(0);
         b.addMouseListener(ms);
         this.add(b);
         this.add(ms);
        }
}
```

```
class MSLabel extends Label implements MouseListener{
private int;
        MSLabel(int initn){
                n=initn;
                setText(n+"");
        }
        public void mouseClicked(MouseEvent e){
                n++;
                setText(n+"");
        }
        public void mouseEntered(MouseEvent e){}
        public void mouseExited(MouseEvent e){}
        public void mousePressed(MouseEvent e){}
        public void mouseReleased(MouseEvent e){}
}
```

ボタンbのイベントを
ラベルmsが聞くように指示

ボタン系のイベントに対応して行う処理を，
ラベル(リスナー)内に実装．

クラス図



java.awt

クラス Button

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Button
```

すべての実装インタフェース:

Accessible, ImageObserver, MenuContainer, Serializable

InterfaceのTIPS

- Classよりむしろのほうが役割という意味に近い.
- 「視点」と考えてもよい.
- インスタンスは実装されているInterfaceで参照できる.
  - Interfaceで参照するとアクセス可能なメソッドは減る可能性がある.
- Classを使うことを想定せず，Interfaceを使うことを想定したクラスのほうが柔軟.

教科書 p.107について

- 「オブジェクトに対する操作方法と，それに対応する振る舞い」を規定？
- 操作法は規定している.
- 振る舞いは規定されているとは言えない.
  - メソッドの名前から直感的な振る舞いはわかる.
  - しかし，そのメソッドが直感的な振る舞い通り実装されているかをインタフェースは保障できない.
- 詳細は反例 Calcable にて