The Observer Pattern, Java Listeners

Prof. Dr. Eric Dubuis
Berner Fachhochschule, Engineering and Information Technology @ Biel

Course "UML and Design Patterns" of module "Software Engineering and Design", version April 2008
The Problem

An object (the *subject*) needs to update various client objects (*observers*):

- The subject needs to inform other interested objects about its state changes
- The subject does not know its observers in advance
- The set of observers may change dynamically
A Solution: The Classical Observer Pattern

The class structure of the classical (GoF) observer pattern (complete):

```
Subject
+attach(o: Observer)
+detach(o: Observer)
#notifyObservers()

ConcreteSubject
-subjectState
+getState(): ...
+setState(...)

ConcreteObserver
-observerState
-myConcreteSubject
+update()

Observer
+update()
```

«interface»
Observer
Collaboration

:Client

aSubject: Subject

setState

observer1: Observer

observer2: Observer
Weakness of Above Solution (from a Java Perspective)

•

•

•
The Java Solution: Typed Observers (Listeners)

Provide an interface and an event class for each type of observer. For example:

```java
// Java version of Observer abstraction:
public interface StateChangedListener { // Java: extends EventListener
    public void stateChanged(StateChangedEvent e);
}

public class StateChangedEvent extends EventObject {
}
```
Clients (observers) can register and unregister themselves at objects (subjects) with the help of the following methods:

```java
// In a class acting as a concrete Subject:
private Set<StateChangedListeners> listeners =
new HashSet<StateChangedListeners>();

public void addStateChangedListener(StateChangedListener l) {
}

public boolean removeStateChangedListener(StateChangedListener l) {
}
```
The subject informs any listener whenever its state changes by sending them a `StateChangedEvent`:

```java
// In a class acting as a Subject:
private void notifyStateChanged() {
    private State myState;
    public void setState(State newState) {
        ...
    }
```
Any object can act as a listener:

```java
public class SomeObserver {
    private StateChangedListener myListener;
    private int stateChangedCounts = 0;
    // Inner class acting as a listener:
    private class MyStateChangeListener implements StateChangedListener {
        public void stateChanged(StateChangedEvent e) {

        }
    }
    public SomeObserver(ConcreteSubject s) {
        this.myListener = new MyStateChangeListener();
    }
}
```
Remarks

- 
- 
- 
- 
-