Design Pattern: Observer (Beobachter)

1 Designprinzip

Bei Änderung des Zustands (bzw. der Daten) des Subjekts werden alle Beobachter benachrichtigt.
Das Subjekt braucht die Struktur der Beobachter nicht zu kennen.
Es gibt zwei Varianten:

- PUSH: Die Daten werden bei der Benachrichtigung der Beobachter gleich mitgeliefert.
- PULL: Die Nachricht vom Subjekt an die Beobachter enthält noch nicht die Daten, sondern nur die Nachricht, dass sich diese geändert haben. Die Beobachter müssen dann selbst die von ihnen benötigten Daten anfordern.

2 Java-Code

2.1 Subject (Observable)

```
public class Subject {
        // Membervariablen fuer Daten
        private int daten = 0;

        // Liste von Beobachtern
        private ArrayList<BeobachterSchnittstelle> beobachter
                                = new ArrayList<BeobachterSchnittstelle>();

        // Beobachter muessen sich registrieren
        public void registrieren(BeobachterSchnittstelle b){
            beobachter.add(b);
        }

        // alle Beobachter erhalten Aenderungsmeldung
        private void broadcast(){
                for (int i = 0; i < beobachter.size(); i++) {
                        beobachter.get(i).update();
                        // bei PUSH: Daten als Argumente uebergeben
                }
        }

        // Methoden zum Lesezugriff auf Daten bei PULL
        public int getDaten() {
            return daten;
        }

        // Broadcast bei Aenderung der Daten
        public void setDaten(int n) {
                daten = n;
                broadcast();
        }
}
```

2.2 Beobachterschnittstelle (Observer)

```
public interface BeobachterSchnittstelle {
        public void update(); // Bei PUSH: Daten als Parameter
}
```

2.3 Konkrete Beobachter

```
public class BeobachterA implements BeobachterSchnittstelle{
        private Subject subject;

        public BeobachterA(Subject subject) {
                this.subject = subject;
                subject.registrieren(this);
        }

        @Override
        public void update() {
            // bei PUSH: Daten werden als Parameter uebergeben
            // bei PULL: Daten muessen extra geholt werden
            int daten = subject.getDaten();
            System.out.println("A:" + daten);
        }
}
```

```
public class BeobachterB implements BeobachterSchnittstelle{
        private Subject subject;

        public BeobachterB(Subject subject) {
                this.subject = subject;
                subject.registrieren(this);
        }

        @Override
        public void update() {
            // bei PUSH: Daten werden als Parameter uebergeben
            // bei PULL: Daten muessen extra geholt werden
            int daten = subject.getDaten();
            System.out.println("B:" + daten);
        }
}
```

2.4 Verwendung des Subjekts

```
Subject subject = new Subject();
BeobachterA beobachterA = new BeobachterA(subject);
BeobachterB beobachterB = new BeobachterB(subject);
subject.setDaten(5); // Ausgabe: "A:5 B:5"
```

Literatur

[Head First Design Patterns] `http://www.oreilly.de/catalog/9780596007126/`

[Wikipedia] `http://de.wikipedia.org/wiki/Observer_(Entwurfsmuster)`