



Drucken, GUI, Design Pattern, ... PDF, Usability, Observer Pattern, MVC

Drucken ist eigentlich ganz einfach ...

- Grafik erzeugen
- PrintJob-Objekt erzeugen
- Grafik an den PrintJob übergeben
- fertig.

Die Realität

- leider nicht so einfach
- Jegliche Toolkits benötigen viel Einarbeitungszeit
- Toolkits z.B.:
 - AWT Printing API (ab Java 1.1)
 - `print` Package (ab Java 1.2)
 - JPS: Java Print Service API (ab Java 1.4)

Toolkits

- Lösungen implementieren Drucken direkt in Java
- teilweise umständliche Handhabung
- können i.d.R. PDF nicht direkt erzeugen (weitere Packages/Toolkits nötig)
- Mögliche Lösung: Toolkit nutzen, das PDF erzeugt und **anschließend** Drucken

PDFs erzeugen

- Wieder viele verschiedene Lösungen (PDFBox, Apache FOP, ...)
- Implementieren den PDF Standard alle nicht vollständig
 - Ist aber kein Problem, da nur selten alles genutzt wird

Beispiel: Apache FOP

- wandelt XML mit XSL-FO nach PDF
- kann als JAR in das Projekt eingebunden werden
- JavaDoc von FOP wird in den IDEs angezeigt
- pingeliger XML-Parser (ein Fehler und wie Welt geht unter...)
- Steile Lernkurve, aber hervorragende Ergebnisse

Links

- JPS: <http://java.sun.com/javase/6/docs/technotes/guides/jps/>
- Apache FOP: <http://xmlgraphics.apache.org/fop/>
- PDFBox: <http://www.pdfbox.org/>
- Tutorial für einige Toolkits:
<http://www.torsten-horn.de/techdocs/java-print.htm>

GUI Editoren

- NetBeans, Visual Editor, JBuilder
- generieren teils seltsamen Code
- man legt sich mittels IDE auf ein GUI-Editor/Toolkit fest
- für dynamische GUIs nicht bzw. kaum zu gebrauchen

Wahl des Toolkits

- Viele verschiedene: SWT, AWT, Swing, Qt, wxWidgets
- man muss immer Kompromisse eingehen:
 - Swing nicht das schnellste, keine nativen Widgets
 - SWT nicht komplett platformunabhängig, teils schlechte Doku
 - AWT veraltet
 - Qt/Jambi ist Wrapper um Wrapper um Wrapper um ...

→ Wahl der Qual

GUI Gestaltungstipps

- Vorhandenes nicht neu Implementieren (z.B. OK Button, Dialoge, ...)
- **Keine** absoluten Breitenangaben
- Fenster sollten sich in ihrer Größe anpassen lassen
- Unsere Kultur liest von links oben nach rechts unten
- Vorhandene GUIs als Referenz ansehen
- Gängige GUI Konventionen **nicht** ignorieren

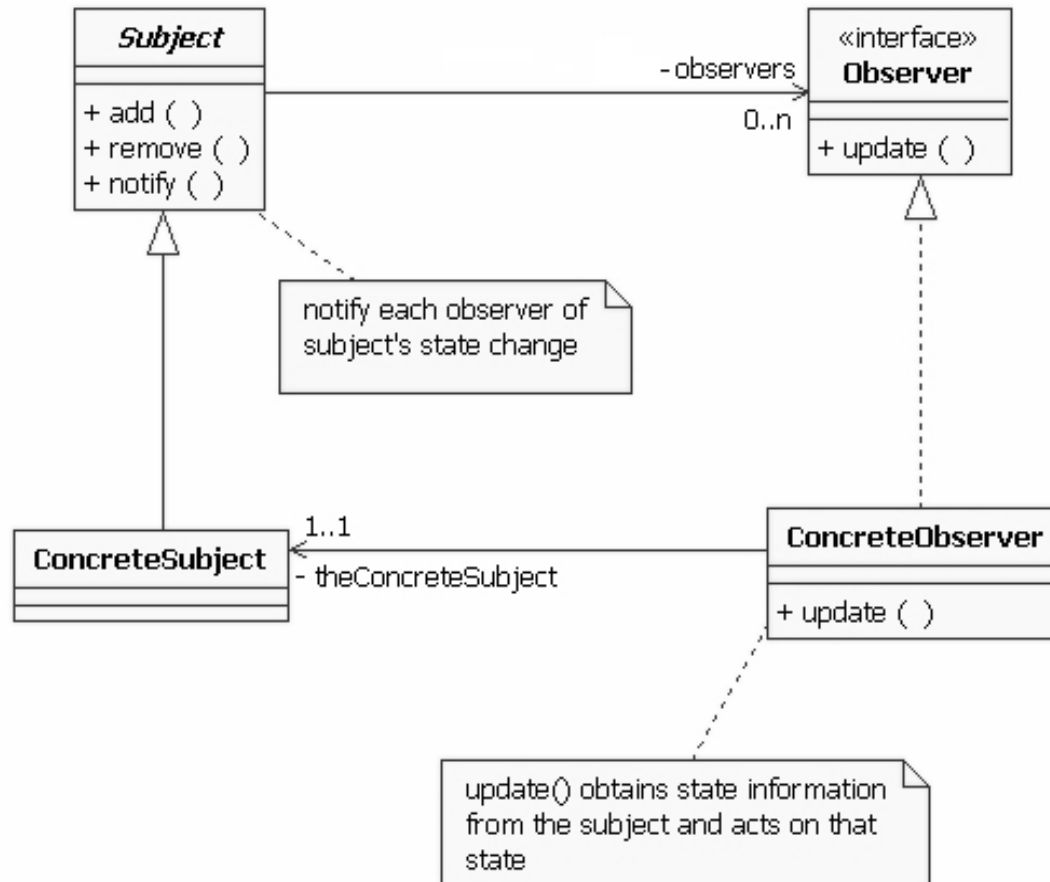
Links

- <http://www.cs.helsinki.fi/u/salaakso/patterns/>
- http://www.isii.com/ui_design.html
- <http://www.eclipse.org/swt/>
- <http://msdn.microsoft.com/en-us/library/ms997506.aspx>
- <http://library.gnome.org/devel/hig-book/stable/>

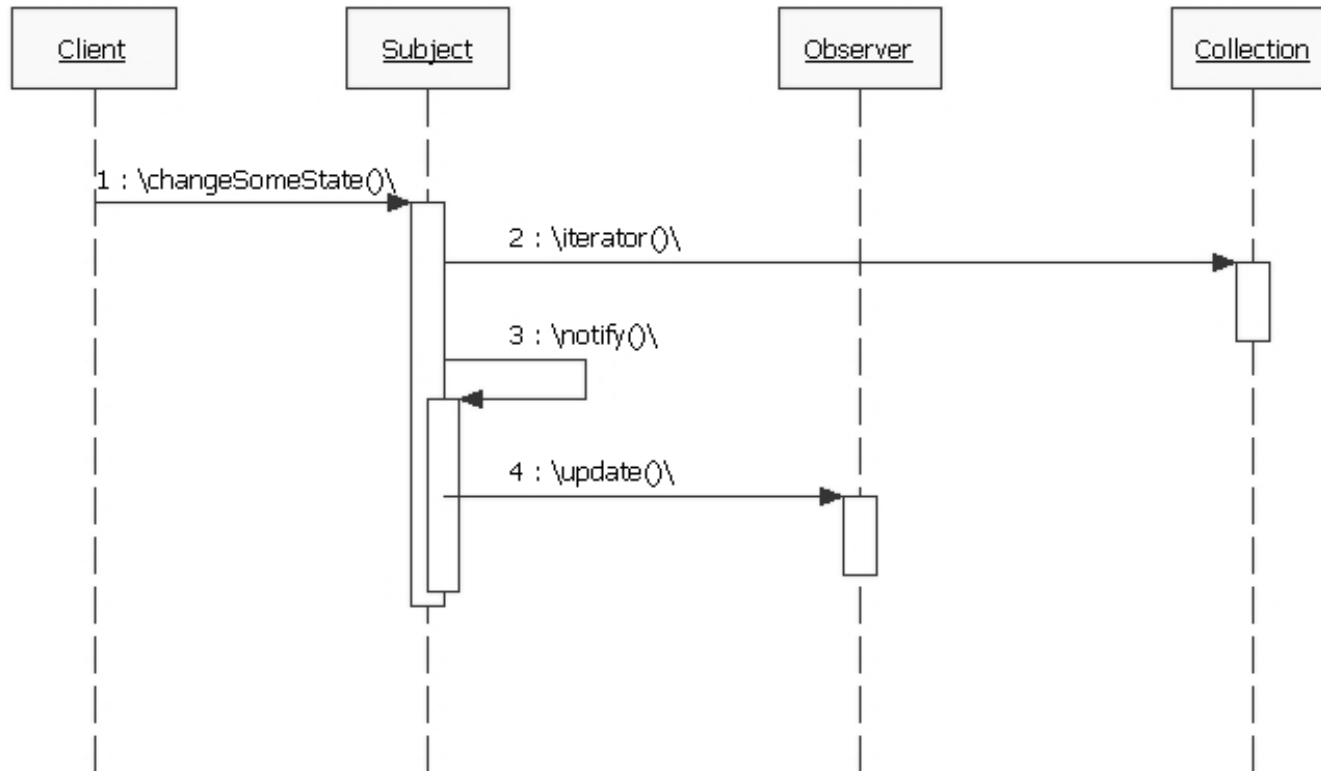
Design Pattern (Entwurfsmuster)

- *bewährte* Lösung für ein bekanntes Problem oder Problemklasse
- Formalisierung der Kommunikation
- Muster für Analyse, Kommunikation, Organisation ...
- Software Engineering, Human Computer Interaction, User Interface Design
- Einteilung in Erzeugende und strukturelle Muster, Verhaltensmuster

Observer Pattern (Beobachtermuster)



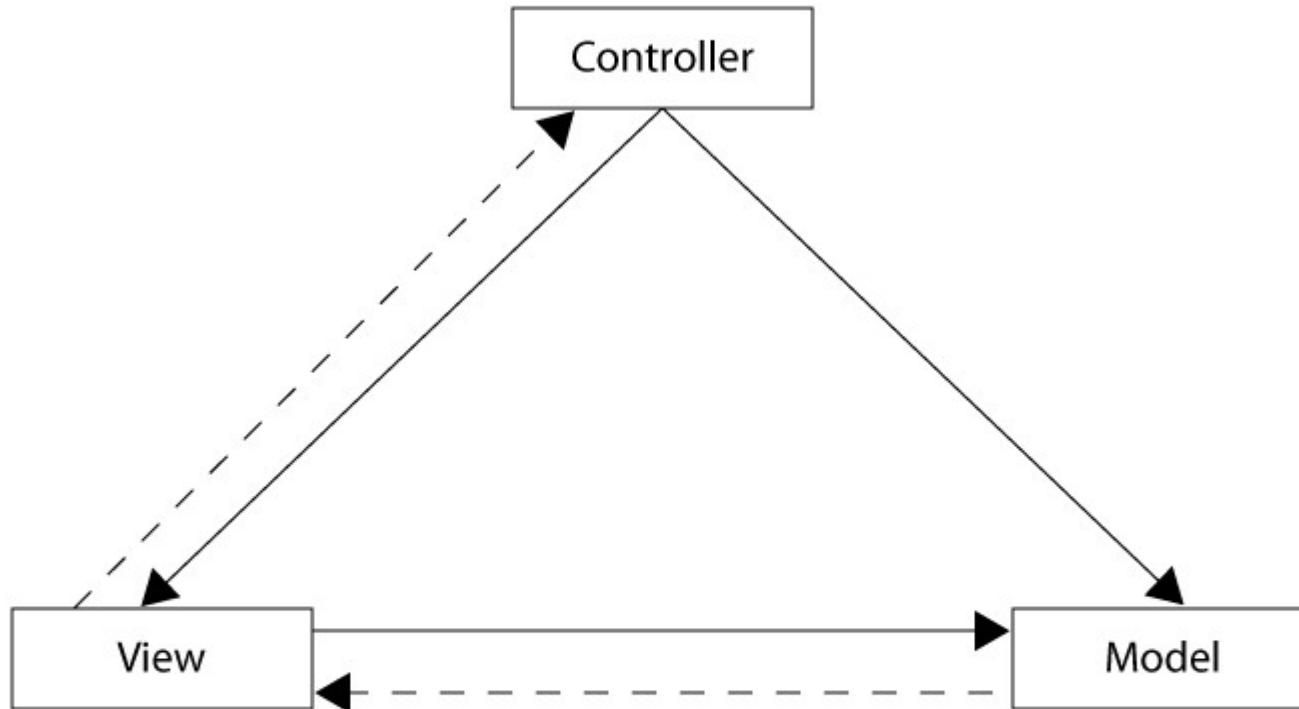
Ablauf



Konsequenzen

- Vorteile
 - automatische Aktualisierung
 - unabhängige Variation von Subjekten und Beobachtern
 - `java.util.Observer`, `java.util.Observable`
- Nachteile
 - Kommunikationskosten
 - Schleifengefahr
 - keine Information über konkrete Änderung

Model View Controller (Modell Präsentation Steuerung)



Komponenten

- Modell
 - hält Daten und Geschäftslogik
 - Bekanntgabe von Änderungen via *Observer Pattern*
- Präsentation
 - Darstellung der Modelldaten
 - Entgegennahme und Weitergabe von Benutzerinteraktionen
 - Struktur via *Composite Pattern*
- Steuerung
 - Begrenzung und Auswertung von Benutzereingaben
 - Verhalten via *Strategy Pattern*

Konsequenzen

- Vorteile
 - Unterstützung mehrerer Präsentationen und Steuerungen durch ein Modell
 - beliebige Änderungen bei identischer Schnittstelle
- Nachteile
 - meist keine klare Trennung der Komponenten möglich
 - Overhead für kleine Anwendungen
- Beispiel: javax.swing