Vorlesungsmodul Softwaretechnik 2
- VorlMod SoftwareTk2-

Matthias Ansorg

14. März 2002 bis 26. Mai 2003

Zusammenfassung

Studentische Mitschrift zur Veranstaltung Softwaretechnik 2 bei Prof. Dr. Franzen (Sommersemester 2002) im Studiengang Informatik an der Fachhochschule Gießen-Friedberg.

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereit: http://homepages.fh-giessen.de/~hg12117/index.html. Wenn sie vollständig ist, kann es auch über die Skriptsammlung der Fachschaft Informatik der FH Gießen-Friedberg http://www.fh-giessen.de/FACHSCHAFT/Informatik/cgi-bin/navi01.cgi?skripte downgeloadet werden.
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieser studentischen Mitschrift liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der verwendeten Quellen zu beachten.
- **Korrekturen:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg, ansis@gmx.de.
- **Format:** Die vorliegende studentische Mitschrift wurde mit dem Programm LyX (graphisches Frontend zu LaTeX) unter Linux erstellt und als pdf-Datei exportiert. Grafiken wurden mit dem Programm xfig unter Linux erstellt und als eps-Datei exportiert.
- **Dozent:** <titel> <name>.
- **Verwendete Quellen:** <quelle> {<quelle>}.
- **Klausur:**

Inhaltsverzeichnis

1 Oberflächenprogrammierung unter Java

1.1 Kommunikation GUI / workflow

Es ist für das Layout sehr zu empfehlen, bestimmte Dialogelemente zu einer Gruppe (Ast der Hierachie) zusammenzufassen. Oberflächenkomponenten sind hierarchisch in einem Baum angeordnet, d.h. Komponenten können einander enthalten. Dabei ist »Frame« der Name der Komponente, die eine Schnittstelle zum umgebenden Fenstersystem hat (ein TopLevel-Fenster; Frame ist eine AWT-Komponente, JFrame eine Swing-Komponente; weitere TopLevel-Fenster z.B. Dialog). Alle anderen Elemente müssen nur eine Schnittstelle zu ihrem Container / übergeordneten Dialogelement haben. Alle TopLevel-Fenster besitzen eine Methode `addWindowListener(WindowListener wl)`, mit der man ein Objekt registrieren kann, das der Schnittstelle WindowListener genügt. Dabei besitzt `interface WindowListener` Aktionen, die angestoßen werden sollen, wenn eine TopLevel-typische Aktion ausgelöst wird (Ereignisse Maximieren, Minimieren, Schließen). Diese Aktionen werden als Namen in interface WindowListener zur Verfühung gestellt und müssen dann ausprogrammiert werden, z.B. `void WindowClosing(WindowEvent we)`. Das Objekt, das diese Methoden besitzt, ist dann ein WindowListener und wird registriert mit `addWindowListener(WindowListener wl)`.

Wird nun ein Ereignis »Fenster schließen« vom Benutzer ausgelöst, so wird das Ereignis von JFrame analysiert und die registrierten WindowListeners dadurch benachrichtigt, dass ihre Methode WindowClosing() aufgerufen wird. So wird eine sehr geringe Kopplung zwischen der Oberflächenkomponente und den Komponenten, die die Arbeit tun, erreicht. Die Oberfläche weiß von der arbeitenden Komponente nur, dass sie das Interface WindowListener bestitzt, nicht einmal von welcher Klasse sie ist. Die Definition der WindowListenerklasse wäre z.B.

```
class Arbeiter1 implements WindowListener {
   windowClosing(WindowEvent we) {
      System.exit(0); //die brutale Variante
   }
// ...
};
```

`implements WindowListener` heißt dabei, dass sich diese Klasse verpflichtet, alle Methoden des `interface WindowListener` zu implementieren. Ein anderes Verhalten wäre z.B.

```
class Arbeiter2 implements WindowListener {
   windowClosing(WindowEvent we) {
      //...
      we.getSource();
      setVisible(false);
      //...
   }
//...
};
```

Dieses Schema der Programmierung wird publisher-subscriber-Schema genannt. Man kann auch mehrere WindowListeners für ein Fenster registrieren. Dabei gibt es keine Garantie einer Reihenfolge, mit der die Methoden der einzelnen WindowListeners aufgerufen werden. Dieses publisher-subscriber-Schema ist die Art, wie in Java generell die Verbindung zwischen GUI und workflow realisiert wird! Es wird damit das Ziel einer geringen Kopplung hervorragend realisiert.

1.2 Layout einer GUI

Container-Komponenten sind solche, die andere Komponenten beinhalten können, z.B. TopLevel-Fenster, Panels usw. Panels sind unsichtbar, wenn sie keine Ränder haben - sie dienen nur der Gruppierung zur Layout-Verwaltung. Wozu nun dienen Container? Container zeichnen sich durch eine Methode `setLayout(LayoutManager lm)` aus: Container müssen den Platz, der ihnen zur Verfügung steht, nach bestimmten Strategien (wechselbare Layouts) an ihre Kinder verteilen. Wichtig: passenden LayoutManager im JBuilder einstellen. Einige LayoutManager:

```
class BorderLayout implements LayoutManager {
   public static final int NORTH=5;
   public static final int CENTER=6;
   //...
}
```

So würde man anfangen, einen eigenen Layout-Manager zu schreiben! Der BorderLayoutManager teilt seinen Platz in 5 Plätze ein: im Norden, Süden, Westen, Osten und in der Mitte. Kinder-Komponenten werden mit einer Methode add zu einem Container hinzugefügt, z.B.

```
jp.setLayoutManager(new BorderLayout());
jp.add(new Button(), BorderLayout.SOUTH);
```

Die Konstanten dienen also nur dazu, den Platz anzugeben, an dem eine neue Komponente eingefüt werden soll. Beim Vergrößern des Fensters bekommen die Ränder nur ihren benötigten Platz, die Mitte den Rest (z.B. für ein EditorFenster). Um nun Elemente innerhalb eines Platzes in einem Container anzuordnen, fügt man dort wiederum Container mit einem bestimmten LayoutManager ein. Ein weiterer Layout-Manager ist GridLayout: hier ist das Fenster in eine einstellbare Anzahl von $m \times n$ gleich großen Feldern eingeteilt, in denen jeweils eine Komponente sein kann. Die primitivste Variante ist FlowLayout, die die einzelnen Komponenten wie Schriftsatz mit Zeilenumbrüchen anordnet.

Man überlässt also die Anordnung der Elemente dem LayoutManager, statt selbst pixelgenau die Positionen festzulegen. Das nämlich würde zu Problemen führen, wenn leicht unterschiedliche Darstellung nötig ist, z.B. auf unterschiedlichen Plattformen und bei unterschiedlichen Bildschirmgrößen. So zum Beispiel kann man das LookAndFeel mit einem Befehl einstellen.

Die Erstellung der Oberfläche sollte im Konstruktor des GUI-Objektes implementiert werden. Man sollte die Oberfläche zuerst einmal vollständig per Hand erstellen, statt den Designer im JBuilder zu verwenden - um zu verstehen, was man macht.

In Java gibt es abstrakte Klassen als Sprachkonzept; mit dem Schlüsselwort abstract erkauft man sich dabei die Erlaubnis, nicht alle deklarierten Methoden auch implementieren zu müssen; diese Aufgabe haben die child-Klassen. (In Java gibt es nur Einfach-Vererbung!). Eine abstract-Klasse hat keinen Konstruktor, d.h. von ihr kann kein Objekt erzeugt werden.

```
abstract class Component {
   //...
   getSize(); //muss nicht implementiert werden
   //...
}
```

Bei Verwendung von Mehrfach-Vererbung entsteht keine Baumstruktur mehr, denn dabei muss jedes Element außer dem Wurzelelement genau einen Vorgänger haben. Nur bei einer Baumstruktur ergibt sich ein Pfad, d.h. eine lineare Liste der Vorgänger.

In Java ist muss Mehrfachvererbung anders modelliert werden, was auch oft besser ist: ein Hausboot ist ein Boot (Vererbung), und hat ein Haus (als Attribut / Komponente). Denn jedes Hausboot ist ein Boot, aber nicht jedes Hausboot ist ein Haus - deshalb passt Mehrfachvererbung hier nicht. Mehrfachvererbung ist meist ein Modellierungsfehler! In Java verwendet man stattdessen z.B. das publisher-subscriber-Schema: ein Objekt übernimmt eine Rolle. Das dabei verwendete »interface«-Konstrukt gibt es in C++ nicht, deshalb geht dort die Vererbung oft durcheinander.

1.3 Registrierung der Objekte zur Ereignisverarbeitung

2 Grobentwurf

Dazu gehört die vollständige Sicht auf das System von außen:

- Benennung der Module, Festlegung ihrer Verantwortlichkeiten
- Zusammenarbeit der Komponenten überlegen (Grobarchitektur)
- Grobarchitektur durch Schnittstellenbeschreibungen festlegen. Eine Schnittstelle ist ein Kontrakt (Vertrag) zwischen Server und Client. Hier sollen die öffentlichen Methodennamen und -aufrufe feststehen, auch die übergebenen Datentypen (ggf. auch Klassen).
- Klassendiagramm. Hier wird statisch festgelegt, was die Klassen voneinander wissen müssen. Man testet, indem man einen Usecase durchspielt. Form der Dokumentation mit JavaDoc-Kommentaren der öffentlichen Methoden möglich, zusätzlich Diagramme zum Zusammenspiel.

Prototyp zum Grobentwurf von EasyFile. Komponenten nach dem MVC-Model:

Model. Von der View-/Control-Komponente werden direkt Methoden aus dem Model, der arbeitenden Komponente, aufgerufen.

View. Aufgabe der GUI ist es, die Parameter zu beschaffen zur Durchführung von Operationen. Und die Operationen aufzurufen. So würde z.B. beim Drücken des Buttons »Datei zum Server übertragen« eine Methode `void getFile(...)` aufgerufen.

Control.

View und Control werden zu UIDelegate zusammengefasst. Zur Realisierung der FTP-Funktionalität kann ipworks (Borland-Komponente aus JBuilder) verwendet werden, auch das net-Paket aus dem JDK - dieses jedoch hat keine derart berauschende Funktionalität. Mit ipworks muss nichts weiter programmiert werden ...

3 Swing

Eine Oberflächenkomponente wie JTree, JButton ist nur ein View auf ein Model, d.i. die darunterliegenden Daten. Das Modell hinter einem Button ist z.B.: aktiviert oder nicht, Schriftzug, Schriftart usw. Beispiel aus [1, S. 75]: Man kann zu einem Modell mehrere Sichten erzeugen, z.B. ein Editor, der in drei getrennten Fenstern

unterschiedliche Ausschnitte desselben Textes darstellt. Wenn in einem Fenster Text geändert wird, soll diese Änderung in den anderen Fenstern auch dargestellt werden. Ein Anzeigefenster will also benachrichtigt werden, wenn eine Änderung im Modell eintritt. Die Implementierung:

```
class SharedModel {
  public static void main(String[]args) {
    JFrame f=new JFrame("Beispiel");
    //JFrame ist ein UIDelegate!
    Container content=f.getContentPane();
    //neues Modell durch parametelosen Konstruktor erzeugen:
    JTextArea ta1= new JTextArea();
    Document doc= ta1.getDocument();
    //Modell doc uebernehmen durch Konstruktorparameter:
    JTextArea ta2= new JTextArea(doc);
    JTextArea ta3= new JTextArea(doc);
    //Komponenten mit Schiebebalken hinzufuegen:
    content.add(new JScrollPane(ta1));
    content.add(new JScrollPane(ta2));
    content.add(new JScrollPane(ta3));
    //Groesse des ganzen Frames setzen:
    f.setSize(300,400);
    //die folgende Methode gibt es fuer jede Komponente:
    f.setVisible(true);
  }
};
```

Alle beinhalteten Komponenten sollten in die ContentPane eines Frame eingefügt werden; es gibt noch eine GlassPane für Transparenz usw. Zu jeder Swing-Komponente gibt es eine Methode, die das Modell der Komponente liefert, bei JTextArea z.B. `getDocument()`. Dieses Modell übergibt man nun, wenn ein anderes View dasselbe Modell benutzen soll. Hier wurde das Beobachter-Entwurfsmuster verwendet. Auch im FTP-Client wird dieses verwendet: registrierte Komponenten der Oberfläche (publicher-subscriber) werden benachrichtigt, wenn sich etwas ändert.

Im FTP-Client sollte man zur Darstellung der Dateien JTree (oder JList) verwenden. Zu den Modellen von JTree gibt es Schnittstellenbeschreibungen. Es gibt abstrakte Klassen für solche Modelle, von denen man sein eigenes Modell ableiten kann. Man kann auch ein vollständiges eigenes Modell schreiben, muss dann aber die vollständige Schnittstelle des Modells implementieren.

Zu den geforderten Funktionalitäten der arbeitenden Komponenten kommt man durch Usecase-Diagramme.

4 Feinentwurf

Das ist die Festlegung der Interna der Komponenten, d.h. private und protected-Methoden. Hier wird festgelegt, wie eine Klasse intern arbeitet. Die Form ist eine JavaDoc-Kommentierung der Klassen. Der Feinentwurf geht Komponente für Komponente vor, ohne auf das Zusammenspiel der Komponenten Rücksicht zu nehmen.

5 Serialisierung von Objekten

Es gibt eine Methode ObjectOutputStream:writeObject(Object o), mit der beliebige Objekte (die die Schnittstelle serializable implementieren!) automatisch serialisiert

in einen Output-Stream geschrieben werden können: oos writeObject(favListe). Das Objekt kann mit ObjectInputStream:readObject() wieder eingelesen werden. Verwendung mit Typnachfrage bei einem Input-Stream:

obj=ois.readObject(); if(obj instance of FavListe)

Es gibt in der Klasse Object eine Methode getClass(), die ein Objekt vom Typ Class zurückliefert; also: if(obj.getClass == FavListe.class). Typangabe mit Casting, Vorsicht vor ClassCastException:

FavListe fl;

fl = (FavListe) obj;

Durch diesen Cast findet die Typprüfung zur Laufzeit statt, nicht zur Übersetzungszeit.

Die Schnittstelle serialiable ist nur eine Markierungsschnittstelle (marker interface):

interface Serializable { }

Ein anderes MarkerInterface ist Clonable.

Im Paket Java.util gibt es Schnittstellen und abstrakte Klassen ähnlich den Listen usw. in der STL. Man kann diese Objekte als Listen usw. im eigenen Programm verwenden.

Together 5.0

bis 2002-06-30

Literatur

[1] John Zukowski's Definite Guide to Swing for Java 2.

[2] Skript zur Vorlesung »Softwaretechnik 1« bei Prof. Franzen. http://hera.mni.fh-giessen.de/~hg7132/swt/swt-1.html

[3] »javadoc - The Java API Documentation Generator«. Handbuchseite zu Javadoc. http://java.sun.com/products/jdk/1.2/docs/tooldocs/win32/javadoc.html

[4] »Javadoc Tips«. FAQ zur Verwendung von Jvadoc. http://java.sun.com/j2se/javadoc/faq/

[5] »RFC1635; References RFC2585«

[6] »RFC2228; FTP Security Extensions«

[7] »RFC2389; Feature negotiation mechanism for the File Transfer Protocol«

[8] »RFC2640; Internationalization of the File Transfer Protocol«

[9] »RFC959; FILE TRANSFER PROTOCOL (FTP)«.