Technische Universität München
Fakultät für Informatik
A. Brandl, M. Göbel

WS 1998/99
Lösungsvorschläge zu Blatt 4
29. Oktober 1998

Übungen zum Programmieren in Java

Aufgabe 1 Arbeiten mit der Java-API-Dokumentation

`java.awt.Frame` erbt die Methode `add(Component)` von der Klasse `java.awt.Container`.

Aufgabe 2 Einfaches GUI-Programm

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

/**
 * Minimal ordnet gitterfoermig bei jedem Buttonclick einen
 * neuen Button an, bis maximal 9 Buttons angezeigt werden.
 */

public class Minimal extends Frame
  implements ActionListener {

  int AnzButtons = 0;

  public Minimal() {
    setLayout(new GridLayout(2,6));
    actionPerformed(null);
    pack();
    setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {
    if (AnzButtons<9) {
       Button nbutton =  new Button("der " + AnzButtons++ + ".te Button");
       nbutton.addActionListener(this);
       add(nbutton);
       pack();
    }
  }

  public static void main(String args[]) {
    Minimal m = new Minimal();
  }
}
```

Gespeichert in der Rechnerhalle unter:

`/usr/proj/javakurs/Tag4/loesungen/Minimal.java`

Aufgabe 3 Einlesen aus Datei

```
import java.io.*;

/**
 * Dateilesen  gibt den Inhalt der uebergebenen Datei aus.
 */

public class Dateilesen {

  public static void main(String[] args) {
    try {
      FileReader fr = new FileReader(args[0]);
      int ieingabe = 1;
      String se="";
      while ((ieingabe = fr.read())!=-1) {
        se+=(char)ieingabe;
      }
      System.out.println("Der eingelesene String : " + se);
    } catch (IOException e) {
      System.out.println("Fehler beim Lesen der Datei " + args[0]);
    }  catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Falsche Anzahl Parameter");
    }
  }
}
```

Gespeichert in der Rechnerhalle unter:

`/usr/proj/javakurs/Tag4/loesungen/Dateilesen.java`

Aufgabe 4 Arbeiten mit `java.awt.List`

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

/**
 * MitarbeiterListfenster bietet dem Benutzer die Moeglichkeit,
 * eine Liste zu verwalten, d.h. Eintraege einzufuegen und zu
 * loeschen.
 */

public class MitarbeiterListfenster extends Frame
  implements ActionListener
```

```
//**********************zuergaenzen**************************
          ,ItemListener {
//***********************Ende zuergaenzen*********************

Button buttonende;
List listmitarbeiter;
//**********************zuergaenzen**************************
Button buttonneu;
TextField textfieldneu;
Button buttonloeschen;
//***********************Ende zuergaenzen*********************

public MitarbeiterListfenster() {
  setLayout(new BorderLayout());

//****************zuergaenzen*********************************
  //Panel zum Anordnen von List und TextField
  Panel panelwest = new Panel(new BorderLayout());
  listmitarbeiter = new List(3,false);
  listmitarbeiter.add("Huber");
  listmitarbeiter.add("Schmidt");
  listmitarbeiter.add("Obermeier");
  listmitarbeiter.add("Adam");
  listmitarbeiter.addItemListener(this);
  panelwest.add(listmitarbeiter,BorderLayout.NORTH);

  textfieldneu = new TextField("neuer Mitarbeiter");
  panelwest.add(textfieldneu,BorderLayout.SOUTH);

  //Panel zum Anordnen der Buttons
  Panel paneleast = new Panel(new BorderLayout());
  buttonneu = new Button("Neu");
  buttonneu.addActionListener(this);
  paneleast.add(buttonneu,BorderLayout.NORTH);

  buttonloeschen = new Button("L<F6>schen");
  buttonloeschen.addActionListener(this);
  //das L"oschen ist nur m"oglich, wenn
  //ein Listeintrag selektiert ist, das ist am Anfang
  //nicht der Fall
  buttonloeschen.setEnabled(false);
  paneleast.add(buttonloeschen,BorderLayout.CENTER);

  buttonende = new Button("Beenden");
  buttonende.addActionListener(this);
  paneleast.add(buttonende,BorderLayout.SOUTH);

  //das Panel mit List und TextField soll im Frame links angeordnet werden
  add(panelwest,BorderLayout.WEST);
  //das Panel mit den Buttons soll im Frame rechts angeordnet werden
  add(paneleast,BorderLayout.EAST);
```

```
//***********************Ende zuergaenzen**********************
  pack();
  show();
  setTitle("MitarbeiterFenster");
}
/**
 * itemStateChanged() wird aufgerufen, wenn der Benutzer
 * einen Listeintrag selektiert oder deselektiert, die
 * Deselektion fkt. nicht auf allen Plattformen;
 * der Button buttonloeschen wird abh"angig davon,
 * ob eine Selektion oder eine Deselektion stattfand, gesetzt
 */
public void itemStateChanged(ItemEvent e) {
  buttonloeschen.setEnabled(e.getStateChange()==ItemEvent.SELECTED);
}

public void actionPerformed(ActionEvent e) {
//***********************zuergaenzen****************************
 if (e.getSource()==buttonende)
   System.exit(0);
 else if (e.getSource()==buttonneu)
   listmitarbeiter.add(textfieldneu.getText());
 else if (e.getSource()==buttonloeschen) {
   listmitarbeiter.remove(listmitarbeiter.getSelectedIndex());
   buttonloeschen.setEnabled(false);
 }
 //**********************Ende zuergaenzen**********************
 }

public static void main(String args[]) {
  MitarbeiterListfenster abtfenster = new MitarbeiterListfenster();
}
}
```

Gespeichert in der Rechnerhalle unter:

`/usr/proj/javakurs/Tag4/loesungen/MitarbeiterListFenster.java`

Aufgabe 5 Spezifikation des GUI-Programms für Abteilungen

Ziel dieser Aufgabe ist es, den Kursteilnehmern die Anwendung einfacher, informeller Spezifikationenstechniken (hier eher im Sinne von Vorüberlegungen) zu zeigen.

Welche Daten werden auf dem Bildschirm angezeigt ?

Es werden die Membervariablen der Klasse `Abteilung` am Bildschirm dargestellt:

- `String name`

- `Mitarbeiter chef`
- `Table personal`

Welche Interaktionsobjekte werden für die Darstellung der Membervariablen verwendet ?

- `String name:`

  `String name` könnte durch ein Objekt vom Typ `java.awt.Label` dargestellt werden. Da `name` jedoch ständig vom Benutzer veränderbar sein soll, bietet sich die Verwendedung eines Objekts vom Typ `java.awt.TextField` an.

- `Mitarbeiter chef:`

  Zur vollständigen Darstellung eines Objekts vom Typ `Mitarbeiter` wäre die Darstellung sämtlicher Membervariablen von `Mitarbeiter` notwendig. Darauf wird hier verzichtet. Es soll hier lediglich der Name des Chefs dargestellt werden. Die weiteren Informationen zum Chef (z.B. Geschlecht) werden in der Personalliste (siehe unten) dargestellt. Weil der Name nicht unbedingt eindeutig ist, wäre es allerdings besser, wenn auch nicht so schön, stattdessen die Personalnummer des Chefs auszugeben.

- `Table personal:`

  Durch `personal` wird eine Menge von Objekten vom Typ `Mitarbeiter` verwaltet. Für jeden einzelnen Mitarbeiter sollen die Werte seiner Membervariablen angezeigt werden. Es gibt gibt dazu eine Reihe von Möglichkeiten. Hier betrachten wir die Verwendung von `java.awt.List` zur Darstellung von `personal`. Mit `java.awt.List` kann dem Benutzer eine Liste von einzeilige Zeichenreihen (`String`-Objekte) angezeigt werden. Da `personal` jedoch aus einer Liste von Objekten vom Typ `Mitarbeiter` besteht, ist es notwendig, `Mitarbeiter`-Objekte in einzeilige Zeichenreihen umzuwandeln. Die Namen der Membervariablen werden dabei in der gleichen Reihenfolge oberhalb der Liste dargestellt, wie ihre Werte in den einzelnen Listeinträgen. Dabei ist es notwendig, eine Schriftart einzustellen, bei der alle Zeichen identische Breite und Abstand voneinander haben (äquidistante Schriftart, z.B. "Courier"). Bei dieser Vorgehensweise gibt es jedoch Einschränkungen bezüglich der darstellbaren Länge von Zeichenreihen, weil die Werte der einzelnen Membervariablen jeweils untereinander stehen müssen (aus optischen Gründen). Wir nehmen hier jedoch die Einschränkung in Kauf.

Wie werden die Daten eingegeben und geändert ?

Es werden wiederum die einzelnen Membervariablen von `Abteilung` betrachtet:

- `String name:`

  Die Eingabe erfolgt direkt durch das Textfeld, in dem `name` auch angezeigt wird.

- `Mitarbeiter chef:`

  Der Benutzer kann einen Mitarbeiter aus der Liste selektieren und anschließend den Button "als ChefIn" betätigen. Somit ist auch sichergestellt, daß ChefIn auch in `personal` enthalten ist.

- `Table personal:`

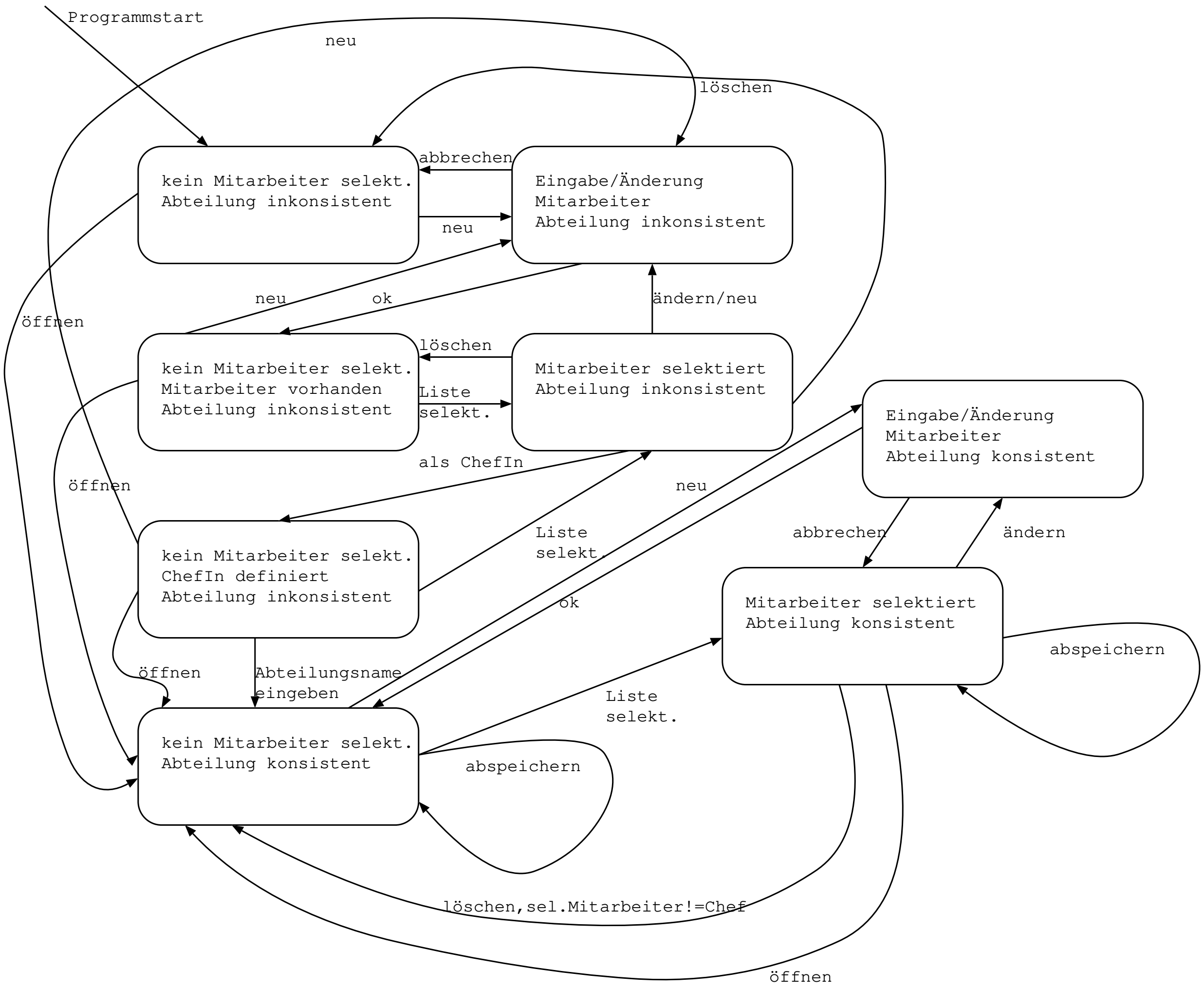
  `java.awt.List` ermöglicht nicht die direkte Eingabe von Zeichenreihen. Es ist also notwendig, dafür eigene Interaktionsobjekte vorzusehen. Diese Interaktionsobjekte können im selben Fenster wie die Liste angeordnet werden, sie können auch in einem eigenen Fenster gekapselt werden. Für `name` und `vorname` bieten sich Objekte vom Typ `java.awt.TextField` an. Die Eingabe von `istWeiblich` kann bequem über Checkboxen (`java.awt.Checkbox`) erfolgen. Zu Überlegen ist die Eingabe bzw. Änderung der Personalnummer eines Mitarbeiters. Hier bietet es sich an, die Personalnummer automatisch zu erzeugen und nicht vom Benutzer eingeben zu lassen, auch damit sie garantiert eindeutig ist. Konsequenterweise kann der Benutzer die Personalnummer auch nicht verändern.

Wie reagiert das GUI-Programm auf Benutzereingaben ?

Im Skript wird zu Begin von Kapitel 8 eine Teilantwort auf diese Frage geliefert und damit auch eine prinzipielle Möglichkeit gezeigt, das Verhalten von GUI-Programmen zu beschreiben. Man spricht dabei auch von einem Zustandsübergangsdiagramm (vgl. mit endlichen Automaten).

Zustandsübergangsdiagramme können bei der Implementierung von GUI-Programmen eine wertvolle Hilfe darstellen.

Die im Skript verwendete Form ist jedoch zu platzaufwendig, um damit die gesamte Funktionsweise des GUI-Programms zu beschreiben. Besser ist es, mit abstrakten Zuständen (Knoten) zu arbeiten. D.h. die Knoten werden mit Bezeichnern beschriftet, die den jeweiligen Zustand beschreiben. Die Zustandsübergänge (Kanten) können beispielsweise mit den Namen der Interaktionsobjekte beschriftet werden, über die sie der Benutzer auslösen kann. Sie können aber auch mit abstrakten Bezeichnern für komplexere Benutzerinteraktionen beschriftet werden, die nicht näher im Zustandsübergangsdiagramm betrachtet werden (siehe unten z.B. "oeffnen").




```
Anmerkungen:
- "Abteilung konsistent" bedeutet, daß aus dem Inhalt der Interaktionsobjekte
  (Textfeld,Liste) ein Objekt vom Typ Abteilung instantiiert werden kann
  (entspr. "Abteilung konsistent"). Dabei wird angenommen, dass ein
  Abteilungsname ungleich dem leeren String sein muss.
- Bei selektierter ChefIn ist kein Löschen möglich
- Mit "öffnen" wird das erfolgreiche Laden eines
  persistent abgespeicherten Objekts vom Typ Abteilung bezeichnet
  (entspr. abspeichern)
- beenden ist von allen Zuständen aus möglich und wird daher
  aus Übersichtsgründen weggelassen
```

Aus welchen Klassen besteht das GUI-Programm ?

- Eine Klasse zur Darstellung einer Abteilung in einem Frame (`Abteilungsfenster`),
- Eine Klasse zur Bearbeitung eines Mitarbeiters in einem Frame (`Mitarbeiterfenster`).

Wie werden die Daten persistent abgespeichert ?

Aus dem Skript sind die Konzepte zum Abspeichern und Öffnen von persistent abgespeicherten Objekten bekannt. Ebenso wurde im Skript die Klasse `java.awt.FileDialog` erwähnt, mit der der Benutzer eine Datei auswählen kann.

Aktionen mit Dateien werden bei GUI-Programmen häufig über Menüs ausgelöst. Im Programm werden daher im Konstruktor von `Abteilungsfenster` geeignete Menüs erzeugt. D.h. es gibt einen Menüeintrag "oeffnen", der zur Ausführung einer Methode

`Abteilungsfenster.oeffnen()`

führt, in der zuerst ein `FileDialog` gestartet wird und anschließend aus der gewählten Datei ein Objekt vom Typ `Abteilung` gelesen wird. Ausserdem gibt es einen Menüeintrag "speichern", der zur Ausführung einer Methode

`Abteilungsfenster.speichern()`

führt, in der ebenfalls ein `FileDialog` gestartet wird und anschließend in die gewählte Datei ein Objekt vom Typ `Abteilung` geschrieben wird.

Aufgabe 6 Implementierung der GUI für Abteilungen

Gespeichert in der Rechnerhalle unter:

`/usr/proj/javakurs/Tag4/loesungen/Abteilungsfenster.java`