University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – Eprog HS13

Übung 10

1 Aufgabe: Loosely Coupled Interaction

1.1 Lernziele

1. Sie können die Interaktion zwischen verschiedenen Objekten mit Hilfe von Interfaces lose gekoppelt gestalten.

1.2 Observer

Gegeben sei die folgende Situation:

An einer Universität existieren Dozenten, sowie Studenten und gelegentliche Gasthörer. Wenn ein Student oder Gasthörer beabsichtigt, eine Vorlesung zu besuchen, dann muss sich dieser beim verantwortlichen Dozenten anmelden. Der Dozent muss dann den Studenten, bzw. Gasthörer irgendwie registrieren, damit sichergestellt werden kann, dass administrative und sonstige vorlesungsrelevante Informationen an alle Hörer verteilt werden. Sollte ein Student vorzeitig aus der Veranstaltung ausscheiden, so meldet er sich erneut beim Verantwortlichen, sodass dieser ihn austragen kann.

1. Implementieren Sie eine Klasse `Lecturer` und definieren Sie die öffentliche Schnittstelle der Hörer anhand eines Interfaces `ILectureListener`.
   - `Lecturer`
     - Konstruktor, welcher den Namen des Dozenten erwartet.
     - `registerStudent`, veranlasst den Dozenten, einen neuen Hörer zu vermerken und in einem typsicheren Vektor abzulegen.
     - `unregisterStudent`, veranlasst den Dozenten, einen Hörer aus der Veranstaltung auszutragen und ihn aus dem Vektor zu löschen.
     - `distributeInformation`, verteilt an alle im Vektor vermerkten Hörer Informationen, die in Textform vorliegen, sprich ruft deren `receiveInformation()`-Methode auf. Der Name des Dozenten soll ebenfalls an diese Methode übergeben werden.
   - `ILectureListener`
     - `receiveInformation`, veranlasst den Hörer, die Informationen, sowie den Namen des zugehörigen Dozenten entgegenzunehmen und nach seinem Gutdünken weiter zu verwerten.

2. Schreiben Sie nun die Klassen `Student` und `GuestListener`, welche ein Attribut Name besitzen und das `ILectureListener`-Interface implementieren. Die Objekte der beiden Klassen sollen die erhaltenen Informationen, sowie den Namen des informierenden Dozenten auf unterschiedliche Weise auf die Konsole ausgeben (z.B. `Student`: „Lecturer XYZ told me, Michael, ...“, `GuestListener`: „I, Petra, was told by lecturer XYZ that ...“).
3. Erzeugen Sie nun in der `main()`-Methode einer Testklasse zwei unterschiedliche Dozenten, sowie je zwei Studenten und Gasthörer, die sich sowohl beim ersten, wie auch beim zweiten Dozenten registrieren. Nun sollen Sie dank der `distributeInformation()`-Methode der Dozenten die Hörer über wichtige Neuigkeiten informieren. Anschliessend sollen sich je ein Student und ein Gasthörer bei einem der Dozenten abmelden, bevor Sie ein letztes Mal Neuigkeiten an die verbliebenen Hörer versenden.

2 Aufgabe: Rekursion

2.1 Lernziele

1. Rekursive Programmierung üben

2.2 Aufgabenstellung

Schreiben Sie je eine rekursive Funktion zur Lösung der folgenden Aufgaben.

- Invertieren eines Strings. Beispiel: invert("Test") -> tseT
- Berechnung der Fakultät. Beispiel: fac(4) -> 1 x 2 x 3 x 4 -> 24
- Grösster gemeinsamer Teiler. Beispiel: gcd(4,8) -> 4

3 Aufgabe: Sorting

3.1 Lernziele

1. Sie können einen Sortieralgorithmus implementieren.

3.2 Aufgabenstellung

Lesen Sie erst die folgende Problembeschreibung, den weiter unten beschriebenen Algorithmus und den gegebenen Programmcode durch, bevor Sie die Teilaufgaben lösen:

Eine Ameise findet sich auf einem ihrer Streifzüge unvermittelt im Schlaraffenland wieder, als sie auf einen Picknickplatz krabbelt, welcher mit Myriaden essbaren Krümeln unterschiedlichster Art übersät ist. Ob Kalorienbomben (viel Gewicht und viele Kalorien) oder kleine leichtbekömmliche Häppchen (wenig Gewicht und wenige Kalorien), für jedes Gourmet-Insekt ist ein Leckerbissen mit dabei.
Der Ameise treten vor lauter Enttäuschung Tränen in die Fühler, als sie versucht möglichst viele dieser Köstlichkeiten auf ihren Rücken zu laden und dabei resignierend feststellen muss, dass sie nur das 300-fache ihres eigenen Körpergewichts tragen kann.
Helfen Sie der Ameise möglichst viel nahrhafte Beute nach Hause zu tragen, in dem Sie im Rahmen der nächsten Teilaufgaben Schritt für Schritt einen Algorithmus in Java umsetzen, der in etwa wie nachfolgend beschrieben funktioniert (Beachten Sie, dass die Ameise mehrere Krümel auf einmal tragen kann und grosse sperrige Krümel daher nicht zwingend die beste Wahl für die Ameise darstellen. Wichtig ist, dass sie möglichst viele Kalorien nach Hause in den Ameisenhügel schaffen kann):

1. Sortiere alle Krümel auf dem Krümelhaufen absteigend nach dem Verhältnis zwischen Kalorien und Gewicht.
2. Packe den ersten/nächsten Krümel vom sortierten Krümelhaufen auf den Rücken der Ameise.
3. Wiederhole den zweiten Schritt solange, bis durch das Hinzufügen des nächsten Krümels das Gesamtgewicht der insgesamt aufgeladenen Krümel die Hebekraft der Ameise übersteigen würde oder der letzte Krümel auf dem Haufen abgearbeitet wurde. Trifft der erste Fall ein (der nächste Krümel würde die Ameise überfordern), gehe zum vierten Schritt. Trifft hingegen der zweite Fall ein (alle Krümel wurden gewogen und je nachdem auf den Rücken der vor Anstrengung ächzenden Ameise verladen oder als zu schwer für sie befunden), gehe zu Schritt 5.
4. Überspringe den nächsten Krümel auf dem Haufen und gehe zu Schritt 2 zurück.
5. Die Ameise ist vollgepackt mit Krümeln und zum Loskrabbeln bereit.

Zusätzlich zur oben stehenden Beschreibung sei der folgende Code gegeben:

```
1 public class Ameise {
2    private static Kruemel[] kruemelHaufen;
3    private java.util.List<Kruemel> aufgeladeneKruemel;
4    private float hebeKraft;
5
6    public Ameise() {
7      aufgeladeneKruemel = new java.util.ArrayList<Kruemel>();
8      hebeKraft = 30;
9      Ameise.verstreueKruemel();
```

```
10      }
11      public static void verstreueKruemel() {
12          java.util.Random r = new java.util.Random();
13          kruemelHaufen = new Kruemel[10];
14          for (int i = 0; i < kruemelHaufen.length; i++) {
15              float gewicht = r.nextInt(10) + 1;
16              float kalorien = r.nextInt(10) + 1;
17              kruemelHaufen[i] = new Kruemel(gewicht, kalorien);
18          }
19      }
20  }
21  class Kruemel {
22      private float gewicht;
23      private float kalorien;
24
25      public Kruemel(float gewicht, float kalorien) {
26          this.gewicht = gewicht;
27          this.kalorien = kalorien;
28      }
29      public float gewicht() {
30          return gewicht;
31      }
32      public float kalorien() {
33          return kalorien;
34      }
35      public float verhaeltnis() {
36          return kalorien / gewicht;
37      }
38  }
```

1. Implementieren Sie eine Methode `sortieren(Kruemel[] kruemelArray)`, welche die Elemente im übergebenen Array nach dem Verhältnis von Kalorien und Gewicht absteigend sortiert (siehe Methode `verhaeltnis()` der Klasse `Kruemel`). Selbstverständlich dürfen Sie weitere Hilfsmethoden für Ihren Sortieralgorithmus implementieren, falls benätigt.
   Ihre Lösung:
   ```
   public void sortieren(Kruemel[] kruemelArray){
   ```
2. Schreiben Sie eine Methode `beladen(Kruemel[] kruemelArray)`, die den übergebenen Array nach dem oben beschriebenen Algorithmus zum Beladen der Ameise verarbeitet. Die Methode soll eine Liste von `Kruemel`-Objekten zurückliefern, sodass die Ameise einerseits mäglichst viele Kalorien mitnimmt und andererseits die Summe der Gewichte der getragenen Krümel nicht die Kraft der Ameise übersteigt (eine Einheit Gewicht entspricht einer Einheit Kraft; die Kraft der Ameise beträgt 30 und wird im obenstehenden Code durch die Instanzvariable `hebeKraft` repräsentiert). Die Funktionalität zum Sortieren aus der vorangehenden Teilaufgabe dürfen Sie wiederverwenden, bzw. als gegeben annehmen, falls Sie die Aufgabe nicht läsen konnten.
   Ihre Lösung:
   ```
   public List<Kruemel> beladen(Kruemel[] kruemelArray){
   ```

4 Aufgabe: Adressbuch-Applikation: Teil 2

4.1 Lernziele

1. Sie können eine dynamische Datenstruktur verwenden.
2. Sie können eine einfache grafische Benutzerschnittstelle (GUI) implementieren.

4.2 Aufgabenstellung

In der letzten Übung haben Sie die Grundfunktionalitäten eines klassischen Adressbuchs programmiert. In dieser Aufgabe werden Sie einige Anpassungen am Code vornehmen, um die Applikation zu optimieren. Zudem werden Sie eine grafische Benutzerschnittstelle implementieren. Verwenden Sie für diese Aufgabe Ihre Lösung von letzter Woche. Vergleichen Sie diese zunächst mit der Musterlösung, um Folgefehler zu vermeiden.

a) Dynamische Datenstruktur

In der Aufgabe von letzter Woche, wurde zum Speichern der Adresse ein Array verwendet. Wie Sie bemerkt haben, stösst diese Datenstruktur an ihre Grenzen, wenn eine Liste ständig vergrössert und verkleinert wird. Für unseren Zweck besser geeignet ist eine dynamische Datenstruktur wie die `ArrayList`.

1. Ersetzen Sie den Datentyp Ihrer Liste durch eine `ArrayList`. Passen Sie Ihre Methoden entsprechen an. Konsultieren Sie hierzu die API oder die Slides aus der Übungsstunde 8.

b) Zusatzaufgabe: Grafische Benutzerschnittstelle (GUI)

Schauen Sie sich das Swing-Tutorial an. Sie brauchen noch nicht alles zu lesen, aber unter dieser Ressource werden Sie alle für diese Aufgabe benötigten Informationen finden. Bei Fragen und Unklarheiten sei auf das OLAT-Forum verwiesen.

1. Schauen Sie sich den Abschnitt *Using Swing Components : How To Make Frames (Main Windows)* an. Unter einem Frame versteht man in Java Swing ein Hauptfenster. Implementieren Sie nun Ihr eigenes Fenster und gehen Sie dabei wie folgt vor. Schreiben Sie eine Subklasse von `javax.swing.JFrame`, so können Sie die Funktionalität von `JFrame` übernehmen und mit der Adressbuch-Funktionalität anreichern. Schreiben Sie einen Konstruktor, der ein Objekt vom Typ `AddressBook` erwartet und dieses als Instanzvariable festhält. Dieser Konstruktor soll einen Superkonstruktor aufrufen, um ein Fenster zu erstellen.
2. Überlassen Sie dem Benutzer Ihrer Applikation die Wahl, ob er die Kommandozeilenschnittstelle (CLI) von letzter Woche oder das neue GUI benutzen will. Bei Starten der Main-Klasse von der Kommandozeile aus soll ein Parameter übergeben werden können, der das Userinterface bestimmt.

   ```
   1 java Main --cli
   ```

   Passen Sie Ihre Main-Klasse so an, dass standardmässig (d.h., wenn kein Parameter angegeben wird) das GUI und mit Angabe des Parameters `--cli` das CLI verwendet wird. Diese Parameter werden beim Aufruf der `main()`-Methode im String-Array `args` gespeichert.
3. Verwenden Sie nun Ihre `JFrame`-Subklasse in Ihrer Main-Klasse. Instanzieren Sie ein Objekt davon. Damit dieses Fenster auch sichtbar wird, müssen Sie noch die Sichtbarkeit setzen (siehe Java API, Dokumentation der Klasse `javax.swing.JFrame`). Wenn Sie jetzt Ihre Applikation laufen lassen, sollte Ihnen ein leeres Fenster angezeigt werden, das heisst nur die Fensterleiste mit den Minimieren-, Maximieren- und Schliess-Buttons (siehe Figure **??**).

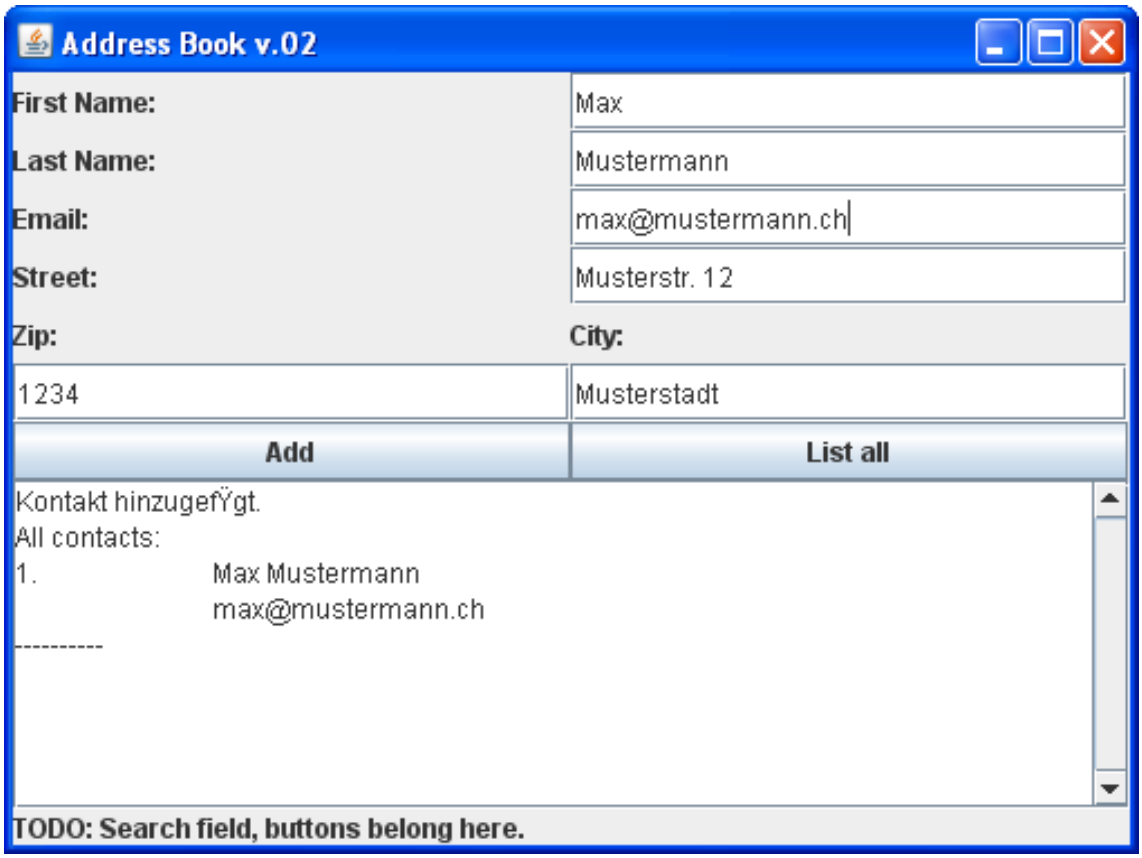

Figure 1: In etwa so sollte Ihr GUI schlussendlich aussehen (in diesem Screenshot sind noch nicht alle Funktionen implementiert).

Figure 2: die Fensterleiste mit den Minimieren-, Maximieren- und Schliess- Buttons.

4. Vielleicht ist Ihnen aufgefallen, dass beim Schliessen des Fensters die Applikation nicht beendet wurde. Lesen Sie den Abschnitt *Responding to Window-Closing Events* und ändern Sie das Verhalten Ihrer Klasse so, dass beim Schliessen des Fensters Ihr Programm beendet wird.
5. Studieren Sie das folgende Codebeispiel genau:

```
1 public class DemoFrame extends JFrame {
2    public DemoFrame() {
3       super("Demo Frame");
4
5       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6
7       createComponents();
8       pack(); // passt die Fenstergroesse den beinhaltenden
            Komponenten an.
```

```
9     }
10
11    private void createComponents() {
12       // Der ContentPane ist der Teil des Fensters, wo wir
13       // eigene Buttons, Textfelder, etc. hinzufuegen koennen.
14       Container pane = getContentPane();
15
16       // Ein Container kann beliebig viele Panels, Labels etc.
17       // beinhalten. In unserem Beispiel wollen wir zwei
18       // Label in einem Container halten.
19       Container someLabels = createSomeLabels();
20
21       pane.add(someLabels, BorderLayout.NORTH);
22    }
23
24    private Container createSomeLabels() {
25       Container someLabels = new JPanel();
26
27       // Layout manager sorgen dafuer, dass der Inhalt des
28       // Fensters automatisch an dessen Groesse angepasst wird.
29       GridLayout layout = new GridLayout(0,2);
30       someLabels.setLayout(layout);
31
32       JLabel label1 = new JLabel("Hallo miteinander!");
33       someLabels.add(label1);
34
35       JLabel label2 = new JLabel("Wie geht es euch?");
36       someLabels.add(label2);
37
38       return someLabels;
39    }
40 }
```

Listing 1: Beispielklasse

Wie Sie sehen, können Sie mit `JLabel` einen einfachen Text in einem Fenster darstellen. Mehrere solcher `JLabel` können Sie in einem `JPanel` (das ist eine Subklasse von `Container`) zusammenfassen.

6. Erstellen Sie ein Panel, das Eingabefelder für das Erstellen eines neuen Kontakts enthält, sowie ein Panel, das die Resultate anzeigt, analog zur Abbildung **??**. Dazu können Sie folgende Klassen verwenden:
   - `JLabel` um Text darzustellen
   - `JTextField` für ein Text-Eingabefeld mit wenig Text (um Name, E-Mail etc. entgegennehmen zu können)
   - `JTextArea` für ein grösseres Textfeld
   - `JButton`s für klickbare Buttons.
7. Damit Buttons beim Klicken auch irgendeine Aktion ausführen, müssen Sie den Button einen sogenannten `ActionListener` hinzufügen. Diese `ActionListener` funktionieren nach dem Listener/Observer-Prinzip, das Sie in der vorherigen Aufgabe kennengelernt haben

(siehe LectureListener-Beispiel). Dazu müssen Sie das `ActionListener`-Interface implementieren. Im Swing-Tutorial finden Sie im Abschnitt *How to Use Buttons, Check Boxes, and Radio Buttons* die notwendigen Informationen hierzu.

8. Optional können Sie noch die Möglichkeit anbieten, nach einem bestimmten Kontakt suchen zu können.