



# Informatik I – Eprog HS11

## Übung 7

### 1 Aufgabe: Interface

#### 1.1 Lernziele

1. Sie können eine Schnittstelle definieren und diese in Klassen implementieren.
2. Sie kennen die Unterschiede zwischen Interfaces und abstrakten Klassen.
3. Sie können typsichere Collections in Kombination mit der erweiterten `for`-Schleife nutzen (Features aus Java 5.0).

#### 1.2 Fläche

1. Erstellen Sie eine Klasse `Car`, welche die Attribute `Modell`, `Hersteller` und `Gewicht` besitzt und eine Klasse `SoccerField`, welche durch die Attribute `Ortschaft` und `Heimteam` charakterisiert wird. Zu allen Instanzvariablen sollen accessor-Methoden existieren.
2. Die soeben erstellten Klassen sollen nun in einem weiteren Schritt so erweitert werden, dass sie über eine gemeinsame Schnittstelle verfügen, über welche die Länge, die Breite und die Höhe von Autos und Fussballfeldern abgerufen werden kann. Erstellen Sie dazu ein Interface namens `ISize`, welches die benötigten Methoden anbietet.
3. Erweitern Sie die Klassen `Car` und `SoccerField` um das Interface `ISize`. Nehmen Sie die dazu notwendigen Erweiterungen an den Klassen vor, damit die Grösse von Objekten über die neue Schnittstelle ermittelt werden kann. Stellen Sie je einen Konstruktor zur Verfügung, welcher die dafür benötigten Daten erwartet.
4. Erstellen Sie im TestDrivers unterschiedliche Objekte beider Klassen und legen Sie diese in einer typsicheren `ArrayList` ab. Ergänzen Sie ausserdem den TestDriver um eine Klassenmethode `getSquare()`, welche die Grundfläche der Objekte einer übergebenen Liste berechnet und diese der Reihe nach auf der Konsole ausgibt.
5. Überlegen Sie sich, weshalb wir für die Schnittstelle `ISize` ein Interface und keine abstrakte Klasse verwendet haben. Notieren Sie Ihre Überlegungen und nennen Sie alle Ihnen bekannten Unterschiede zwischen Interfaces und abstrakten Klassen.

## 2 Aufgabe: Interface

### 2.1 Lernziele

1. Sie können eine gemeinsame Schnittstelle schreiben und andere Klassen diese implementieren lassen.
2. Sie verstehen anhand einem ersten Beispiel die Vorteile einer gemeinsamen Schnittstelle.

### 2.2 Dreidimensionale Körper

#### a) Geometrische Figuren

Modellieren Sie die folgenden geometrischen Figuren jeweils in einer eigenen Klasse. Die Implementierung soll dabei nur so detailliert sein, dass für jede Figur das Volumen abgefragt werden kann.

- Quader
- Reguläre Pyramide
- Gerader Kreiszylinder
- Kugel

Falls Ihnen die einzelnen Voluminaberechnungen nicht geläufig sind, hilft [diese Auflistung](#) weiter.

#### b) Volumina

Erweitern Sie die Klassen aus a) dahingehend, dass es möglich ist, unterschiedliche Instanzen in einem geeigneten Array abzulegen und in einer Schleife alle Volumina auszugeben.

## 3 Aufgabe: Loosely Coupled Interaction

### 3.1 Lernziele

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

### 3.2 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, spricht 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 verwenden.
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.

## 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 6.

#### b) Grafische Benutzerschnittstelle (GUI)

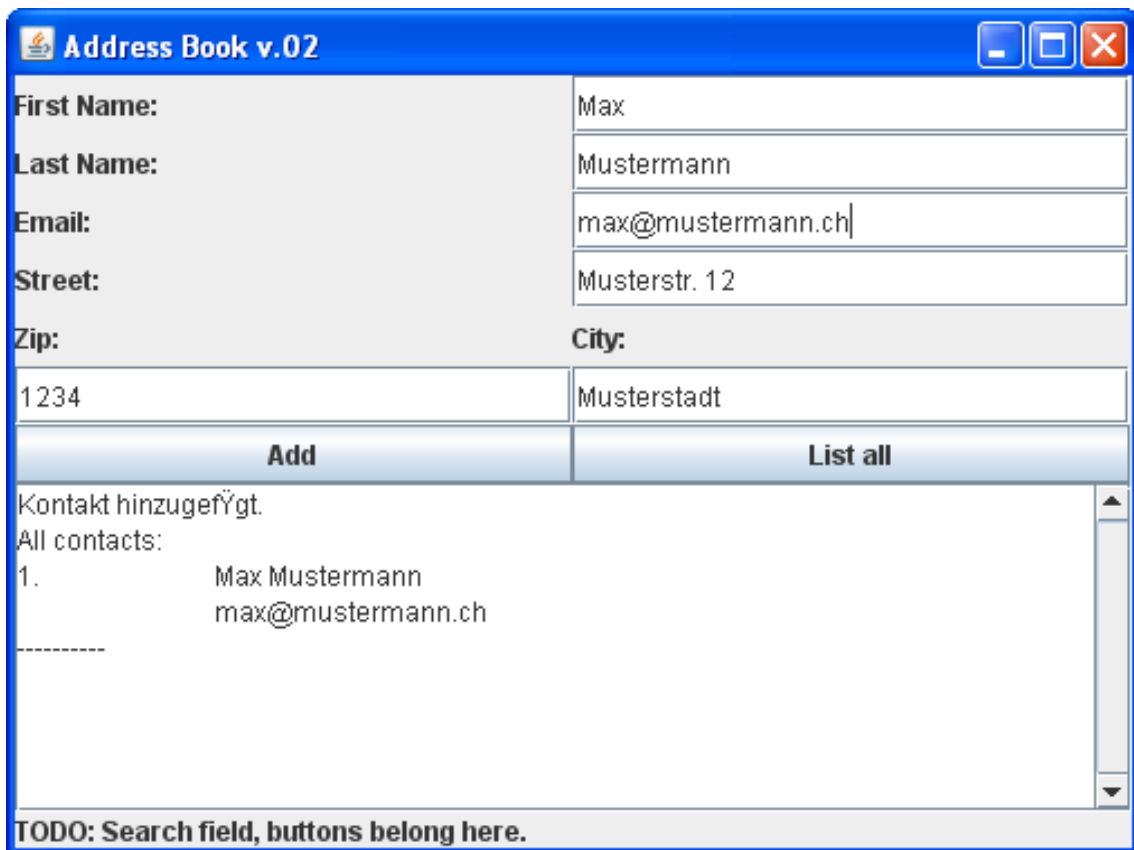
Schauen Sie sich das [Swing-Tutorial](#) an. Sie brauchen noch nicht alles zu lesen, aber unter dieser Ressource werden Sie alle für diese Aufgabe benötigten Informationen finden. Bei Fragen und Unklarheiten sei auf das OLAT-Forum verwiesen.

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 2).



**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 1. 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
  - `JButtons` 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.