University of Zurich[UZH]

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

Informatics I – EProg HS15

Exercise 10

1 Task: Loosely Coupled Interaction

1.1 Learning Objectives

1. You are able to implement a loosely coupled interaction between different objects by using interfaces.

1.2 Task Description

Given is the following situation:

A soccer team has a coach, assistants and players. Each of them can be added to a team and also be removed. If the team wins a match, all team members get a bonus. This bonus variates depending on team that has been beaten. Therefore, if a bonus will be paid all team members get a message that informs them about the amount of the bonus.

1. Implement a class `SoccerTeam` and define the public interface of Team Member using a Java interface `ITeamMember`.
   - `SoccerTeam`
     - constructor that expects the name of the soccer team
     - `registerTeamMember`, registers a new team member
     - `unregisterTeamMember`, remove a team member from the team
     - `distributeBonusInformation`, communicates a textual message to all its registered team members, which can be done by calling their method `receiveBonusInformation()`. The name of the team that has been beaten should be passed too.
   - `ITeamMember`
     - `receiveBonusInformation`, Allows a team member to receive the bonus information and the name of the beaten team.
2. Write two classes `Coach` and `Player` which both should contain a name and implement the interface `ITeamMember`. The objects of the two classes should print the received bonus information and the name of the beaten team to the console in different manners. (e.g. `Coach`: „My team beat the team XYZ because of my clever strategy and therefore I get...“, `Player`: „I, Lionel Messi, was the key of our victory over team XYZ and therefore I get...“)

3. Write a TestDriver and create two different soccer teams as well as two coaches and four soccer players who register themselves for one soccer team. Now the coaches should communicate some bonus news to the team members. One of the coaches and one of the players decides to leave their soccer team and inform the soccer team. The soccer team then sends some other bonus informations to the remaining team members.

2 Task: Recursion

2.1 Learning Objectives

1. Practice recursive programming

2.2 Task Description

Write a **recursive** method that solves the following tasks:

- Write a method that returns the fibonacci sequence of 2 starting numbers with a given length. The k-th element of the fibonacci sequence is defined by the formula $k_i = k_{i-2} + k_{i-1}$. The first input of the method should be the length of the returned sequence. The second and third input should be k_0 and k_1 which are the first two numbers of the sequence. The return value should be the computed sequence as an array of integers. In the main method, the returned sequence should be written to the console.
  Example: The call `fibonacci(6, 0, 1)` shall return `[0 1 1 2 3 5]`.
- Write a method that prints all the anagrams of a given word (i.e. all possible permutations that can be built with the characters of the given word).
  Example: The call `printAnagram("abc")` shall print the words *abc*, *acb*, *bac*, *bca*, *cab*, and *cba*.

3 Task: Sorting

3.1 Learning Objectives

1. You are able to implement a sorting algorithm.

3.2 Task Description

Read the following problem description and the further down described algorithm before implementing the subtasks.

An ant is on a picnic place and wants to collect the huge amount of crumbs that are strewed. The crumbs have a different weight and a different amount of calories.

The ant breaks out in tears when it realized that it is not able to bring all the crumbs at home, because it can only wear 300 times its body weight. Help the ants to wear as much alimentary crumbs as possible by implementing step by step an algorithm in Java that allows the ant to bring as much calories as possible at home (hint: large crumbs may not be the best selection if they do not have many calories):

1. Sort all crumbs according to their proportion between calories and weight (calories/weight) in descending order.
2. Load the first/next crumb on the sorted heap of crumbs up to the back of the ant.
3. Repeat the last step until the next added crumb would exceed the maximum lifting force of the ant or the last crumb of the heap has been passed. In the first case continue with step 4, in the second case continue with step 5.
4. Jump over the crumb that is overweighted and go to the next crumb on the heap and continue with step 3.
5. The ant is completly loaded up and can start walking.

In addition to the above description the following code is given:

```
1  import java.util.List;
2
3  public class Ant {
4
5    private static Crumb[] heapOfCrumbs;
6    private List<Crumb> loadedUpCrumbs;
7    private float liftingForce;
8
9    public Ant() {
10     loadedUpCrumbs = new java.util.ArrayList<Crumb>();
11     liftingForce = 30;
12     Ant.strewCrumbs();
13   }
14
15   public static void strewCrumbs() {
16     java.util.Random r = new java.util.Random();
17     heapOfCrumbs = new Crumb[10];
18     for (int i = 0; i < heapOfCrumbs.length; i++) {
```

```
19      float weight = r.nextInt(10) + 1;
20      float calories = r.nextInt(10) + 1;
21      heapOfCrumbs[i] = new Crumb(weight, calories);
22    }
23  }
24
25  public void sortHeapOfCrumbs() {
26    // Task: sort these crumbs according the proportion in descending order
27  }
28 }
29
30 class Crumb {
31  private float weight;
32  private float calories;
33
34  public Crumb(float weight, float calories) {
35    this.weight = weight;
36    this.calories = calories;
37  }
38
39  public float getWeight() {
40    return weight;
41  }
42
43  public float getCalories() {
44    return calories;
45  }
46
47  public float getProportion() {
48    return calories / weight;
49  }
50 }
```

1. Implement a method `sortHeapOfCrumbs()` that sorts elements of the `heapOfCrumbs` array of the ant according to the proportion of calories and weight (see method `getProportion()` of the class `Crumb`). Use **Selection Sort** as sorting algorithm. Of course you are allowed to implement helper methods for your sorting algorithm, if necessary.

2. Write a method `loadUp()` that loads the crumbs of the sorted array onto the back of the ant. The method shall return a list of crumbs that the ant take home such that the total number of calories is maximized and the total weight of the crumbs doesn't exceed the lifting force of the ant. The lifting force of the ant is 30.

4 Task: Adressbuch-Applikation: Teil 3 (optionale Zusatzaufgabe)

Important: *This exercise covers some material that has not been covered by the lecture so far. It furthermore covers GUI programming, which is NOT relevant for the final exam. The exercise is therefor facultative. Code solutions will be distributed but the task will not be discussed in the exercise lectures. We apologize that the task description itself is only provided in german.*

4.1 Learning Objectives

1. 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 Übung 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.

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).
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:

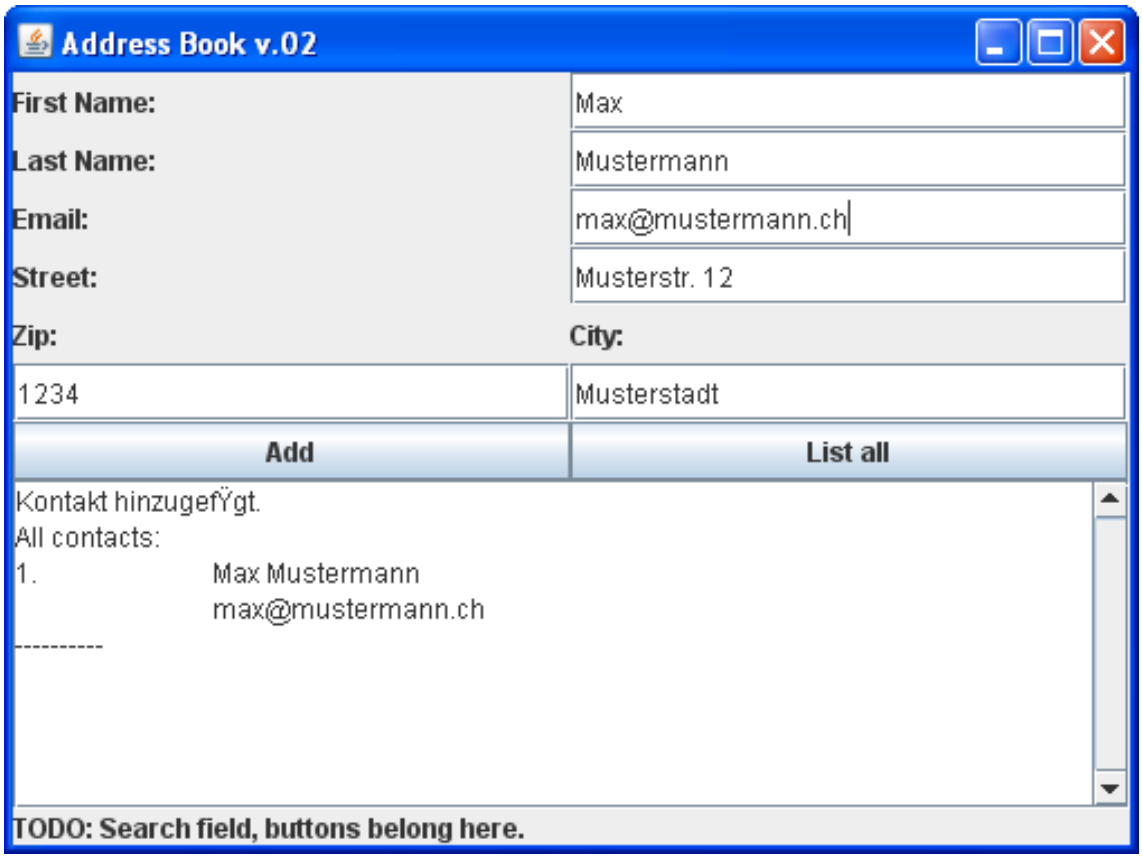

Figure 1: In etwa so sollte Ihr GUI schlussendlich aussehen (in diesem Screenshot sind noch nicht alle Funktionen implementiert).

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

```
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
   - `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.