Funktionale und objektorientierte Programmierkonzepte

Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.3

Übungsblatt 11

Themen:	GUI, Streams + Files
Relevante Folien:	Threads + GUIs, Streams + Files
Abgabe der Hausübung:	01.02.2019 bis 23:55 Uhr

V Vorbereitende Übungen

V1 Theoriefragen

☆ ☆ ☆

Welche der folgenden Aussagen ist wahr?

(A) Streams können aus Listen und Arrays erzeugt werden.

(B) Streams bieten keine Möglichkeit für elementweisen Zugriff.

(C) Ein Objekt der Klasse Path verwaltet den Pfadnamen einer Datei oder eines Verzeichnisses oder eines anderen Objektes, das im jeweiligen Dateisystem über einen Pfadnamen ansprechbar ist.

(D) Streams können in sich Daten speichern.

(E) Byteweiser Zugriff ist nur dann sinnvoll, wenn eine Datei nur Text und keine Bilder, Audio- oder Videodateien enthält, da bei einem Text die Bytes leichter gelesen werden können.

(F) Runnable ist ein Functional Interface. Die funktionale Methode heißt run, hat keine Parameter und ist void.

(G) Einzelne Threads können nicht terminiert werden, sie enden alle mit dem Ende des Gesamtprogramms.

(H) Eine Parallelisierung mit Threads verkürzt immer die Laufzeit eines Programms.

(I) Wann immer auf einen Button geklickt wird, wird Methode `actionPerformed` jedes bei diesem Button registrierten Listeners aufgerufen.

(J) Für jede Art von Listener gibt es eine eigene Registrierungsmethode.

(K) Die Klasse Canvas bietet eine unbegrenzte Zeichenfläche in einem Fenster.

V2 Vereinfachung mittels Lambda-Ausdrücken ★ ☆ ☆

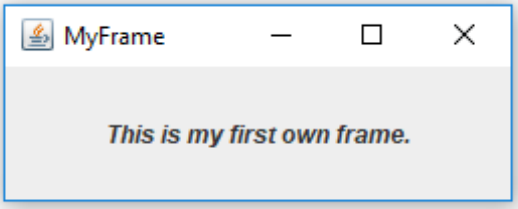
Gegeben sei nachstehender Java-Code. In diesem wird aus einer Liste von Zahlen der Durchschnittswert aller *positiven geraden* Zahlen berechnet, in einer Variablen gespeichert und zurückgegeben.

```
1  double averageOfEvenNumbers(int[] numbers) {
2    int sum = 0, count = 0;
3    for (int i = 0; i < numbers.length; i++) {
4      if (numbers[i] > 0 && numbers[i] % 2 == 0) {
5        sum += numbers[i];
6        count++;
7      }
8    }
9    return ((double)sum) / count;
10 }
```

1. Nutzen Sie Streams und Lambda-Ausdrücke um den Code zu verkürzen.
2. Beide Implementierungen – die "normale" und die Stream-Variante – weisen eine konzeptionelle Unsauberkeit auf. Bei beiden kann es zum Auftreten einer Exception kommen.
   (a) An welcher Stelle kann es bei der "normalen" Implementierung zu einer Exception kommen? Um welche handelt es sich?
   (b) An welcher Stelle kann es bei der Implementierung mittels Streams zu einer Exception kommen? Um welche handelt es sich?
3. Modifizieren Sie den Code entsprechend, dass die Exceptions aus Teilaufgabe 2 gar nicht mehr auftreten können, sondern die Methode für alle übergebenen Arrays fehlerfrei durchläuft.

V3 Mein eigenes kleines Fenster ★ ☆ ☆

Implementieren Sie ein kleines Programm mit einer GUI, welche genauso aussieht wie in der nachfolgenden Abbildung:



V4 Innere Klassen und Scope ★★☆

Betrachten Sie den untenstehenden Java-Code, welcher eine innere Klasse `InnerClass` enthält. In dieser wiederum ist eine Methode definiert, die einen Lambda-Ausdruck enthält, dessen Operation mit der Methode `accept` des Interfaces `Consumer` aufgerufen wird.

```
1  import java.util.function.Consumer;
2  public class LambdaScope {
3    public int x = 0;
4
5    class InnerClass {
6      public int x = 11;
7
8      void methodInInnerClass(int x) {
9        int z = 55;
10
11       Consumer<Integer> myConsumer = (y) ->
12          {
13            System.out.println("x = " + x);
14            System.out.println("y = " + y);
15            System.out.println("this.x = " + this.x);
16            System.out.println("LambdaScope.this.x = " +
                 LambdaScope.this.x);
17            System.out.println("z = " + z);
18          };
19        myConsumer.accept(x);
20       }
21   }
22
23   public static void main(String[] args) {
24     LambdaScope scope = new LambdaScope();
25     LambdaScope.InnerClass fl = scope.new InnerClass();
26     fl.methodInInnerClass(123);
27   }
28 }
```

Welche Ausgabe wird bei der Ausführung des Codes auf der Konsole ausgegeben? Überlegen Sie sich die Antworten ohne die Nutzung eines Compilers!

V5 Zeilen nummerieren ★★☆

Implementieren Sie die Methode `void insertRowNumbers(String path)`, welche den Pfad einer Text-Datei als String übergeben bekommt. Die Methode soll den Text der Datei Zeile für Zeile einlesen. Beginnend ab der ersten Zeile soll dann in jeder zweiten Zeile nun die Zeilennummer eingefügt werden. War die alte Text-Datei folgendermaßen aufgebaut `"Row1 \n Row2"` so soll die neue Text-Datei so aussehen: `"1 \n Row1 \n 2 \n Row2"`. Dabei steht `"\n"` für einen Zeilenumbruch.

V6 Bäckerei gesucht ★ ★ ★

Sie planen eine Party und benötigen eine Menge Brötchen dafür. Sie suchen nun den Bäcker in Ihrer Umgebung, der Ihnen die günstigsten Brötchen verkaufen kann. Dafür gibt es Objekte des Typs `Bakery`, welche zum einen zwei `public double`-Attribute `distance` (gibt die Distanz zu Ihnen in km an) und `price` (gibt den Preis pro Brötchen an) besitzt. Außerdem gibt es Objekte des Typs `BakeryOffer`, welche ebenfalls zwei `public`-Attribute `bakery` vom Typ `Bakery` und `totalPrice` vom Typ double besitzt. Der Konstruktor der Klasse sieht folgendermaßen aus:

```
public BakeryOffer(Bakery b, double tp){
this.bakery = b;
this.totalPrice = tp; }
```

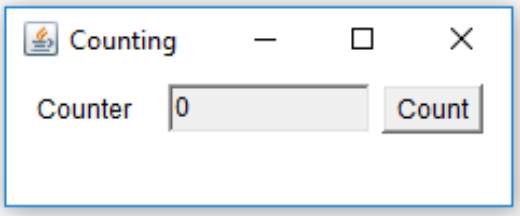
Implementieren Sie nun eine `public`-Methode vom Rückgabetyp `List<BakeryOffer>` mit dem Methodenkopf:

```
sortedBakeryOffers(Collection<Bakery> bakeries, double maxDistance)
```

Die Methode erhält eine Sammlung von Bäckereien und die maximale Distanz zu einer solchen. Zurückgeliefert werden soll eine nach Gesamtpreis aufsteigend sortierte Liste mit Angeboten des Typs `BakeryOffer`. Bäckereien, die weiter als die übergebene maximale Distanz entfernt sind, sollen von der Betrachtung ausgeschlossen werden.

V7 Zähler ★ ★ ★

Implementieren Sie ein kleines Programm mit einer GUI, welche genauso aussieht wie in der nachfolgenden Abbildung:



Jedes Mal wenn der *Count*-Button gedrückt wird, soll sich der Wert im Feld um 1 erhöhen.

Hinweis: Ihr Programm besteht aus drei Komponenten:

1. `java.awt.Label "Counter"`
2. `non-editable`[1] `java.awt.TextField`
3. `java.awt.Button "Count"`

[1] `http://programmedlessons.org/java5/Notes/chap60/ch60_13.html`

H Elfte Hausübung — Gesamt 7 Punkte

Campus-Management

In dieser Hausübung befassen wir uns mit einem kleinen, selbstgeschriebenen Campus Management System, welches Sie in Abbildung 1 sehen.

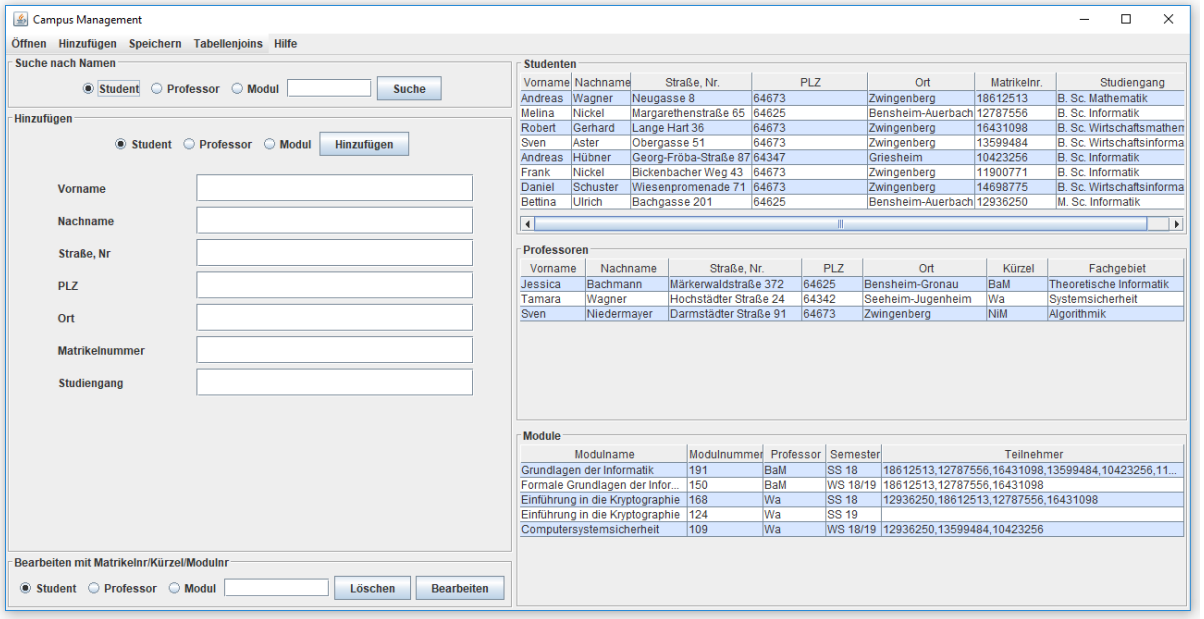


Abbildung 1: Screenshot aus dem Hauptmenü unseres Campus Management Systems.

Wie Sie leicht sehen können, verwaltet unser System drei Objektarten - Studenten, Professoren und Module. Jede dieser Gruppen hat dabei verschiedene Attribute, welches sie klassifiziert. Die Hauptfunktionen unseres Systems liegen darin, dass Einträge dieser Gruppen hinzugefügt, bearbeitet oder gelöscht werden können. Gespeichert werden die Daten dann in Dateien des Typs `.txt`. Um diese Dateien zu öffnen oder zu speichern, stehen in der oberen Taskleiste Drop-Down Menüs bereit.

H1 Entdecken des Campus Management Systems — 0 Punkte

Verschaffen Sie sich zunächst einen Überblick über das System. Die graphische Oberfläche ist bereits fertig gestaltet, Sie müssen zunächst also nur die Funktionen selbst implementieren.

Werfen Sie also neben dem Blick auf die GUI auch einen tieferen Blick in die Codebasis! Die angegebenen relevanten Foliensätze sollten Ihnen dabei enorm helfen.

H2 Suche 2 Punkte

Ergänzen Sie in der Klasse `Manager` die drei Methoden:

- `public ArrayList<Student> searchStudentName(String name)`
- `public ArrayList<Professor> searchProfName(String name)`
- `public ArrayList<Module> searchModuleName(String name)`

In der Klasse gibt es drei `ArrayLists` namens `students`, `profs` und `modules`. Diese Listen sollen dahingehend durchsucht werden, ob der übergebene String `name` Teilstring der jeweiligen Namen ist. Bei den Studenten und Professoren durchsuchen Sie den Vor- und Nachnamen nach dem String, bei den Modulen den Modulnamen. Zurückgegeben wird eine Liste von Studenten/Professoren/Modulen, bei denen im Vor-, Nach- oder Modulnamen der String `name` gefunden wurde.

Wurde in einer der drei Methoden nichts gefunden, so ist über die Methode `showMessageDialog` der Klasse `JOptionPane` eine Meldung auszugeben. Für Studenten könnte diese beispielsweise folgendermaßen lauten:

`"Es gibt keinen Studenten mit Teil - Namen: xyz"`

H3 Anzeigen 1 Punkt

Ergänzen Sie in der Klasse `Manager` die drei Methoden:

- `public void showStudents(ArrayList<Student> list)`
- `public void showProfs(ArrayList<Professor> list)`
- `public void showModules(ArrayList<Module> list)`

Die drei Methoden sollen die jeweils übergebene Liste in den MainFrame schreiben. Dabei können Sie die Liste des MainFrames über `mainFrame.studentModel` beziehungsweise `mainFrame.profModel` oder `mainFrame.moduleModel` ansprechen. Sollten gerade schon Objekte im MainFrame angezeigt werden, so löschen Sie diese zunächst und zeigen Sie danach die neuen Objekte der übergebenen Liste an.

Ist die Länge der übergebenen Liste ungleich der Länge der Klassenattribute `students`, `profs` und `modules` so setzen Sie das Attribut `filteredStudents` bzw. `filteredProfs` oder `filteredModules` auf `list`, andernfalls auf `null`.

Nutzen Sie am Ende `mainFrame.resizeColumnWidth(mainFrame.studentTable)` bzw. `mainFrame.profTable` oder `mainFrame.moduleTable`, um die Tabellenbreite anzupassen.

H4 SearchHandler 2 Punkte

Ergänzen Sie die Methode `public void actionPerformed(ActionEvent e)` in der Klasse `SearchHandler`. Greifen Sie zunächst auf den Text zu, der in das Suchfeld namens `searchfield` im MainFrame eingetippt wurde. Ist das Suchfeld leer, so geben Sie wieder eine Meldung über `showMessageDialog` aus. Andernfalls machen Sie eine Fallunterscheidung, welcher der drei Buttons Student/Professor/Modul ausgewählt wurde. Nach dem Klick auf den *Suche* Button sollen nur noch gefilterte Ergebnisse angezeigt werden, welche im Namen den gesuchten String enthalten.

Hinweis:

Setzen Sie an dieser Stelle auch die Attribute `manager.filterStudents`, `manager.filterProfs` und `filterModuls` geeignet.

H5 Neue Suchfunktion 2 Punkte

Zum Abschluss eine kreativere Aufgabe. Erweitern Sie die Suchfunktion des Campus Management Systems, sodass auch nach anderen Attributen als dem Namen gesucht werden kann. Überlegen Sie sich selbst eine Darstellungsmöglichkeit (z.B. Drop-Down-Menüs o.Ä.), welche Ihnen hier passend erscheint.

Beachten Sie außerdem, dass Studenten, Professoren und Module alle andere Attribute besitzen. Der Nutzer muss also je nachdem welche Gruppe ausgewählt ist, aus den passenden Attributen zur Suche wählen können.

Um die beiden Punkte dieser Aufgabe zu erhalten, muss folgendes umgesetzt werden:

1. Die erweiterte Suchfunktion muss in der GUI auswählbar sein und die passenden Attribute für die jeweiligen Suchen anzeigen.
2. Die Funktionalität muss korrekt implementiert sein.