252-0842-00L Programmieren und Problemlösen **Aufgabenblatt 4**
FS 2015 Herausgabe: 2015-03-11
Dr. A. L. Schüpbach, D. Sidler, Z. István Besprechung: 2015-03-18
Departement Informatik, ETH Zurich

1 Eigene Exceptions Auslösen

In Aufgabe 2.2 des letzten Aufgabenblattes mussten Sie Exceptions wie `IOException` und `ArrayIndexOutOfBoundsException` abfangen. Vielleicht haben Sie gemerkt, dass es nützlich sein könnte eigene Exceptions zu generieren, z.B. wenn eine Datei eingelesen wird, die nicht dem Format entspricht, dass Sie definiert haben.

In dieser Übung werden Sie gebeten, Ihren Code aus Übung 2.2 (Aufgabenblatt 3) mit eigenen Execptions zu erweitern. Erstellen Sie dazu spezielle Exception-Klassen, die von der Klasse Exception ableiten und überschreiben Sie entsprechende Methoden. Mittels

```
throw new IhreException();
```

können Sie diese Execptions dann auslösen.

Tipp: Auch wenn zwei Matrizen korrekt in zwei verschiedenen Dateien abgespeichert wurden, kann es immer noch passieren, dass die Dimensionen für die Multiplikation nicht übereinstimmen. Dies hat im bisherigen Code zu einer `ArrayIndexOutOfBoundsException` geführt. Wahrscheinlich macht es auch hier Sinn, eine aussagekräftigere Exception zu werfen.

2 Vererbung

In dieser Aufgabe geht es um das Konzept der Vererbung. Dazu werden Sie eine Applikation programmieren, die mit Personenobjekten zu tun hat.

2.1 Klasse `Person`

Implementieren Sie eine `Person`-Klasse mit folgenden Eigenschaften: *(i)* Vorname, *(ii)* Nachname, *(iii)* Geburtsdatum, *(iv)* Vater, *(v)* Mutter, *(vi)* Adresse. Beachten Sie, dass `Person`-Objekte mit einem Elternteil oder gar keinen Eltern erstellt werden können.

Implementieren Sie für die Klasse `Person` eine `equal(Person p)`-Methode, die prüft, ob es sich bei zwei `Person`-Objekten, um die gleiche Person handelt. Zwei `Person`-Objekte werden als gleich angesehen, wenn sie den selben Namen, das selbe Geburtsdatum und die Namen der Eltern gleich sind. Überschreiben Sie auch die Methode `toString()` (die jede Klasse von `Object` erbt), so dass alle Informationen einer Person als Komma separierte Liste ausgegeben werden können.

2.1.1 Stammbaum

Benutzen Sie die eben programmierte `Person`-Klasse, um Ihren Stammbaum zu repräsentieren. Erstellen Sie einen Stammbaum mit mindestens zehn Personen. Speichern Sie die Personen dazu in einer geeigneten Datenstruktur ab. Datenstruktur soll es ermöglichen, dass einfach auf Personen via {*Vorname,Nachname*} zugegriffen werden kann.

Implementieren Sie nun einen Methode:

```
int AnzahlVorfahrenVon(String Vorname, String Nachname)
```

die eine entsprechende Person findet und die Anzahl vorfahren berechnet.

Tipp: Gehen Sie einfachheitshalber für diese Übung davon aus, dass {*Vorname,Nachname*} eindeutig eine Person in der von Ihnen gewählten Datenstruktur identifiziert.

2.2 An der Uni

Nun möchten wir zusätzliche Informationen zu einer Person abspeichern, z.B. sind gewisse Personen an einer Universität `Student` und andere `Professor`. Studenten haben Matrikelnummern, Professoren hingegen haben eine Personalnummer und Doktoranden haben beides.

Professoren und Doktoranden haben zusätzlich zu ihrer Heimadresse auch noch eine Adresse an der Uni. Wir gehen davon aus, dass die Adresse für alle Mitarbeiter der selben Universität die gleiche ist, die Büro-Adresse kann sich jedoch jeweils ändern.

Überlegen Sie sich, wie sie Studenten, Doktoranden und Professoren mittels Vererbung von der Klasse `Person` am besten erfassen können.

Überschreiben Sie die `toString()` Methode, so dass für Angestellte der Universität, die Adresse an der Uni ausgegeben wird, während für Studenten die Heimadresse benutzt wird.

3 Mini-Datenbank für Personen

Im letzten Aufgabenblatt mussten Sie Matrizen in Dateien abspeichern. In dieser Aufgaben sollen Sie nun dasselbe für Personen tun. Programmieren Sie die Methoden `store_persons(...)` und `load_persons(...)`.

`store_persons(...)` nimmt als Argument die von Ihnen gewählte Datenstruktur zum speichern von Personen und schreibt die entsprechenden Informationen in einen CSV[1]-Datei.

Tipp: Beachten Sie, dass die Anzahl Attribute für verschiedene Subtypen (z.B. Student und Professor) variieren. Erstellen Sie deshalb einen `serialize()` Methode, in der `Person`-Klasse, die die jeweilige Zeile in der CSV-Datei ausgibt und von allfälligen Subklassen entsprechend überschrieben werden kann.

`load_persons()` soll eine CSV-Datei einlesen und wieder die ursprüngliche Datenstrukture mit den gespeicherten Personen herstellen.

Tipp: Damit Sie den richtigen Subtyp einer Person generieren können, sollten Sie diese Information ebenfalls in der CSV-Datei abspeichern.

[1] CSV=Comma-separated values