1 Dateien Lesen/Schreiben

Überlegen Sie sich ein Format, das sich eignen würde, um eine Matrix in einer Textdatei zu speichern. Schreiben Sie nun eine Methode `load_matrix(String filename)` sowie eine Methode `store_matrix(int[][] matrix, String filename)`, die eine Matrix in diesem Format von einer Datei lesen bzw. in eine Datei schreiben kann.

Schreiben Sie ein Program, an das drei Argumente übergeben werden können (<input-file-1> <input-file-2> <ouput-file-3>). Das Programm soll zwei Matrizen, die in separaten Dateien gespeichert sind, multiplizieren und das Resultat in einer dritten Datei speichern. Verwenden Sie dazu die zwei oben erwähnten Methoden. Für die Matrix-Multiplikation können Sie die Methode aus Aufgabe 3.1 des letzen Aufgabenblatts verwenden.

Tipp: Sie müssen in dieser Übung noch keine Exceptions abfangen. Leiten Sie allfällige Execptions einfach weiter, indem Sie `throws Excpetion` zu Ihren Methoden hinzufügen, z.B.

```
public static void main(String[] args) throws Exception { ... }
```

2 Kommandozeile und Execptions

In Aufgabe 1 wurden die Argumente beim Starten der Java-Applikation direkt an das Programm übergeben. In dieser Aufgabe Entwickeln Sie eine Kommandozeile, so dass Sie das Programm mehrfach mit verschiedenen Argumenten ausführen können, ohne es jedes Mal neu zu starten.

2.1 Scanner Klasse

Mit der Scanner Klasse können Sie in Ihrem Java Program Input von der Konsole lesen. Im folgenden Beispiel wird jeweils eine Zeile gelesen und dass array `String[] args` wird mittels `split(" ")` Methode generiert. Die Endlos-Schleife sorgt dafür, dass das Programm nicht beendet wird, nachdem die erste Zeile gelesen wurde.

```
import java.util.Scanner;
public static void main(String[] prog_args) {
    Scanner input = new Scanner(System.in);
    while(true) {
      String command = input.nextLine();
      String[] args = command.split(" ");
      System.out.println(args[0]);
    }
}
```

Ändern Sie Ihr Programm aus Aufgabe 1 so ab, dass <input-file-1> <input-file-2> <ouput-file-3> mehrfach direkt über die Konsole an das Programm übergeben werden können. Mit einem "exit" Befehl soll das Programm beendet werden können.

2.2 Exceptions Abfangen

Was passiert wenn Sie als Input eine Datei angeben, die es nicht gibt? Das Programm stürzt vermutlich ab, da Sie die `FileNotFoundException` noch nicht abfangen. Ändern Sie das Programm, so dass der Benutzer darauf hingewiesen wird, dass eine Datei nicht exisitert, aber verhindern Sie, dass das Programm beendet wird.

Kann ein Benutzer nun das Programm nicht mehr zum Absturz bringen? Was passiert zum Beispiel, wenn die Datei, welche die Matrix enthält zwar existiert, aber korrumpiert ist? Ist es möglich allfällige Probleme ebenfalls via try & catch zu beheben?

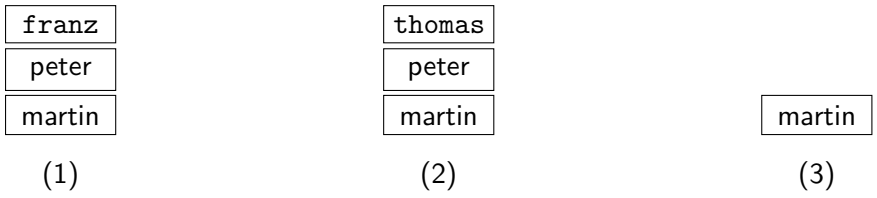
3 XML Parsen mit eigener Stapel-Datenstruktur

In dieser Aufgabe werden Sie mit der Hilfe einer eigenen Stapel (Stack) Klasse prüfen, ob XML-Dokumente wohlgeformt sind.

3.1 Stapel Implementierung

Ein Stapel ist eine Datenstruktur, bei der mit der Methode `push()` Elemente auf den Stapel aufgeladen werden und mit der Methoe `pop()` Elemente in der umgekehrten Reihenfolge wieder abgebaut werden. Das Beispiel unten illustriert, wie ein Stapel nach folgenden Operationen aussehen würde:

1. `push(martin),push(peter),push(franz)`
2. `pop(),push(thomas)`
3. `pop(),pop()`



Programmieren Sie diese Datenstruktur in einer eigenen (inneren) Klasse. Die Elemente des Stapels sollen vom Typ `String` sein. Implementieren Sie folgende Methoden als Schnittstelle zur Aussenwelt:

- `void push(String s)`: fügt ein neues Element zum Stapel hinzu.
- `String pop()`: Löscht das oberste Element vom Stapel und gibt den entsprechenden Wert zurück.
- `boolean empty()`: Wahr wenn der Stapel keine Elemente hat.
- `String top()`: Gibt den Wert des obersten Elementes zurück, ohne es zu löschen.

3.2 Wohl-geformte XML Dokumente

XML Dokumente müssen nach gewissen Regeln erstellt werden, z.B., müssen alle Tags <tag> wieder mit einem schliessenden Tag </tag> geschlossen werden. Zudem müssen Tags in der selben Reihenfolge geschlossen werden, wie sie geöffnet wurden. Damit ein XML Dokument als wohlgeformt gilt müssen noch weiter Kriterien erfüllt sein, aber wir beschränken uns in dieser

Übung auf die zwei oben genannten Punkte. Unten ist links ein XML Dokument abgebildet, das wir als korrekt einstufen würden und rechts ein inkorrektes XML Dokument:

```
<foo>
  <bar>
    <foo>
      <barbar>
      </barbar>
    </foo>
  </bar>
</foo>
```

```
<foo>
  <bar>
    <foo>
      <barbar>
      </barbar>
    </bar>
  </foo>
</foo>
```

Benutzen Sie Ihre Staple Klasse, um zu testen, ob ein XML Dokument gemäss den oben genannten Regeln wohlgeformt ist. Sie können zur Vereinfachung annehmen, dass pro Zeile maximal ein XML-Tag steht. Zudem können Sie davon ausgehen, dass die Tags selber korrekt formatiert sind (Sie müssen also nicht Fälle wie `<foo<bar>` erkennen).

Tipp: Die Klasse `String` bietet die Methode `trim()` an, die Leerzeichen am Anfang und am Ende eines Strings löscht. Zudem kann mit der Methode `substring()` auf einen Teil-String zugegriffen werden.

4 Java Datenstrukturen

In dieser Aufgabe sollen Sie eine von uns bereitgestellte CSV-Datei[1] einlesen, welche Daten zu den wichtigsten Schweizer Bergen enthält. Ihre Aufgabe ist es, ein Programm zu schreiben, dass den höchsten Berg in jedem Kanton auflistet. Die relevanten Informationen dazu finden Sie in den Spalten 4 (Höhe) und 5 (Kanton) der CSV-Datei.

Probieren Sie verschiedene Datenstrukturen aus der Java-Bibliothek aus, um das Problem zu lösen. Speichern Sie pro Kanton einen Eintrag in dieser Datenstruktur und aktualisieren Sie die jeweiligen Einträge währenddem, Sie die CSV-Datei einlesen. Bitte beachten Sie auch, dass ein Berg in mehreren Kantonen sein kann (z.B. VS/F) und natürlich hat es mehrere Berge pro Kanton.

Tipps: Die CSV-Datei hat eine Kopfzeile, welche Sie in Ihrem Code ignorieren sollten. Ändern Sie aber nicht die CSV-Datei ab. Sie müssen die Kantone nicht hart-kodieren; Sie können neue Kantone in die Datenstruktur einfügen, währenddem Sie die CSV-Datei einlesen. Eine geeignete Datenstruktur, wäre eine, welche schnellen Zugriff auf zufällige Elemente erlaubt. Überlegen Sie sich, welche Datenstrukturen geeignet sind und welche eher nicht.

[1] https://svn.inf.ethz.ch/svn/systems/teaching/ppl/vorlesungsprogramme/gipfelverzeichnis.csv