

Übungsblatt 10 - Abgabe 10.01.05

Aufgabe 1 (Infix/Postfix)

In Abschnitt 5.1.3.2 der Vorlesung wurde ein Algorithmus zur Auswertung von Infix-Ausdrücken vorgestellt. Der Algorithmus verwendet einen Operanden- und einen Operatoren-Keller. Die Operatoren werden so bald wie möglich angewendet. Hierbei wird im Prinzip die Infix-Notation in eine Postfix-Notation umgewandelt, die jedoch durch die Anwendung der Operatoren gleich wieder abgebaut wird.

Entwickeln Sie in Anlehnung an Abschnitt 5.1.3.2 einen Algorithmus, der Infix-Ausdrücke in Postfix-Ausdrücke umwandelt. Ausdrücke sollen aus den Operatoren $+ - * / ^$ mit den bekannten Vorrängen und Variablennamen der Länge 1 (einzelne Buchstaben) bestehen und keine Leerstellen enthalten, z.B.

Infix: $a * (c + d) / (f - e + g)$

Postfix: $acd+*fe-g+ /$

Achten Sie auf die Robustheit Ihres Algorithmus bezüglich Syntaxfehler.

Aufgabe 2 (Iteratoren)

In Anlehnung an Funktionale (Funktionen, die Funktion als Parameter haben), die aus der funktionalen Programmierung bekannt sind, werden die Signaturen einiger interner Iteratoren wie folgt vereinbart:

```
interface Traversable<T> {
    void map(UnaryFunction<T,T> f);    // in situ replacement
    <V> V fold(BinaryFunction<T,V,V> f, V init);
    List<T> filter(Predicate<T> p);
}
interface UnaryFunction<X,Y> {...}
interface BinaryFunction<X,Y,Z> {...}
interface Predicate<T> {...}
```

- Vervollständigen Sie die angedeuteten Schnittstellen und geben Sie Beispiel-Klassen für deren Implementierung an! (Zur Erinnerung: 1.2.3)
- Entwickeln Sie eine Klasse

```
class TraversableStack<T>
    implements Traversable<T> extends Stack<T>
```

(mit `java.util.Stack`). Die Iteratoren sollen so effizient wie möglich arbeiten; sie müssen daher direkt auf die Repräsentation des Kellers zugreifen.

Aufgabe 3 (Praktische Javakenntnisse 2 - Konfigurationsdateien)

Häufig müssen Anwendungsprogramme Einstellungen auf einem Rechner sichern und beim nächsten Start laden, damit ein Benutzer nicht bei jedem Start

der Anwendung alle Einstellungen erneut vornehmen muss. Mit Einstellungen sind z.B. Vorgaben für die Erstellung neuer Dokumente, die letzte Position eines Fensters, nicht aber die vom Benutzer mit der Anwendung erstellten Dokumente gemeint.

Solche Einstellungen sollten generell nicht in einer Konfigurationsdatei in dem Ordner, in dem sich die Anwendung befindet, gesichert werden: Zum Einen würden sich die Einstellungen dann automatisch auf alle Benutzer des Rechners auswirken, zum Anderen hat ein normaler Benutzer (also nicht Administrator) auf einen solchen Ordner keine Schreibrechte, so dass die Sicherung der Einstellungen fehlschlagen würde.

Ein anderer Ansatz wäre es, eine Konfigurationsdatei im Heimatverzeichnis des ausführenden Benutzers abzulegen. Es stößt jedoch auch nicht unbedingt auf Gegenliebe beim Benutzer, wenn Programme dessen Heimatverzeichnis mit Konfigurationsdateien "beschmutzen".

In der Java-API wird die Klasse `java.util.prefs.Preferences` mitgeliefert, mit deren Hilfe man Einstellungen in der für das jeweilige Betriebssystem typischen Art und Weise sichern kann (Windows: registry; Mac OS X: Ordner `~/Library/Preferences/` usw.).

Eure Aufgabe ist es, die gegebene Anwendung Currency Converter (<http://www.inf.fu-berlin.de/lehre/WS05/ALP3/material/U10.A3.zip>) so anzupassen, dass beim Beenden der Anwendung, die Position des Fensters (siehe Klasse `View`, Aufruf `frame.setLocation`) und der letzte gültigen Wechselkurs (siehe Klasse `Model`) mit Hilfe der Klasse `Preferences` gesichert und beim nächsten Start der Anwendung wieder geladen werden.

Tipp:

Ein Exemplar der Klasse `Preferences` erhaltet ihr, über die statische Methode `userNodeForPackage`. Als Parameter wird ein Klassenobjekt erwartet. Verwendet hier einfach `Main.class`, wobei `Main` die Startklasse der Anwendung Currency Converter ist.

Abgabe:

- Modifizierte Quellcodefragmente mit Angabe der zugehörigen Datei auf Papier.
- Vollständiger Quellcode und ausführbare JAR-Datei per Mail an den Tutor.