Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Dipl.-Systemwiss. Mathias Menninghaus

Universität Osnabrück, 05.06.2018
Testat bis 13.06.2018, 14:00 Uhr

Übungen zu Informatik B

Sommersemester 2018

Blatt 7

Aufgabe 7.1: Vergleich von Java Collections (26 Punkte)

Beurteilen Sie welche der `Collection`-Implementationen `java.util.LinkedList`, `java.util.ArrayList` und `java.util.HashSet` hinsichtlich der Methoden `add(T)` `remove(T)` und `contains(T)` die besten Laufzeiteigenschaften hat. Führen Sie dazu eine rein quantitative Analyse durch. Bilden Sie also durchschnittliche Werte für die Laufzeit der drei Methoden über eine ausreichende Anzahl von Testfällen. Geben Sie die Ergebnisse in einer aussagekräftigen Tabelle aus. Erklären Sie Ihrem Tutor/ Ihrer Tutorin schriftlich ob und warum die Ergebnisse Ihren Erwartungen entsprechen. Gestalten Sie Ihre Implementation derart, dass möglichst einfach neue Klassen in den Vergleich mit aufgenommen werden können, ohne viel Quellcode duplizieren zu müssen.

Hinweis: Die aktuelle Systemzeit kann in Java mit `System.nanoTime()` abgefragt werden.

Aufgabe 7.2: Visitor und Visitable (20 Punkte)

Betrachten Sie die Interfaces `Visitor` und `Visitable` und machen Sie sich mit deren Funktionsweise vertraut. Jede Klasse, die das Interface `Visitable` implementiert, soll beim Aufruf der Methode `accept(Visitor)` all ihre Elemente durchlaufen und für jedes Element die Methode `visit(Object)` der übergebenen `Visitor`-Instanz aufrufen. Dies wird so lange gemacht, bis entweder alle Elemente durchlaufen wurden, oder bis der `Visitor` `false` zurück liefert. Ein `Visitor` liefert also `true`, solange er noch weitere Elemente besuchen will.

Implementieren Sie das Interface `Visitable` in der Liste aus der Lösung des letzten Aufgabenblattes, so dass mit einem Aufruf von `accept` die Liste einmal vollständig durchlaufen wird, wenn der `Visitor` dies mit seiner Rückgabe zulässt.

Testen Sie Ihre Implementierung durch mindestens eine `Visitor`-Implementierung und lassen Sie das Testprogramm prüfen, ob auch wirklich alle Elemente durchlaufen wurden.

Aufgabe 7.3: Visitor - Pattern und Dateisystem (28 Punkte)

Setzen Sie das *Visitor-Pattern* nun auf andere Art und Weise um. Entwickeln Sie dazu zunächst eine Klasse, die das Dateisystem ab einer bestimmten Wurzel-Datei bzw. einem -Verzeichnis repräsentiert. Ein jedes solches Dateisystem soll mit einem von Ihnen definierten *Visitor* besucht werden können und damit alle Dateien, die sich unterhalb des Wurzelelements in der Verzeichnishierarchie befinden. Also

würde, wenn das Dateisystem nur eine einzelne Datei repräsentiert, auch nur eine `File` - Instanz, die diese Datei darstellt, dem *Visitor* vorgeführt. Beruht das Dateisystem auf einem Verzeichnis, würde ein *Visitor* alle Dateien in diesem und in all seinen Unterverzeichnissen, jeweils in Form einer `File`-Instanz, rekursiv vorgeführt bekommen.

Des Weiteren soll der *Visitor* noch in Teilen beeinflussen können, welche Teile des Dateisystems ihm vorgeführt werden. *Visitable* und *Visitor* sollen folgende Operationen möglich machen:

- Der Besuch aller weiteren Elemente soll abgebrochen werden können.
- Der Besuch aller Elemente in einem (Unter-)Verzeichnis soll ausgelassen werden können.
- Vor dem rekursiven Einstieg in ein Verzeichnis soll eine Methode im Visitor aufgerufen werden, so dass auf Implementierungsebene reagiert werden kann.
- Nachdem ein Verzeichnis vollständig durchlaufen wurde, soll eine Methode im Visitor aufgerufen werden, so dass auf Implementierungsebene reagiert werden kann.
- Über die Rückgabewerte der Visitor-Instanzen kann jeweils entschieden werden, ob und wie der Besuch in dem Verzeichnisbaum weiter geht.

Wenden Sie Ihre Interpretation des *Visitor*-Patterns nun an. Implementieren Sie dazu eine vereinfachte Version des `Unix`-Kommandos *ls*. Dies sei folgendermaßen definiert:

```
java List [-r] [DateiOderVerzeichnisname]
```

Generell soll die auf der Kommandozeile mit angegebene Datei, bzw. das Verzeichnis, soweit vorhanden, auf der Standardkonsole aufgelistet werden. Mit dem Kommando `-r` sollen auch alle Dateien und Unterverzeichnisse in dem angegebenen Verzeichnis rekursiv aufgelistet werden. Ist keine Datei und kein Verzeichnis angegeben, soll das Verzeichnis aufgelistet werden, von dem aus das Kommando ausgeführt wird.

Achten Sie auf eine strukturierte, übersichtlich Ausgabe, indem Sie Unterverzeichnisse ebenso wie Dateien passend zu Ihren Verzeichnissen einrücken.

Aufgabe 7.4: Persistentes Array (26 Punkte)

Implementieren Sie eine Wrapper-Klasse mit der `Integer`-Arrays persistent abgespeichert, durchlaufen und ihre Einträge verändert werden können. Eine Instanz dieser Klasse soll mit einem `Integer`-Array und einem Namen, unter dem das Array als Datei abgespeichert werden soll, instanziiert werden können. Existiert unter dem Namen bereits eine Datei, soll diese überschrieben werden. Alle Array-Einträge werden dann in die Datei geschrieben. Es soll auch möglich sein, auf ein bereits existierendes, persistentes Array durch Instanziierung der Wrapper-Klasse nur unter Angabe des richtigen Dateinamens Zugriff zu erlangen. Mit einer Instanz schließlich soll man die einzelnen Einträge einsehen und verändern können. Alle Änderungen sollen sofort persistent in die Datei geschrieben werden. Achten Sie darauf, das man auch die Anzahl der Einträge erfragen und die Datei explizit schließen kann.

Schreiben Sie anschließend eine Testklasse, die automatisiert die von Ihnen implementierten Funktionen testet. Testen Sie auch darauf, ob die von Ihnen angekündigten Exceptions korrekt geworfen werden.

Hinweis: Sie brauchen nicht mit Streams arbeiten.