RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

Lehrstuhl für Hochleistungsrechnen
RWTH Aachen · D-52074 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Christian Bischof, Ph.D.
Andre Vehreschild, Jakob T. Valvoda

Übung *Programmierung WS 06/07* – Blatt 10

Lösungen müssen bis zum **15. Januar 2007, 17:00 Uhr** in den Kasten Ihrer Übungsgruppe eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe**, sowie die **Namen** und **Matrikelnummern** der Mitglieder Ihrer 2er-Gruppe auf jedes Lösungsblatt Ihrer Abgabe zu schreiben. Bitte heften Sie ihre Blätter zusammen.

Alle erstellten Haskell-Programme sind sowohl in gedruckter Form abzugeben, als auch per E-Mail an den jeweiligen Tutor zu senden. Vergessen Sie auch bei der elektronischen Abgabe per E-Mail nicht die Angabe der **Nummer Ihrer Übungsgruppe**, sowie die jeweiligen **Namen** und **Matrikelnummern**.

Bitte beachten Sie, dass Studierende der Fachrichtungen CES, Werkstoffinformatik und Computermathematik ihre Übungen von Prof. Naumann erhalten (und diese dann auch lösen müssen).

Aufgabe 1 (7,5 + 1,5 = 9 Punkte)

In einem System soll die Behandlung von Problemfällen mit neu zu entwickelnden Fehlerklassen gesteuert werden. Alle Fehler sollen unabhängig von ihrer konkreten Art zentral durch die Klasse `FehlerManager` verarbeitet werden. Hierzu enthält die Klasse die Methoden `behandleFehler()`, `zeichneFehler()` und `korrigiereFehler()`, welche jeweils geeignete Fehler als Parameter erhalten.

Derzeit können im System fünf verschiedene Fehlerarten vorkommen: Graph-, Konversions-, Datei-, Bildaufbau- und Sortierfehler. Die Fehlerarten sind wie folgt spezifiziert:

- Es gibt zwei Kategorien von Fehlern: Eingabe- und Ablauffehler.
- Alle Eingabefehler enthalten eine Referenz auf die Klasse `Quelle`, welche die Eigabequelle repräsentiert. Zu den Eingabefehlern zählen Graph-, Konversions- und Dateifehler.
- Ablauffehler enthalten eine Referenz auf denjenigen Prozess, der den Ablauffehler verursacht hat (repräsentiert durch die Klasse `Prozess`). Zu den Ablauffehlern gehören Bildaufbau- und Sortierfehler.
- Ablauffehler sind korrigierbar und können mit ihrer Methode `bool korrigiere()` den Problemfall beseitigen. Diese Funktionalität wird vom `FehlerManager` in der Methode `korrigiereFehler()` genutzt.
- Im System sollen nur Instanzen der fünf beschriebenen Fehlerklassen zulässig sein.
- Jeder Fehler besitzt eine Methode `drucke()` welche eine Fehlermeldung auf der Konsole ausgibt.

- Graphfehler entstehen bei der Eingabe von Initialisierungsdaten der Klasse `Graph`. Die Daten stammen aus einer nicht näher spezifizierten Quelle. Der Graphfehler enthält eine Referenz auf den initialen Graphen. Neben der Methode `drucke()` besitzen Graphfehler die Möglichkeit den Problemfall mit ihrer Methode `zeichne()` zu visualisieren. Diese wird vom `FehlerManager` in der Methode `zeichneFehler()` aufgerufen.
- Konversionsfehler entstehen, wenn eine Zeichenkette in einen anderen Typen konvertiert werden soll. Diese Fehler enthalten eine Referenz auf die fehlerhafte Eingabe.
- Dateifehler werden erzeugt, wenn die Eingabe aus einer Datei fehlschlägt (z.B. wenn eine Datei nicht gefunden werden konnte). Die Referenz auf die Eingabedatei wird in der zugehörigen Quelle des Fehlers gespeichert. Dateifehler sind ebenfalls wie Ablauffehler korrigierbar und enthalten entsprechende Methoden.
- Bildaufbaufehler entstehen während der Verarbeitung. Sie können analog zu Graphfehlern visualisiert werden.
- Sortierfehler entstehen ebenfalls während der Verarbeitung und besitzen keine weiteren Merkmale.

a) Modellieren Sie die obigen Fehlerklassen und ihre Hierarchie. Verwenden Sie hierbei die in Übung 7 Aufgabe 3 vorgestellte UML-Notation. Nutzen Sie Gemeinsamkeiten der Fehlerarten und modellieren Sie diese als gemeinsame ggf. abstrakte Oberklassen bzw. Schnittstellen (interfaces). Geben Sie alle Attribute der einzelnen Klassen an. Vergessen Sie hierbei nicht die Sichtbarkeit und den Typen des Attributs anzugeben. Verwenden Sie als Typ die oben genannten Klassennamen bzw. die Klasse `String`. Geben Sie die Signaturen der Methoden in allen Klassen an, welche die Methoden deklarieren oder implementieren. Geben Sie alle Selektoren (`get`-Methoden) der einzelnen Attribute in den Klassen an. Beachten Sie, dass die Attribute nicht von aussen gesetzt werden können und nur bei der Konstruktion des Fehlers durch den `FehlerManager` initialisiert werden.

b) Geben Sie die vollständige Signatur der Methoden `behandleFehler()`, `zeichneFehler()` und `korrigiereFehler()` für Ihre Klassenhierarchie an, so dass die Methoden nur diejenigen Klassen bearbeiten, welche die spezifische Funktionalität haben. Gehen Sie hierbei davon aus, dass die Methoden jeweils `void` zurückliefern.

Hinweis: In dieser Aufgabe sollen **keine** Exceptions von Java genutzt werden.

Aufgabe 2 (4 Punkte)

Beantworten Sie folgende Fragen im Hinblick auf das Konzept der funktionalen Programmierung. Formulieren Sie Ihre Antworten in eigenen Worten!

a) Erläutern Sie den Begriff „referentielle Transparenz“.

b) Was sind Funktionen höherer Ordnung?

c) Diskutieren Sie den Unterschied zwischen dem „parametrischen Polymorphismus“ funktionaler Programmiersprachen und dem in Java verwendeten Polymorphismus.

d) Ein imperatives Programm besteht aus einer Abfolge von Anweisungen und Kontrollstrukturen und beschreibt, wie ein Problem gelöst werden soll. Charakterisieren Sie ein funktionales Programm. Was beschreibt ein funktionales Programm im Vergleich zu einem imperativen Programm?

Aufgabe 3 (3 Punkte)

Gegeben sind die folgenden Listendeklarationen:

a) `[0,2,7]`	b) `0:[2,7]`	c) `[0,2 .. 7]`
d) `[[0,2,7]]`	e) `[0,2:7]`	f) `[(0,2,7)]`

Geben Sie zu jeder Deklaration an, ob es sich um eine syntaktisch korrekt deklarierte Liste handelt. Falls es sich um eine syntaktisch korrekt deklarierte Liste handelt, dann geben Sie an, wieviele Elemente sie besitzt und welchen Typ sie hat.

Aufgabe 4 (2 + 2 + 1 = 5 Punkte)

Implementieren Sie die folgenden Funktionen in Haskell:

a) `istGleich :: Int -> Int -> Bool`, welche die Gleichheit zweier positiver Zahlen ermittelt und entsprechend `True` oder `False` zurückliefert. Das Ergebnis ist für negative Zahlen nicht definiert (sie müssen sich also um diese Fälle nicht kümmern). Verwenden Sie zur Definition der Funktion `istGleich` ausschliesslich nur die Operatoren , `+`‘ und , `-`‘ zur Addition und Subtraktion von ganzen Zahlen.

Hinweis: Verwenden Sie rekursive Aufrufe. Nutzen Sie ebenfalls das Pattern Matching, d.h. sie deklarieren die Funktion `istGleich` mehrfach. Die einzelnen Deklarationen werden in Abhängigkeit von den Argumenten ausgeführt. Beispielsweise wird die Deklaration mit dem Anfang `func x 0 = ...` nur dann ausgeführt, wenn das zweite Argument 0 ist (das erste Argument kann einen beliebigen Wert annehmen). Vergleichen Sie hierzu die Deklaration der Funktion `len` im Kapitel III.1 (Folie 7 und 8).

b) `maxZahl :: Int -> Int -> Int`, welche das Maximum zweier positiver Zahlen bestimmt und dieses zurückliefert. Verwenden Sie auch hier ausschliesslich nur die Operatoren , `+`‘ und , `-`‘ zur Addition und Subtraktion von ganzen Zahlen.

Hinweis: Beachten Sie den Hinweis im Aufgabenteil a).

c) `dreiVerschieden :: Int -> Int -> Int -> Bool`, welche drei `Int`-Zahlen vergleicht und `True` zurückliefert, falls alle drei verschieden sind. Verwenden Sie ausschliesslich nur die von Ihnen deklarierten Funktionen aus Aufgabenteil a) und b) sowie die boolschen Operatoren , `||`‘, , `&&`‘ und , `not`‘.

Hinweis: Testen Sie Ihre Funktion z.B. für die Eingabe `dreiVerschieden 3 4 3`.