Programmierparadigmen

Wintersemester 2012/13
Torsten Görg, Sandro Degiorgi

7. Übung
Einsendung bis spätestens 04.02.2013, 13:00 Uhr
Besprechungen am 06./07./08. Februar 2013

Aufgabe 7.1 ..... 21 Punkte

Der englischsprachige Foliensatz beschäftigt sich mit *OOP and Maintenance*, also der Wartung von objektorientierten Softwaresystemen und den Problemen, die hierbei auftreten können. Auf der Webseite der Veranstaltung finden Sie ein solches objektorientiertes Softwaresystem unter dem Dateinamen `Aufgabe_7_1.zip`. Das System besteht aus einer Bibliothek zur Erstellung grafischer Benutzeroberflächen. Die zur Bibliothek gehörenden Dateien sind in den Paketen `oop.lang` und `oop.awt` zu finden. Die eigentliche Benutzer-Applikation finden Sie in der Datei `MyGUIComponent.java`.

1. Welche Ausgaben liefert dieses Beispielsystem?

2. Bei einem Update der OOP-Bibliothek wird die Klasse `Object` durch eine neue Version ersetzt. Dieses Update finden Sie auf der Webseite der Veranstaltung unter dem Dateinamen `Aufgabe_7_1_Update.zip`. Welche drei Fehler bzw. Probleme entstehen durch dieses Update im Zusammenspiel zwischen Bibliotheks- und Applikationscode? Erläutern Sie bitte, wie sich das jeweilige Problem auf die Ausführung des Programmes auswirkt.

3. Nach der Identifikation der Probleme, bittet Sie Ihr Vorgesetzter, diese zu beheben. Hinweis: Obwohl Ihnen der Bibliotheks-Quellcode zur Verfügung steht, dürfen Sie ausschließlich Anpassungen an der Klasse `MyGUIComponent` vornehmen. Geben Sie eine korrigierte Version des Applikations-Codes an, in der die von Ihnen entdeckten Probleme (soweit möglich) behoben sind.

4. Gibt es geeignete Maßnahmen, mit denen Sie diese Probleme im Vorfeld gänzlich verhindern oder zumindest frühzeitig (z.B. durch den Compiler) hätten entdecken können? Hinweis: Beachten Sie unter anderem die Möglichkeiten der Standard-Annotationen in Java.

Aufgabe 7.2 ..... 20 Punkte

In dieser und den nächsten drei Aufgaben werden einige Konzepte der funktionalen Programmierung, die in der Vorlesung vorgestellt worden sind, an Hand von Beispielen in der rein-funktionalen Programmiersprache *Haskell* vertiefend betrachtet. Es werden keine Vorkenntnisse in Haskell oder einer anderen funktionalen Sprache vorausgesetzt.

Alle Informationen, die Sie zur Bearbeitung der Aufgaben benötigen, finden Sie im Vorlesungsskript und in einem Haskell-Tutorial, das auf der Veranstaltungs-Homepage als Material zu dieser Übung bereitgestellt ist (`HaskellTutorial.pdf` [1]). Arbeiten Sie zunächst dieses Tutorial durch, bevor Sie die Beantwortung der folgenden Fragen angehen. Wenn Sie die Haskell-Beispiele ausprobieren möchten, installieren Sie den Haskell-Interpreter `Hugs` (`http://www.haskell.org/hugs`) oder den `Glasgow Haskell Compiler` (`http://www.haskell.org/ghc/`).

1. Geben Sie die notwendigen Haskell-Ausdrücke an, um die folgenden Objekte in Haskell zu erzeugen:
   - Eine Liste aus den `Char`-Elementen `B`, `i`, `m`, `f` und `i`.
   - Ein Tupel aus den `Char`-Elementen `B`, `i`, `m`, `f` und `i`.
2. Fügen Sie nun sowohl der Liste als auch dem Tupel das Integer-Element 2012 hinzu. Was stellen Sie hierbei fest und wie erklären Sie Ihre Feststellungen?
3. Schreiben Sie eine Haskell-Funktion `mittelDreier`, die als Tupel-Parameter drei Integer Werte annimmt und als Ergebnis den Mittelwert der drei Integer Werte als Float zurückgibt.
4. Wandeln Sie die Funktion `mittelDreier` in die Curryform um. Sie können diese Umformung entweder von Hand durchführen oder Sie nutzen hierfür die im Modul `Blatt7` bereitgestellte Funktion `curry3`. Sie finden dieses Modul ebenfalls auf der Webseite der Veranstaltung unter dem Dateinamen `Blatt7.hs`. Egal wie Sie vorgehen, geben Sie in jedem Fall die Signatur Ihrer Funktion in der Curryform an.
5. Implementieren Sie die Funktion `mittelListe`, welche für eine Liste von Elementen des Typs Integer den Mittelwert (als Float) der Elemente berechnet.
   - Überlegen Sie, wie Sie die ggf. notwendigen Hilfsvariablen hinzufügen können.
   - Übergeben Sie die Initialwerte der Hilfsvariablen direkt beim Aufruf der Funktion
     oder
   - Führen Sie eine Hilfsfunktion ein, die lediglich die Liste als Parameter benötigt und dann diesen Aufruf an die eigentliche Berechnungsfunktion mit den Initialwerten durchreicht.

Hinweise:

- Für die eventuell notwendigen Umwandlungen zwischen Integer und Float nutzen Sie bitte die Funktion `fromIntegral` (`http://www.haskell.org/haskellwiki/Converting_numbers`)

[1] Dieses Dokument wird leider nicht von allen PDF-Viewern korrekt angezeigt. Achten Sie bitte darauf, dass auf der ersten Seite der Vermerk "korrigierte Version: Torsten Görg 29. Januar 2012" steht, dann sollten alle Korrekturen sichtbar sein.

- Um das Hilfsmodul einzubinden, nutzen Sie als erste Zeile Ihres Quellcodes den Befehl `import Blatt7`. Die Datei `Blatt7.hs` muss sich im gleichen Verzeichnis befinden, wie Ihr Quellcode.

Aufgabe 7.3 ..... 10 Punkte

Implementieren Sie eine Haskell-Funktion, die eine Liste von Strings als Parameter entgegennimmt und eine reduzierte Liste von Strings zurückliefert, die alle Strings der als Parameter übergebenen String-Liste umfasst, die das Zeichen 'n' enthalten. Diese Funktion habe die Signatur:

```
stringListFilter :: [String] -> [String]
```

Z.B. soll der Aufruf

```
stringListFilter ["Bernd", "Anne", "Saskia", "Nadine", "Michael"]
```

die Liste `["Bernd", "Anne", "Nadine"]` liefern.

Hinweise:

- Strings sind in Haskell Listen von `Char`-Elementen.
- Verwenden Sie in Ihrer Implementierung die im Tutorial beschriebene Funktion höherer Ordnung `listeFilter`.
- Zur Realisierung der Funktionalität dürfen Sie natürlich zusätzliche Hilfsfunktionen hinzufügen.

Aufgabe 7.4 ..... 11 Punkte

In der Aufgabe 1.4 vom Aufgabenblatt 1 hatten wir den Sortieralgorithmus Insertion-Sort betrachtet. Hier ist nochmal eine Ada-Implementierung des Algorithmus, in der die Arrays durch Listen ausgetauscht sind und die dem funktionalen Paradigma genügt (entsprechend Aufgabenteil 1.4.3):

```
function Sort( Data : Int_List ) return Int_List is
   function Insert( Data : Int_List; Element : Integer ) return Int_List is
   begin
      if Data = null or else Element <= Data.Element then
          return new Int_List_Element'( Element, Data );
      else
          return new Int_List_Element'( Data.Element,
                                        Insert( Data.Next, Element ) );
      end if;
   end Insert;
begin
   if Data = null then return null; end if;
   return Insert( Sort( Data.Next ), Data.Element );
end Sort;
```

Erstellen Sie zu dieser Version des Algorithmus eine äquivalente Haskell-Implementierung. Geben Sie dabei auch die Signaturen aller Haskell-Funktionen an.

Aufgabe 7.5 ..... 18 Punkte

Gegeben seien folgende Haskell-Datentyp-Definitionen:

```
data Color = Blue | Green | Orange | Pink | Red

data Tree = Nil | Node Color [Tree]
```

Mit diesen Typen lassen sich Bäume konstruieren, deren Knoten (erstellbar mit dem Konstruktor `Node`) jeweils eine Farbe speichern (Komponententyp `Color`) und beliebig viele Unterbäume aufspannen (Listenkomponententyp `[Tree]`). Ein einfacher, kinderloser Knoten ist z.B. erstellbar mit dem Ausdruck `(Node Orange [])`.

1. Geben Sie einen Beispiel-Ausdruck an, der einen Baum mit eine Höhe von mindestens 3 erstellt.

2. Implementieren Sie eine Haskell-Funktion `treeWeight`, die das Gesamtgewicht eines Baums vom Typ `Tree` berechnet. Geben Sie auch die Signatur dieser Funktion an.

   - Das Gewicht eines Baumes sei das Gewicht seines Wurzelknotens zuzüglich des Drittels der Summe der Gewichte aller Unterbäume.
   - Das Gewicht eines Knotens ist abhängig von der in im gespeicherten Farbe: Ein `Blue`-Knoten hat das Gewicht 6, ein `Green`-Knoten das Gewicht 2, ein `Orange`-Knoten das Gewicht 14, ein `Pink`-Knoten das Gewicht 9 und ein `Red`-Knoten das Gewicht 123.
   - Ein leerer Baum hat das Gewicht 0.

3. Verallgemeinern Sie nun die Datentyp-Definitionen und die Funktion `treeWeight` aus dem vorigen Aufgabenteil so, dass die Datenkomponente eines Baumknotens nicht auf `Color` festgelegt sondern mit einem beliebigen Typen `a` parametrisierbar ist. Zur Ermittlung des Gewichts eines Knotens ist der Funktion `treeWeight` als zusätzlicher Parameter eine passende Funktion zu übergeben, die die Datenkomponente eines Knotens auf einen Gewichtswert abbildet. Geben Sie auch die modifizierte Signatur von `treeWeight` an. Testen Sie Ihre Implementierung mit folgender an `treeWeight` zu übergebenden Beispielfunktion:

```
nodeWeight :: Char -> Float
nodeWeight 'a' = 11
nodeWeight 'b' = 33
nodeWeight _ = 5
```