Compiler: Blatt 2

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 27.04.2017 09:44

Wir werden auf diesem Blatt eine kleine Verwaltung für eine Videothek beginnen um Haskell besser kennen zu lernen.

Aufgabe 1 — GitHub und GitHub Classroom

Den Starter-Code für dieses Übungsblatt bekommen Sie über GitHub. Öffnen Sie dazu folgenden Link: https://classroom.github.com/assignment-invitations/150d097f340c20d6835999c5e0bd9b60. Klicken Sie auf `Accept this assignment` damit ein privates Repository für Sie erzeugt wird. Es wird außerdem ein Jenkins-Job erzeugt.

Auch wenn Sie die Lösung für Blatt nicht abgeben müssen, können Sie Ihren Lösungsansatz natürlich committen und pushen. Mit jedem Push wird das Projekt auf dem Jenkins gebaut und getestet. Das Repository ist nur für Sie und für mich sichtbar.

Sollten Sie Fragen zu Ihrem Code haben, erzeugen Sie in Ihrem Repository einfach ein Issue und weisen Sie es mir (Benutzername `obcode` auf GitHub) zu.

Das Haskell-Projekt in ihrem Repository ist, ausgehend von dem `new-template` schon ein wenig an die folgende Aufgabenstellung angepasst.

Sie können mit den `stack`-Befehlen die Sie auf Blatt 1 kennen gelernt haben, das Projekt bauen, die simple `main`-Funktion ausführen und die Haddock-Dokumentation erzeugen.

Die `main`-Funktion ist ein einfach CLI mit dem Sie, sobald Sie alles implementiert haben, Filme leihen und zurück geben können.

Sehen Sie sich als erstes den Code von `Main.hs` an und versuchen Sie zu verstehen, was dort alles steht.

Die Signaturen aller im Folgenden beschriebenen Funktionen sowie deren Dokumentation sind bereits vorhanden.

Aufgabe 2 — Filme als Tupel und Listen

Wir wollen unsere Videothek folgendermaßen repräsentieren:

```
type Id         = Int
type Name       = String
type FSK        = Int
type Movie      = (Id, Name, FSK)
type Moviestore = ( [Movie] {- verfügbare Filme-}
                  , [Movie] {- ausgeliehene Filme -})
```

Dies ist bereits in der Library in den zwei Modulen `Movie` und `Moviestore` umgesetzt.

Mit `type` können wir sogenannte Typsynonyme definieren, die den Code lesbarer machen. Wir sagen dem Compiler: Immer wenn Du den Typ `Name` siehst, ersetze das einfach durch den Typ `String`. Das ist speziell für Typen so ähnlich wie die `#define`-Direktiven für den C-Präprozessor.

Ein Film ist also ein Tripel bestehend aus ID, Name und FSK und eine Videothek ist ein Tupel bestehend aus zwei Listen von Filmen. Die erste Liste sind die verfügbaren, also nicht ausgeliehenen, Filme, die zweite Liste die ausgeliehenen.

Anmerkung: Wir können das natürlich auch mit selbst definierten Typen schöner machen, aber irgendwie müssen wir ja mal anfangen.

Eine Beispielvideothek könnte also folgendermaßen aussehen:

```
myMovieStore :: Moviestore
myMovieStore = ( [ (1, "Matrix"                         , 16)
                 , (2, "Alpen - unsere Berge von oben",  0)
                 ]
               , [ (3, "The Breakfast Club"             , 12)
                 ]
               )
```

a) Implementieren Sie eine Funktion

```
showMovie :: Movie -> String
```

die einen Film in eine Zeichenkette umwandelt. Z.B.

```
showMovie (2,"Alpen - unsere Berge von oben",0)
    == "Alpen - unsere Berge von oben (Id: 2, FSK 0)"
```

Also zuerst der Titel und dann in Klammern die ID und die FSK-Freigabe.

Achtung: Nutzen Sie die Funktion `show` zum Umwandeln von `Int`s in `String`s.

b) Implementieren Sie eine Funktion

```
showMovieList :: [Movie] -> String
```

die eine Liste von Filmen in eine Zeichenkette umwandelt, die alle Filme getrennt durch Zeilenumbrüche enthält.

c) Implementieren Sie eine Funktion

```
showMoviestore :: Moviestore -> String
```

die aus einem Moviestore eine Zeichenkette macht und zwar in der Form, dass zuerst alle verfügbaren, dann alle ausgeliehenen Filme angezeigt werden. Im GHCi soll das dann beispielsweise so aussehen:

```
*Main> putStrLn $ showMoviestore myMovieStore
Verfügbare Filme
================
Matrix (Id: 1, FSK 16)
Alpen - unsere Berge von oben (Id: 2, FSK 0)

Ausgeliehene Filme
==================
The Breakfast Club (Id: 3, FSK 12)
```

d) Erweitern Sie die Funktion `showMoviestore` so, dass hinter der jeweiligen Überschrift in Klammern die Anzahl der verfügbaren bzw. ausgeliehenen Filme steht. Der Doppelstrich unter der Überschrift soll dann entsprechend verlängert werden.

Die neue Ausgabe soll also so aussehen:

```
Verfügbare Filme (2)
====================
Matrix (Id: 1, FSK 16)
Alpen - unsere Berge von oben (Id: 2, FSK 0)

Ausgeliehene Filme (1)
======================
The Breakfast Club (Id: 3, FSK 12)
```

Probieren Sie insbesondere auch aus, ob bei 10 oder mehr Filmen in einer Liste ein ausreichend langer Doppelstrich erzeugt wird.

e) Implementieren Sie eine Funktion

```
extract :: Id -> [Movie] -> (Maybe Movie, [Movie])
```

die zu einer gegebenen `Id` den entsprechenden Film aus einer Filmliste entfernt. Wenn der Film nicht in der Liste enthalten ist, soll ein Tupel bestehend aus `Nothing` und der unveränderten Liste zurück gegeben werden. Sonst `Just movie` und die Liste ohne den Film.

`Nothing` und `Just x` sind vom Typ `Maybe a`, wenn `x` den Typ `a` hat. Die Idee von `Maybe` wurde in Java 8 mit der `Optionl`-Klasse aufgegriffen und führt dazu, dass Sie endlich auch in Java besseren Code ganz ohne `null`-Pointer schreiben können!

Also z.B.

```
*Main> extract 0 [(1,"Matrix",16),(2,"Alpen - unsere Berge von oben",0)]
(Nothing,[(1,"Matrix",16),(2,"Alpen - unsere Berge von oben",0)])
*Main> extract 1 [(1,"Matrix",16),(2,"Alpen - unsere Berge von oben",0)]
(Just (1,"Matrix",16),[(2,"Alpen - unsere Berge von oben",0)])
```

f) Implementieren Sie unter Verwendung von `extract` die beiden Funktionen

```
rent :: Age -> Id -> Moviestore -> (Bool, Moviestore)
handBack :: Id -> Moviestore -> Moviestore
```

`Age` ist ein Typsynonym für `Int`.

Mit der Funktion `handBack` können Sie einen Film zurück geben, mit `rent` ausleihen. Natürlich muss der Film dazu in der entsprechenden Liste vorhanden sein und beim Ausleihen muss die FSK-Freigabe zum Alter passen.

Aufgabe 3 — Ein kleines bisschen Testen und Tipps

Selbstverständlich sollen Sie Ihre Implementierungen auch wieder testen. Die beiden Spezifikationen `MovieSpec` und `MoviestoreSpec` sind bereits vorbereitet und enthalten jeweils einen Test.

Erweitern Sie die Tests sinnvoll.

Mit HLint gibt es ein Tool mit dem Sie Tipps zu Ihrem Code bekommen können.

Installieren Sie es mit

```
stack install hlint
```

Anschließend können Sie mit dem Befehl

```
hlint .
```

Tipps bekommen, was Sie verbessern können.