Programmierkurs Julia
Johannes Nägele
Lehrstuhl Prof. Dr. Schlather
FSS 2024

Übungsblatt 6

Aufgabe 1 (5 Punkte)

Diese Aufgabe soll eine grobe Idee dafür vermitteln, wie und warum man für Anwendungsprobleme eigene Typen definiert.

(a) Erstellt euch einen `abstract type` namens `AbstractIdentification` (aka „Ausweis“).

(b) Betrachtet erneut die Aufgabe 3 von Blatt 2. Erstellt einen `struct Perso` (aka „Personalausweis“) mit den Feldern `surname`, `name` und `age`. Dieser soll ein Subtyp von `AbstractIdentification` sein.

(c) Reimplementiert die Funktion `personal_info` für den Inputtyp `Perso`.

(d) Ein Personalausweis soll bei fehlender Altersangabe automatisch mit Alter 0 ausgestellt werden. Schreibe dafür einen äußeren Konstruktor.

(e) Nun wollen wir sicherstellen, dass in den Datenfeldern nur sinnvolle Datentypen verwendet werden. Parametrisiert `Perso` dahingehend, dass `surname` und `name` vom gleichen Typ sein sowie ein Subtyp von `AbstractString` müssen und dass `age` vom Typ `Int` ist (Tipp: Um einen `struct` umzudefinieren, müsst ihr Julia killen. Diese sind nämlich über die *lifetime* eines Programms unveränderlich, ihr bekommt also einen Error `invalid redefinition of type Perso`).

(f) Schreibt euch einen inneren Konstruktor für `Perso` (ihr müsst Julia wiederum killen). Mit diesem wollen wir ermöglichen, dass ein Alter auch als nicht-`Int` eingegeben werden kann (etwa `1.0`). Damit der Endbenutzer keine kryptischen Fehler bei einer nicht-konformen Eingabe bekommt (etwa `1.1`) soll gecheckt werden, ob das Alter nicht-negativ und ganzzahlig ist (Tipp: `Int`, `@assert`, Modulo).

(g) Erstellt euch einen weiteren `struct` namens `StudiID` (aka „Studierendenausweis“). Dieser soll ebenfalls Subtyp von `AbstractIdentification` sein und die Felder `surname`, `name` und `matrikelnummer` haben.

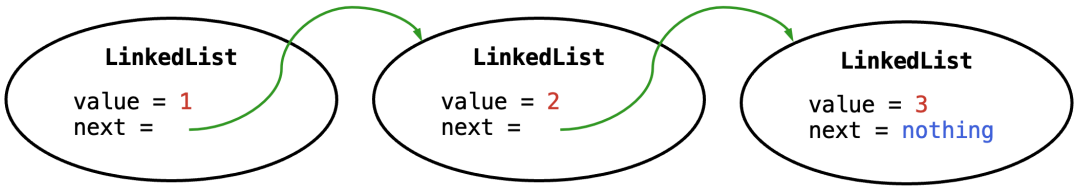
(h) Schreibt euch eine Funktion `name`, die den (Nach-)Namen einer Variablen, die vom (Sub-)Typ `AbstractIdentification` ist, zurückgibt.

(i) Definiert euch noch einen letzten Typ `SpyID`. Dieser soll nur das Feld `country` haben.

(j) Unsere bisherige Funktion `name` funktioniert offensichtlich nicht für unseren neuen Typ. Für Variablen vom Typ `SpyID` soll nun stets `"Mr. X"` zurückgegeben werden (Intuition: Man sagt, `AbstractIdentification` *implementiert* die Funktion bzw. das Interface `name`).

Aufgabe 2 (3 Punkte)

Eine der bekanntesten und primitivsten Datenstrukturen ist die Linked List. Diese funktioniert rekursiv: Die Liste hat einen Startknoten; in diesem können wir einen Wert sowie die Adresse des nächsten Knotens speichern. Das machen wir im nächsten Knoten genauso und solange, bis wir alle Werte abgespeichert haben (siehe Abbildung; wir unterscheiden nicht zwischen Knoten und Liste).



(a) Schreibt einen `mutable struct` namens `LinkedList{T}`, wobei `T` der Typ der abgelegten Werte ist (Tipp: Wir haben zwei Felder, eines davon ist der gespeicherte Wert und das andere ist entweder `nothing` oder die nächste `LinkedList`).

(b) Schreibt eine Funktion `insert!(list, value)`, welche einen Wert `value` ans Ende der Liste hinzufügt.

(c) Schreibt eine Funktion `get(list, i)`, welche uns den `i`-ten Wert unserer Linked List zurückgibt.

Aufgabe 3 (7 Punkte)

Data handling.

(a) Im GitHub-repository unter data findet Ihr den Datensatz `weatherAUS.csv`. Ladet diesen als `DataFrame` in Julia ein und speichert ihn unter einer sinnvoll benannten Variable (Tipp: Stelle sicher, dass fehlende Werte als `missing` interpretiert werden!).

(b) Was beschreibt der Datensatz? Wie viele missings gibt es? Über welchen Zeitraum erstrecken sich die Observationen?

(c) Wir benötigen nur die ersten fünf Spalten sowie die Spalte 22. Extrahiere diese. Wie viele missings gibt es jetzt noch?

(d) Füge eine neue Spalte hinzu, welche die Temperaturschwankung `MaxTemp - MinTemp` berechnet. Wann und wo gab es die größte Schwankung (Tipp: Verwende `argmax`)?

(e) Extrahiere alle Einträge, die in Wollongong gemessen wurden. Wie hoch ist hier die durchschnittliche MinTemp? (Tipp: `skipmissing`; Wollongong sollte mit einem `String` und nicht mit einem `Symbol` verglichen werden.)

(f) Betrachte die Spalte `RainToday`. Wir wollen wissen, ob diese Werte überhaupt Sinn ergeben. Gebt Euch daher das Maximum von `Rainfall` an regnerischen Tagen aus.

(g) Ist die durchschnittliche `MinTemp` an Tagen mit Regen oder ohne Regen höher?

Aufgabe 4 (Zusatzaufgabe, 3 Punkte)

In dieser Aufgabe wollen wir uns eine Funktion `subtypetree` schreiben, mit der wir uns nicht nur direkt folgenden Subtypen, sondern auch Subtypen beliebig viele Ebenen tiefer ausgeben können. Die Ausgabe für `Real` soll dann so aussehen:

```
Real
   AbstractFloat
       BigFloat
       Float16
       Float32
       Float64
   AbstractIrrational
       Irrational
   Integer
       Bool
       Signed
           BigInt
           Int128
           Int16
           Int32
           Int64
           Int8
       Unsigned
           UInt128
           UInt16
           UInt32
           UInt64
           UInt8
   Rational
```

Tipp: Verwende Rekursion; wir brauchen mindestens zwei Inputs: Erstens den entsprechenden Typ sowie zweitens ein Argument, dass uns angibt, wie viele Ebenen wir den Baum bereits runtergelaufen sind.