Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Lehrstuhl Softwaretechnik und Programmiersprachen
Philipp Körner

18. Februar 2022

Klausur

Einführung in die Funktionale Programmierung
Wintersemester 2021/22

Nachname: _____ Vorname: _____

Matrikelnummer: _____

Zugelassene Hilfsmittel: verkürztes Clojure Cheat Sheet
Dauer: 60 Minuten

Diese Klaujur enthält 8 nummerierte Seiten. Prüfen Sie bitte zuerst, ob alle Seiten vorhanden sind. Mit Ihrer Unterschrift versichern Sie, eine vollständige Klausur erhalten zu haben.

Unterschrift: _____

Schalten Sie bitte jegliche elektronischen Geräte aus

Ich habe an den meisten Terminen der Lehrveranstaltung
- ☐ in Präsenz
- ☐ online
- ☐ nicht

teilgenommen. (Freiwillige Angabe)

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	Σ
Punktzahl	12	10	10	11	7	50
Erreicht						

Aufgabe 1 [12 Punkte]

Eine Multimenge ist eine unsortierte Datenstruktur ähnlich zu einem Set, jedoch können Elemente mehrfach vorkommen. Daher ist eine mögliche Darstellung der Multimenge $\{1, 1, 1, 2, 3, 3, 4\}$ eine Clojure Map `{1 3, 2 1, 3 2, 4 1}`, die ihre Elemente auf die entsprechende Häufigkeit abbildet.

(a) [3 Punkte] Implementieren Sie eine Funktion `create-mm`, die eine Multimenge aus einem Seqable erstellt. Es gibt keine Restriktionen aus den `clojure.core` Funktionen. Ggf. kann Ihnen auch `conjoin` aus Teilaufgabe (b) weiterhelfen.

Beispielaufruf:

```
user=> (create-mm [1 1 1 2 3 3 4])
{1 3, 2 1, 3 2, 4 1}
```

(b) [6 Punkte] Implementieren Sie eine Funktion `conjoin`, die zu einer Multimenge ein Element hinzufügt.

Beispielaufrufe:

```
user=> (conjoin {1 3, 2 1} 1)
{1 4, 2 1}
user=> (conjoin {1 3, 2 1} 3)
{1 3, 2 1, 3 1}
```

(c) [3 Punkte] Gegeben sei das folgende Protokoll:

```
(defprotocol Countable
  (cnt [c]))
```

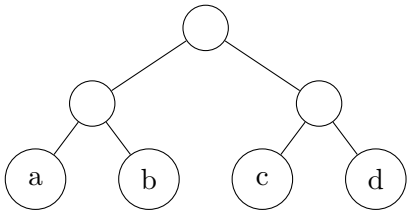
Implementieren Sie das `Countable` Protocol für Multimengen. Dabei sollen von der Funktion `cnt` alle Elemente *inklusive* aller Duplikate gezählt werden. Für alle anderen Maps ist das Verhalten undefiniert.

Beispielaufruf:

```
user=> (cnt {1 3, 2 1})
4
```

Aufgabe 2 [10 Punkte]

Betrachten Sie den folgenden Binärbaum, der immutable ist:



(a) [4 Punkte] Markieren Sie die minimale Menge an Knoten, die kopiert bzw. neu angelegt werden müssen, um einen Binärbaum zu erstellen, der anstatt "c" den Wert "foo" speichert. Zeichnen Sie diesen Binärbaum auch ein.

(b) [2 Punkte] Warum ist es wichtig, dass der Baum immutable ist? Was müsste man tun, wenn er nicht immutable wäre?

(c) [4 Punkte] Wie sind Listen in Clojure implementiert? Zeigen Sie an einem Beispiel, in dem ein Element eingefügt wird, wie Struktur geteilt wird.

Aufgabe 3 [10 Punkte]

In einem nebenläufigen Medienwiedergabeprogramm sollen Informationen zum aktuellen Musikstück in zwei Identitäten gespeichert werden: der Titel (`current-track`) und der Künstler (`current-artist`). Dazu ist der folgende unvollständige Codeschnippsel gegeben:

```
(def current-track (... "Derefs Don't Lie"))
(def current-artist (... "Shajira"))
```

Beide Identitäten sollen nur zusammen verändert werden, sollen aber einzeln *oder* zusammen ausgelesen werden können.

(a) [4 Punkte] Welche Konstrukte oder welche Kombination an Konstrukten eignen/eignet sich für `current-track` und `current-artist`? Begründen Sie Ihre Antwort und begründen Sie insbesondere auch, warum Sie andere Konstrukte oder Kombinationen ausschließen.

(b) [3 Punkte] Implementieren Sie eine Funktion `(next-track track artist)`, die das aktuelle Musikstück verändert.

(c) [3 Punkte] Implementieren Sie eine Funktion `(current-song)`, die das aktuelle Musikstück ausliest und den Titel gemeinsam mit dem Interpreten wie im Beispiel formatiert ausgibt. Verwenden Sie die minimale Menge an Funktionsaufrufen, die mit den Identitäten interagieren (die auf dem Cheatsheet unter "Concurrency" gelistet sind).

Beispielaufruf:

```
user=> (current-song)
"Derefs Don't Lie - Shajira"
```

Aufgabe 4 [11 Punkte]

(a) [6 Punkte] Welche zwei Arten Komplexität ("types of complexity") definieren Moseley und Marks in ihrem Artikel "Out of the Tar Pit"? Benennen Sie diese und erklären Sie *kurz*, wie sie sich unterscheiden. Geben Sie auch jeweils ein Beispiel.

(b) [5 Punkte] Definieren Sie die Begriffe *simple*, *easy*, *hard* und *complex*. Erläutern Sie insbesondere, inwiefern sich *simple* und *easy* unterscheiden.

Aufgabe 5 [7 Punkte]

Schreiben Sie ein Macro `(defmacro assert [x])`, das einen beliebigen Ausdruck nimmt. Evaluiert der Ausdruck gegen einen truthy Wert, soll nichts passieren. Evaluiert der Ausdruck gegen einen falsey Wert, soll zur Laufzeit ein AssertionError *mit dem (originalen) Ausdruck im Fehlerstring* geworfen werden. Dafür dürfen Sie (eine Version) des folgenden Codeschnippsel verwenden:

```
user=> (throw (AssertionError. (pr-str [1 2 3])))
Execution error (AssertionError) at ...   ;; eine Exception wurde geworfen
[1 2 3]   ;; Fehlerstring
```

Das `assert` aus clojure.core darf *nicht* verwendet werden.

Beispielaufrufe:

```
user=> (assert 42)
nil
user=> (assert false)
Execution error (AssertionError) at user/eval2125 (REPL:1).
false  ;; Fehlerstring
user=> (assert (get {} :foo))
Execution error (AssertionError) at user/eval2127 (REPL:1).
(get {} :foo) ;; Fehlerstring
```

Geben Sie auch an, wogegen der Aufruf `(assert false)` expandieren soll.