Übungen Funktionale Programmierung (in Clojure) Serie 10

1. Konten in Clojure

(a) Schreiben Sie eine Funktion `(make-account name no)`, die ein Konto für den Namen `name` und die Kontonummer `no` erzeugt.

Das Konto soll so konstruiert sein, dass sichergestellt wird, dass der Saldo des Kontos niemals negativ wird.

(b) Schreiben Sie eine Funktion `(deposit account amount)`, die auf dem Konto `acount` den Betrag `amount` zubucht, sowie eine Funktion `(withdraw account amount)`, die ihn abbucht.

(c) Schreiben Sie eine Funktion `(balance account)`, die den Saldo des Kontos ausgibt.

(d) Schreiben Sie eine Funktion `(transfer acc1 acc2 amount)`, die `amount` von Konto `acc1` auf `acc2` überweist.

2. Eine Bank in Clojure

(a) Repräsentieren Sie die Konten einer Bank in einer geeigneten Datenstruktur in Clojure. Dabei sollen die Kontonummern eindeutig sein.

(b) Schreiben Sie eine Funktion `(add-account bank account)`, die dem Bestand der Konten der Bank ein weiteres Konto hinzufügt. Beachten Sie die Eindeutigkeit der Kontonummer.

(c) Schreiben Sie eine Funktion `(bank-transfer bank no1 no2 amount)`, die in der Bank den Betrag `amount` vom Konto mit der Nummer `no1` auf das mit Kontonummer `no2` überweist.

(d) Schreiben Sie eine Funktion `(deposits bank)`, die die Summe der Guthaben bei der Bank ermittelt.

3. Konkurrierende Zugriffe in Java

(a) Konstruieren Sie Beispiele der Verwendung folgenden Java-Codes, die zeigen, dass man es so nicht machen sollte. Das Beispiel stammt aus dem Buch „Java Concurrency in Practice“ von Brian Goetz et al.

```
/**
 * UnsafeSequence
 *
 * @author Brian Goetz and Tim Peierls
 */

@NotThreadSafe
public class UnsafeSequence {
    private int value;

    /**
     * Returns a unique value.
     */
    public int getNext() {
```

```
            return value++;
        }
    }
```

(b) Schreiben Sie entsprechenden Code in Clojure und überprüfen Sie Ihre Szenarien aus (a) damit.