

Aufgabe 1 (4 Punkte)

Entwickeln Sie eine Klasse `PoorLock` mit Methoden `trylock` und `unlock`, wie sie in `java.util.concurrent.locks.Lock` spezifiziert sind! (*Nur* diese beiden Methoden!) Kann man mit diesen Methoden den wechselseitigen Ausschluss kritischer Abschnitte realisieren?

Aufgabe 2 (4 Punkte)

In 3.1.6, Seite 12, wurde darauf hingewiesen, dass *konservatives* 2-Phasen-Sperren nicht fair ist. Geben Sie ein entsprechendes Beispiel-Szenario an, z.B. unter Verwendung von `deposit`, `withdraw`, `transfer`! (Zeitschnitte verwenden!)

Aufgabe 3 (18 Punkte)

In 3.1.7, Seite 21-22, wird eine Implementierung von geordneten Mengen vorgeschlagen, bei der die Mengenobjekte ohne Schmutzeffekte *nebenläufig* benutzbar sind (obwohl keine vollständige Serialisierbarkeit gegeben ist).

a) (4 P.) Geben Sie eine natürlichsprachliche – aber präzise! – Spezifikation an, die auch Operationen `remove` und `contains` umfasst. Die Spezifikation soll sich von der in der Vorlesung angegebenen auch dadurch unterscheiden, dass die Kardinalität der in einem Objekt enthaltenen Menge durch einen bei der Initialisierung anzugebenden Wert beschränkt ist.

b) (9 P.) Entwickeln Sie eine Implementierung, bei der die Daten in einem *sortierten Feld* gehalten werden. Gehen Sie mit Synchronisationsmaßnahmen sparsam um! (Auf keinen Fall alle Methoden `synchronized` machen!)

[Zur Erinnerung: `T[] array = (T[])new Object[n];`]

c) (5 P.) Testen Sie ihre Implementierung unter Einsatz von $n > 2$ Fäden und *erläutern Sie Ihre Teststrategie!*