1. Simulation von Mensakassen (Synchronisation über Semaphore)

Es soll ein Programm zur Simulation von mehreren Mensakassen entwickelt werden. Wenn ein Student (Kunden sind zur Vereinfachung ausschließlich Studenten) bezahlen möchte, geht er zu der Kasse, an der die wenigsten Studenten warten (falls es Warteschlangen gibt). Wenn er an der Reihe ist, bezahlt er. Während des Bezahlens ist die Kasse für ihn in exklusivem Zugriff, d.h. für andere Studenten gesperrt. Nachdem er bezahlt hat, isst er (Dauer: Zufallszeit) und kommt irgendwann wieder (ebenfalls nach einer Zufallszeit).

1.1. Schreiben Sie ein JAVA-Programm, welches eine Mensa mit beliebig vielen Kassen sowie den Zugang der Studenten mit Hilfe der Klassen `Semaphore` bzw. `ReentrantLock` simuliert. Benutzen Sie hierfür ***unbedingt*** die passenden Standard-Klassen aus dem Package `java.util.concurrent`. Jeder Student soll dabei als eigener Thread modelliert werden. Die Simulation soll nach einer vorgegebenen Zeit durch den Aufruf der Methode `interrupt()` beendet werden.

1.2. Testen Sie Ihr Programm mit 10 Studenten-Threads und 1, 2 und 3 Kassen!

Tipps:

- *Geben Sie jedem Studenten-Thread bei der Erzeugung Informationen über alle Kassen (z.B. in Form einer Kassenliste), damit er seine Auswahlentscheidung treffen kann.*
- *Denken Sie daran, dass alle Zugriffe auf Objekte, die von mehreren Threads verändert werden können, synchronisiert werden müssen!*

2 Smoker-Problem (Synchronisation über JAVA-Objektmonitor / Conditions)

Ein Problem aus einer langsam aussterbenden Welt: Wir betrachten ein System mit drei Rauchern („smoker“) und einem Agenten. Jeder Raucher rollt sich so oft wie möglich eine Zigarette und raucht diese anschließend. Um eine Zigarette rollen und rauchen zu können benötigt ein Raucher drei Zutaten: Tabak („tobacco“), Papier („paper“) und Streichhölzer („matches“). Einer der drei Raucher hat Tabak, ein anderer Papier und der Dritte hat Streichhölzer. Der Agent hat eine unendliche Menge von allen drei Dingen. Alle sitzen um einen Tisch versammelt.
Das Verhalten in diesem System ist folgendermaßen:

1. Der Agent legt zwei der drei Zutaten auf den Tisch (zufällig ausgewählt).
2. Der Raucher, der die fehlende dritte Zutat besitzt, nimmt die beiden Zutaten vom Tisch, rollt sich eine Zigarette und raucht sie anschließend (das kostet Zeit!). Wenn er mit dem Rauchen fertig ist, signalisiert er dies dem Agenten.
3. Der Agent beginnt wieder einen neuen Zyklus → Schritt 1.

2.1. Schreiben Sie ein JAVA-Programm, welches das o.g. Systemverhalten mit einem Agenten-Thread und drei Smoker-Threads simuliert! Benutzen Sie die JAVA-Synchronisationsmechanismen (Eintritt in einen Objekt-Monitor mittels „synchronize“) und die Methoden zur Threadsynchronisation (`wait(), notify(), notifyAll()`). Die Simulation soll nach einer vorgegebenen Zeit durch den Aufruf der Methode `interrupt()` beendet werden.
Testen Sie ihr Programm und dokumentieren Sie die erfolgreiche Simulation durch geeignete Testausgaben!

2.2. Erzeugen Sie eine weitere Programmversion, die für die Synchronisation ausschließlich die im JAVA-Package `java.util.concurrent.locks` zur Verfügung gestellten Mechanismen (`locks` und `conditions`) mit mehreren ConditionQueues benutzt.

Tipps:

- *Versuchen Sie, die Problemstellung auf eine aus der Vorlesung bekannte Synchronisations-aufgabe zurückzuführen (als Spezialfall)!*
- *Orientieren Sie sich am Beispiel-Code aus der Vorlesung!*