Fortgeschrittene Programmiertechniken

Prof. Dr. J. Pauli, Dipl.-Inform. J. Hoefinghoff, J. Kapitza, M. Cherubim

Aufgabe 1

Package im Projekt erstellen

Die Lösung dieser Aufgabe wird nicht direkt in die graphische Oberfläche des Hardware Shops eingebunden. Trotzdem soll sie im gleichen Projekt mit abgegeben werden. Erstellen Sie daher bitte in ihrem Projekt ein neues Package mit dem Titel *problem4* und implementieren Sie alle Klassen, die zur Lösung dieser Aufgabe nötig sind, in dieses Package.

Aufgabe 2

Threads

Um einen kleinen Einstieg in das Thema Threads zu erhalten und später einige Probleme und Aufgaben lösen zu können, sollen Sie als Erstes ein kleines Kassenproblem lösen. Dabei haben Sie zwei Klassen, die das Interface *Runnable* implementieren und durch einen Thread gestartet werden. Die erste Klasse *Acquisition* kümmert sich um die Beschaffung von Kunden, die dann in einer *WaitingQueue* (Warteschlange) eingereiht werden. Die zweite Klasse *Cashpoint* (Kasse) muss nun diese Warteschlange abarbeiten. Dabei sollen die folgenden Anforderungen erfüllt werden.

a) Die Abarbeitung eines Kunden in der Warteschlange dauert zufällig zwischen 6 und 10 Sekunden.

b) Eine Kasse schließt, wenn keine Person mehr in der Warteschlange ist.

c) Es soll unterschiedlich lange dauern einen neuen Kunden zu akquirieren - zufällig von 0 bis maximal zwei Sekunden. Der neue Kunde wird anschließend in die Warteschlange der Kasse eingereiht.

d) Die Akquise der Kunden endet, wenn sich in der Warteschlange mehr als 8 Personen aufhalten und wird auch nicht wieder gestartet. Dadurch läuft das Programm in einen Endzustand.

Implementieren Sie die beschriebenen Verhaltensweisen für die beiden Klassen.

Aufgabe 3

Kritischer Bereich

Sie sollten nun bereits wissen, wie Sie einen Thread starten, wie dieser implementiert ist und dass der laufende Thread auch terminieren kann. Erweitern Sie nun Ihre Implementierung aus Aufgabe 2 so, dass zusätzlich die folgenden Punkte erfüllt werden.

a) Immer, wenn 6 Kunden in einer Warteschlange warten, soll eine neue Kasse aufgemacht werden.

b) Es können maximal 6 Kassen aufgemacht werden. Sind sechs Kassen offen, wird keine neue Kasse geöffnet.

c) Wenn nicht alle Kassen offen sind, wird die Kasse mit der niedrigsten Nummer geöffnet. Beispiel: Wenn Kasse 1 und 2 offen sind, wird als nächstes Kasse 3 und nicht Kasse 5 geöffnet.

d) Das Öffnen einer Kasse dauert 6 Sekunden. Es können sich aber schon Kunden an dieser Kasse anstellen.

e) Ein akquirierter Kunde wird immer in die Warteschlange einer offenen Kasse eingereiht, an der die wenigsten Kunden anstehen.

f) Die Kundenakquise endet, sobald an einer Kasse 8 Kunden eingereiht sind.

g) Während der Abarbeitung eines Kunden wird eine Bilanz geschrieben. Erstellen Sie dazu eine Klasse *Balance*, die für jede Kasse eine ID und den Umsatz speichert. Alle Kassen sollen dieselbe Bilanz nutzen. Beachten Sie hierbei, dass ein kritischer Bereich entsteht.

h) Jeder Kunde kauft mindestens einen Artikel und erhöht somit innerhalb der Bilanz den Umsatz der Kasse an der er bezahlt hat.

i) Wie sich der Umsatz bei der Abarbeitung der Warteschlange zusammensetzt, ist Ihnen überlassen. Für diese Teilaufgabe könnte das Aufsummieren der Preise von zufällig gewählten Produkten oder einfach ein zufälliger double Wert $Zahl > 0$ pro Kunde genutzt werden.

j) Nachdem sich die Bilanz geändert hat, wird die Liste neu sortiert, sodass immer die Kasse an Position 0 steht, die den höchsten Umsatz hat.

k) Bitte achten Sie darauf, dass kein Kunde an einer Kasse steht, wenn das Geschäft geschlossen bzw. ihr Programm terminiert ist.

l) Implementieren Sie Ihr Programm so, dass alle Schritte nachvollziehbar in der Console ausgegeben werden. Ausgegeben werden soll

- Akquise und Abarbeitung eines Kunden mit ID der Kasse und Anzahl der wartenden Kunden
- Öffnen und Schließen einer Kasse
- Starten und Enden der Kundenakquise
- die sortierte Bilanz nach jedem Abarbeiten eines Kunden

Implementieren Sie die beschriebenen Verhaltensweisen. Machen Sie sich einen Plan bzw. Notizen zu allen möglichen kritischen Bereichen und verhindern Sie die resultierenden Probleme durch geeignete Programmierung mit Synchronisation oder geeigneten Klassen aus dem Paket java.util.concurrent.

4

Hinweise

Threads können nicht mehrfach verwendet werden. Bei Bedarf kann ein Thread-Pool verwendet werden.
Für die Erzeugung von Zufallszahlen nutzen Sie die Klasse Random.
Die Grafik zeigt, dass alle Threads (Kassen) dieselbe Bilanz haben. [Aufgabe 2]

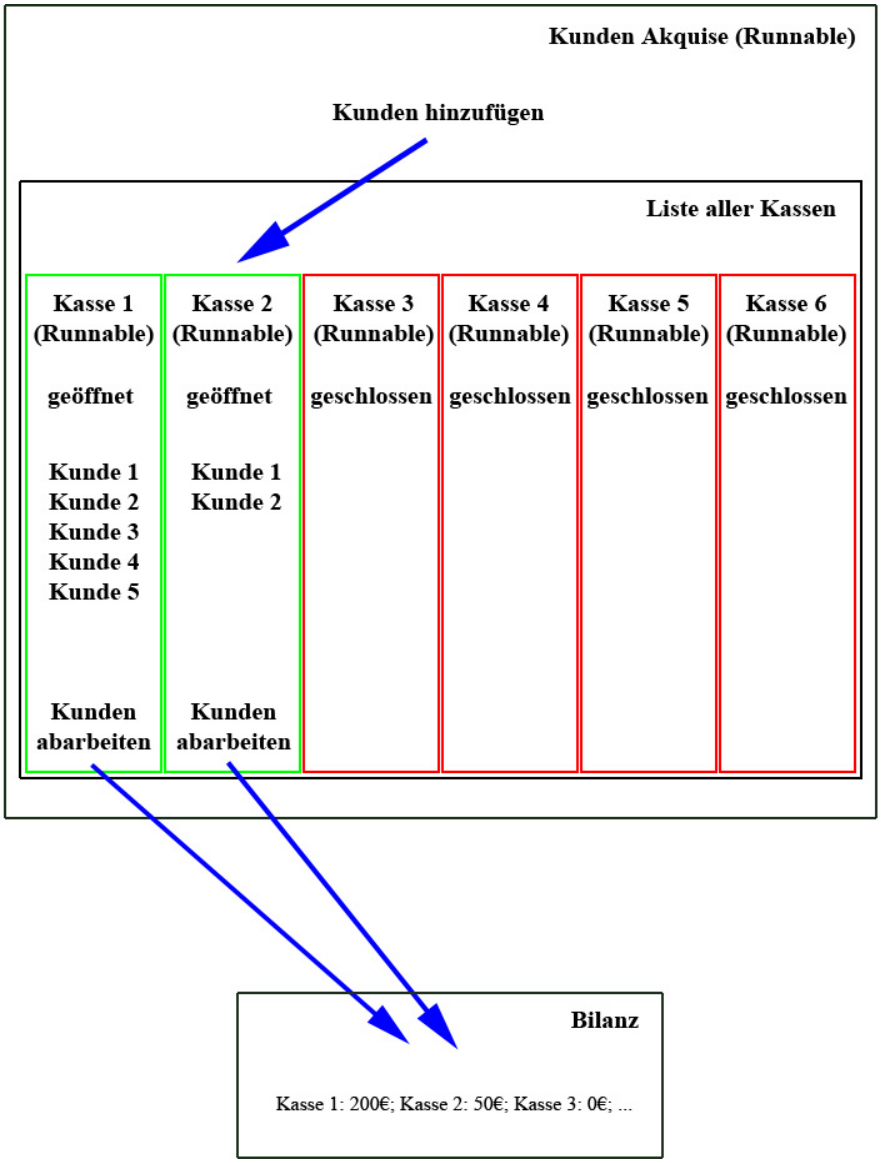

Abbildung 1: Aufbau des Programms