Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Betriebssysteme im Wintersemester 2019/2020
Übungsblatt 6

Abgabetermin: 02.12.2019, 18:00 Uhr

Besprechung: Besprechung der T-Aufgaben in den Tutorien vom 25. – 29. November 2019
Besprechung der H-Aufgaben in den Tutorien vom 02. – 06. Dezember 2019

Aufgabe 28: (T) Grundlagen von Threads

(– Pkt.)

a. Nennen Sie zwei Gründe, warum es nicht sinnvoll ist, zu viele Threads zu verwenden.

b. Nennen Sie zwei Gründe, warum Threads sinnvoll/wichtig sind.

Aufgabe 29: (T) Threads in Java

(– Pkt.)

Laden Sie das Java-Programm `SimpleThread.java` von der Betriebssysteme-Homepage herunter.

a. Betrachten Sie das Java-Programm! Als Eingabeparameter verlangt es eine Integer-Zahl. Wie hängt diese Zahl mit der Ausgabe zusammen? Welche Art der Ausgabe erwarten Sie für verschiedene Integer-Eingaben (zum Beispiel 1, 2, 100 oder 10000)?

b. Basierend auf dem Java-Programm aus Teilaufgabe a:
Geben Sie in Abhängigkeit von `c` eine allgemeine Formel für die Anzahl der möglichen Konsolenausgaben an.

Aufgabe 30: (T) Java: Koordination von Threads

(– Pkt.)

In dieser Aufgabe sollen Sie eine Lösung implementieren, die es ermöglicht, Züge koordiniert über einen **eingleisigen** Streckenabschnitt (AB) fahren zu lassen. Der Streckenabschnitt AB unterliegt folgenden Einschränkungen:

- Der Streckenabschnitt AB verfügt über genau ein Gleis, d.h. es kann gleichzeitig nur in genau eine Richtung gefahren werden (entweder West oder Ost).
- Es können sich maximal drei Züge gleichzeitig auf dem Streckenabschnitt befinden.
- Jeder Zug verlässt den Streckenabschnitt nach endlicher Zeit.

Die Klassen `TrainNet` und `Train` sind bereits gegeben. Sie können sich den Quelltext von der Website zur Vorlesung herunterladen.

Die Klasse `TrainNet` erzeugt einen Streckenabschnitt AB (Instanz der Klasse `RailAB` und startet die Züge (Instanzen der Klasse `Train`).

Die Klasse `Train` repräsentiert Züge und ist als Thread implementiert. Innerhalb der `run()`-Methode werden auf die Instanz der Klasse `RailAB` die Methoden `goEast()` und `goWest()` aufgerufen. Diese dienen dazu, einen Zug auf den Streckenabschnitt von West nach Ost bzw. von Ost nach West zu schicken. Zudem wird die Methode `leaveAB()` aufgerufen, durch deren Aufruf ein Zug den Streckenabschnitt AB wieder verlässt.

Implementieren Sie nun die Klasse `RailAB` unter Berücksichtigung der oben genannten Einschränkungen. Die Lösung muss frei von Deadlocks sein und darf Züge nicht unnötig blockieren. Implementieren Sie

a. einen passenden Konstruktor (siehe Klasse `TrainNet`),

b. die Methode `goWest()`, welche die Züge, die nach Westen fahren, koordiniert,

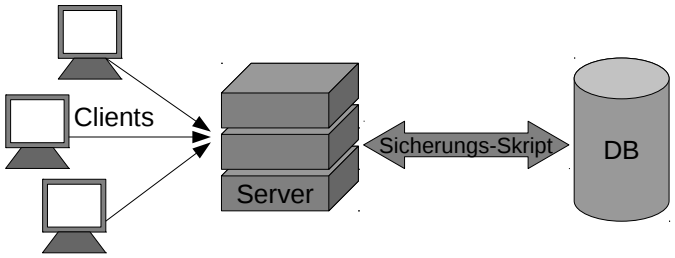
c. die Methode `goEast()`, welche die Züge, die nach Osten fahren, koordiniert, und

d. die Methode `leaveAB()`, welche von den Zügen aufgerufen wird, die den Streckenabschnitt wieder verlassen.

Aufgabe 31: (H) Threads in Java

(11 Pkt.)

In dieser Aufgabe soll eine einfache Client/Server Kommunikation in Java simuliert werden, bei welcher mehrere Clients Anfragen an eine Server-Schnittstelle stellen können, um dort Daten abzulegen. Die Daten werden in regelmäßigen Abständen von einem Sicherungs-Skript auf eine Datenbank geschrieben. Das beschriebene Szenario ist in folgender Abbildung schematisch dargestellt:



Aus Performanzgründen erlaubt der Server nur, dass eine maximale Anzahl von `maxClients` gleichzeitig Daten auf dem Server ablegen darf. Eine Client-Anfrage muss also ggf. mit dem Beginn der Datenablegung warten, damit diese Bedingung nicht verletzt wird.
Das Sicherungs-Skript wird regelmäßig aktiviert und speichert die abgelegten Daten zu einem dedizierten Zeitpunkt auf die Datenbank. Dazu meldet das Skript einen Sicherungswunsch am Server an, so dass kein weiterer Client mehr Daten ablegen darf. Anfragen können jedoch weiterhin gestellt werden und Clients, die zum Zeitpunkt des Sicherungswunsches bereits Daten ablegen, können natürlich ihre Aufgabe noch erledigen.
Das aktive Sicherungs-Skript muss solange warten, bis kein Client mehr Daten ablegt. Nach Beenden der Sicherung wird das Skript wieder deaktiviert und damit der Sicherwunsch aufgehoben, so dass die anfragenden Clients wieder Daten ablegen können.
Im Folgenden soll eine Klasse `Server` implementiert werden. Laden Sie sich bitte dazu zunächst von der Betriebssysteme-Homepage die Dateien `Simulation.java, Client.java,`

`Sicherung.java` sowie den Code-Rahmen der Server-Klasse `ServerAbstract.java` herunter. Die Beispielimplementierungen der Klassen `Client`, `Sicherung` und `Simulation` sollen Ihnen verdeutlichen, wie die Klasse `Server` verwendet werden kann.
Bearbeiten Sie nun die folgenden Aufgaben:

a. Implementieren Sie den Konstruktor der Klasse `Server`. Verwenden Sie dabei den gegebenen Code-Rahmen! Die Klassenattribute sind dort bereits deklariert und müssen durch den Konstruktur initialisiert werden.

b. Implementieren Sie die Server-Methode `daten_ablegen(Client c)`, welche die Client Anfrage zum Daten ablegen modelliert, sowie die die Methode `daten_ablegen_beenden()`, welche vom Client aufgerufen wird, nachdem die Daten abgelegt wurden. Beachten Sie dazu die folgenden Randbedingungen:

   – Alle oben genannten Anforderungen müssen beachtet werden.
   – Maximal `maxClients` dürfen gleichzeitig Daten ablegen. Die Variable wird in der Klasse `Simulation` festgelegt.
   – Es darf kein neuer Client mehr Daten ablegen, sobald das Sicherungs-Skript den Sicherungswunsch geäußert hat. Neue Anfragen können aber weiterhin gestellt werden und noch laufende Daten-Ablegungen werden noch vollständig beendet.

   Ergänzen Sie dazu den Code-Rahmen am Ende der Aufgabe.
   Hinweis: Sie können davon ausgehen, dass die Methoden `daten_ablegen(Client c)` bzw. `daten_ablegen_beenden()` immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse `Client`).

c. Implementieren Sie nun die Methoden für das Sicherungs-Skript. Vervollständigen Sie dazu den Code-Rahmen für die Methoden `sicherungAktivieren()` und `sicherungDeaktivieren()` in dem Code-Rahmen am Ende der Aufgabe.
   Hinweis: Sie können davon ausgehen, dass die Methoden `sicherungAktivieren()` und `sicherungDeaktivieren()` immer in einer sinnvollen Reihenfolge aufgerufen werden (siehe Beispielimplementierung der Klasse `Sicherung`).

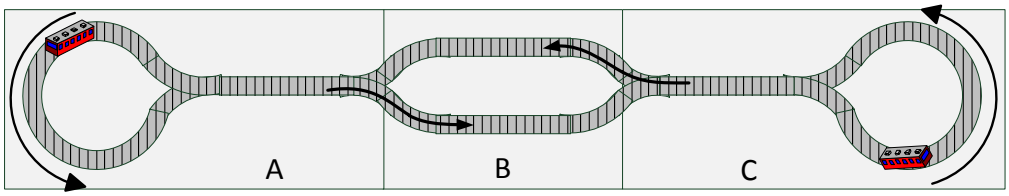
d. Zeigen Sie zwei kritische Bereiche in ihrem Programm auf. Wie wird hier sichergestellt, dass die Bedingung der wechselseitige Ausschluss erfüllt ist?

Aufgabe 32: (H) Petri-Netze

(10 Pkt.)

In dieser Aufgabe wird von folgendem Szenario ausgegangen: Zwei Züge befahren eine gemeinsame Rundbahn (Eingleisig), die sich in drei Gleisabschnitte A, B und C unterteilen lässt. Jeder Zug fährt nur in eine vorgegebene Richtung, weshalb sich jeweils an den Enden der Strecke Wendeschleifen befinden. In der Mitte, im Gleisabschnitt B, besteht die Möglichkeit, dass beide Züge einander passieren, um vom Abschnitt A in Abschnitt C (bzw. umgekehrt) zu gelangen. Aus Sicherheitsgründen dürfen die Abschnitte A und C zu jedem Zeitpunkt nur von einem Zug befahren werden (im Anfangszustand befindet sich je einer der Züge in diesen beiden Abschnitten). Nur im Abschnitt B dürfen sich beide Züge gleichzeitig befinden, wobei der Zug in Richtung Abschnitt C das untere und der Zug in Richtung Abschnitt A das obere Gleis verwenden soll.

Das beschriebene Szenario ist in folgender Abbildung schematisch dargestellt:



Modellieren Sie die Fahrt der beiden Züge durch die Gleisabschnitte anhand eines Petri-Netzes. Gewährleisten Sie bei der Modellierung ebenfalls die Sicherheitsbedingung, dass sich beide Züge nur gleichzeitig in Abschnitt B aufhalten dürfen, nicht aber in A oder C. Verwenden Sie die minimal notwendige Anzahl an Stellen, Marken, Kanten und Transitionen. Versehen Sie alle Stellen und Transitionen **mit aussagekräftigen Bezeichnern**.
Achtung: Verwenden Sie für Ihre Modellierung nur Stellen mit der Kapazität ∞ und Kanten mit dem Gewicht 1.

Aufgabe 33: (H) Einfachauswahlaufgabe: Threads in Java

(5 Pkt.)

Für jede der folgenden Fragen ist eine korrekte Antwort auszuwählen („1 aus n“). Nennen Sie dazu in Ihrer Abgabe explizit die jeweils ausgewählte Antwortnummer ((i), (ii), (iii) oder (iv)). Eine korrekte Antwort ergibt jeweils einen Punkt. Mehrfache Antworten oder eine falsche Antwort werden mit 0 Punkten bewertet.

a) Was ist kein Bestandteil eines Monitors (Softwaremodul)?			
(i) eine oder mehrere Prozeduren	(ii) lokale Daten	(iii) eine Warteschlange für ankommende Prozesse	(iv) eine Liste der Prozesse, die den Monitor verlassen haben
b) Wie viele Prozesse dürfen sich zu jeder Zeit maximal in einem Monitor befinden?			
(i) 0	(ii) 1	(iii) 2	(iv) 3
c) Wie muss die Methode einer von `java.lang.Thread` abgeleiteten Klasse heißen, die den Code beinhaltet, der parallel ausgeführt werden soll?			
(i) *start()*	(ii) *notify()*	(iii) *run()*	(iv) *wait()*
d) Mit welchem Schlüsselwort müssen Eintrittspunkte von Monitoren in Java gekennzeichnet werden?			
(i) `pooled`	(ii) `synchronized`	(iii) `harmonize`	(iv) `locked`
e) Was ist keine Java Direktive, um das Betreten bzw. Verlassen von in Java synchronisierten “Monitor”-Methoden/Blöcken zu organisieren?			
(i) *wait()*	(ii) *notify()*	(iii) *notifyAll()*	(iv) *run()*