**A01**

Nebenläufige Roboterfabrik

Kalauner Paul, Ritter Mathias

**Exposee**

Simulation einer Roboterfabrik mit mehreren Threads in Java

Inhaltsverzeichnis

Aufgabenstellung 3

Zeitabschätzung & Zeitaufzeichnung 5

Requirementsanalyse 6

Benötigte Klassen 6

Lagermitarbeiter 6

Threadee 6

Teil 6

Lager 6

Teiltyp 6

Sekretariat 6

Montagemitarbeiter 6

Lieferant 6

Simulation 6

# Aufgabenstellung

Es soll eine Spielzeugroboter-Fabrik simuliert werden. Die einzelnen Bestandteile des Spielzeugroboters (kurz Threadee) werden in einem Lager gesammelt. Dieses Lager wird als Verzeichnis und die einzelnen Elementtypen werden als Files im Betriebssystem abgebildet. Der Lagermitarbeiter verwaltet regelmäßig den Ein- und Ausgang des Lagers um Anfragen von Montagemitarbeiter und Kunden zu beantworten. Die Anlieferung der Teile erfolgt durch Ändern von Files im Verzeichnis, eine Lagerung fertiger Roboter ebenso.  
  
Ein Spielzeugroboter besteht aus zwei Augen, einem Rumpf, einem Kettenantrieb und zwei Armen.    
Die Lieferanten schreiben ihre Teile ins Lager-File mit zufällig (PRNG?) erstellten Zahlenfeldern. Die Art der gelieferten Teile soll nach einer bestimmten Zeit gewechselt werden.  
  
Die Montagemitarbeiter müssen nun für einen "Threadee" alle entsprechenden Teile anfordern und diese zusammenbauen. Der Vorgang des Zusammenbauens wird durch das Sortieren der einzelnen Ganzzahlenfelder simuliert. Der fertige "Threadee" wird nun mit der Mitarbeiter-ID des Monteurs versehen.  
  
Es ist zu bedenken, dass ein Roboter immer alle Teile benötigt um hergestellt werden zu können. Sollte ein Monteur nicht alle Teile bekommen, muss er die angeforderten Teile wieder zurückgeben um andere Monteure nicht zu blockieren. Fertige "Threadee"s werden zur Auslieferung in das Lager zurück gestellt.  
  
Alle Aktivitäten der Mitarbeiter muss in einem Logfile protokolliert werden. Verwenden Sie dazu Log4J [1].  
  
Die IDs der Mitarbeiter werden in der Fabrik durch das Sekretariat verwaltet. Es dürfen nur eindeutige IDs vergeben werden. Das Sekretariat vergibt auch die eindeutigen Kennungen für die erstellten "Threadee"s.  
  
Beachten Sie beim Einlesen die Möglichkeit der Fehler von Files. Diese Fehler müssen im Log protokolliert werden und entsprechend mit Exceptions abgefangen werden.

**Tipps und Tricks**

Verwenden Sie (optional) für die einzelnen Arbeiter das ExecutorService mit ThreadPools. Achten Sie, dass die Monteure nicht "verhungern". Angeforderte Ressourcen müssen auch sauber wieder freigegeben werden.

**Beispiel für Teile-Files**  
-- "auge.csv"  
Auge,11,24,3,4,25,6,8,8,9,10,11,12,13,14,15,16,17,18,195,5  
Auge,91,62,3,4,54,6,7,8,9,10,11,12,13,14,15,16,17,18,119,32  
Auge,91,62,3,4,54,6,7,8,9,10,11,12,13,14,15,16,17,18,119,520  
  
-- "rumpf.csv"  
Rumpf,91,62,3,4,54,6,7,8,9,10,11,12,13,14,15,16,17,18,119,21

**Beispiel für Threadee-File**  
-- "auslieferung.csv"  
Threadee-ID123,Mitarbeiter-ID231,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Rumpf,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Kettenantrieb,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20  
Threadee-ID124,Mitarbeiter-ID231,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Auge,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Rumpf,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Kettenantrieb,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,Arm,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20  
  
  
**Ausführung**  
  
Zu bedenken sind die im Beispiel angeführten Argumente. Diese können mit eigenem Code oder mit einer CLI-Library implementiert werden (z.B. [2]).  
Alle Argumente sind verpflichtend und die Anzahl muss positiv sein. Die obere Grenze soll sinnvoll festgelegt werden. Vergessen Sie auch nicht auf die Ausgabe der Synopsis bei einer fehlerhaften Eingabe! Sollten Sie zusätzliche Argumente benötigen sind diese erst nach einer Rücksprache implementierter.  
  
java tgm.sew.hit.roboterfabrik.Simulation --lager /verzeichnis/zum/lager --logs /verzeichnis/zum/loggen --lieferanten 12 --monteure 25 --laufzeit 10000

# Zeitabschätzung & Zeitaufzeichnung

Schätzung: 4h/Person, insgesamt 8h

Vorbereitung

|  |  |  |
| --- | --- | --- |
| Aufgabe | Name | Zeitaufwand |
| Requirementsanalyse | Kalauner Paul, Ritter Mathias | 90 Minuten |
| UML-Diagramm | Kalauner Paul, Ritter Mathias | 120 Minuten |
|  |  |  |
|  |  |  |

Implementierung

|  |  |  |
| --- | --- | --- |
| Aufgabe | Name | Zeitaufwand |
| Git und Maven eingerichtet | Kalauner Paul | 15 Minuten |
| Klassen und Methoden erstellt | Ritter Mathias | 15 Minuten |
| Überprüfung der Konsolenargumente | Kalauner Paul | 20 Minuten |
| Lager, Lagermitarbeiter, Lieferant, Sekretariat | Kalauner Paul | 70 Minuten |
| Überprüfung der Teileanzahl | Ritter Mathias | 15 Minuten |
| Einlagern der Teile | Ritter Mathias | 20 Minuten |
| Bereitstellen der Teile | Ritter Mathias | 30 Minuten |
| Auslesen der Files | Kalauner Paul | 30 Minuten |
| Zusammenbauen eines Threadees | Ritter Mathias | 40 Minuten |
| Letzte Zeile aus File löschen Bug Fix, kleines Redesign | Kalauner Paul | 70 Minuten |
| Schreiben in die Files implementiert | Ritter Mathias | 35 Minuten |
| Bugfixing | Ritter Mathias | 20 Minuten |
|  |  |  |

Testen

|  |  |  |
| --- | --- | --- |
| Klassen Lager und Lagermitarbeiter | Ritter Mathias | 65 Minuten |
| Klassen Lieferant, Montagemitarbeiter, Teil, Sekretariat, Threadee | Ritter Mathias | 150 Minuten |
| Simulation | Kalauner Paul | 35 Minuten |
|  |  |  |

Requirementsanalyse

## Benötigte Klassen

* Threadee
* Teil
* Lager
* Teiltyp
* Sekretariat
* Lagermitarbeiter
* Montagemitarbeiter
* Lieferant
* Simulation

### Lagermitarbeiter

Lagert Teile eines Lieferanten ein und stellt Montagemitarbeitern Teile zur Verfügung

Schreibt und liest die Lagerfiles

### Threadee

Enthält zwei Augen, zwei Arme, einen Rumpf und einen Kettenantrieb

### Teil

Enthält die zufällig generierte Zahlenfolge und eine Angabe über den Typ des Teils

### Lager

Enthält die für den Roboter benötigten Teile

### Teiltyp

Gibt an, um welches Teil es sich handelt

### Sekretariat

Generiert die IDs für alle Mitarbeiter und Threadees

Übernimmt die Kommunikation zwischen Lieferant und Lagermitarbeiter

### Montagemitarbeiter

Baut die Threadees zusammen

### Lieferant

Stellt neue Teile zur Verfügung

### Simulation

Startet die Simulation und enthält die main-MethodeKlassendiagramm

Lessons Learned:

**Verwenden von mehreren Threads:**

Mehrere Threads können parallel Programmcode abarbeiten, um das gesamte Programm zu beschleunigen (Vorausgesetz, der Prozessor verfügt über mehrere Kerne).

Threads können auf 2 verschiedene Arten implementiert werden:

Entweder eine Klasse implementiert (implements) die Klasse Thread oder erbt (extends) davon.

Die Verwendung von implements ist sinnvoller, da sonst die Vererbungshirarchie blockiert wird.

Ein Thread wird durch aufrufen von run() gestartet. Er sollte nicht durch ein Interrupt beendet werden, da Interrupts eine Ausnahme darstellen und auch anderswertig ausgelöst werden können. Stattdessen sollte ein Watchdog, der alle Threads überwacht, sie z.B. durch setzen einer boolean-Variable stoppen.

**Synchonisieren von Threads:**

Das synchronized Schlüsselwort stellt sicher, dass immer nur 1 Thread gleichzeitig auf eine Methode oder einen bestimmten Block zugreift.

Der Bereicht, indem sich immer nur ein Thread befinden darf, heißt kritischer Abschnitt.

Man sollte solche Bereiche als kritische Abschnitte definieren, in denen auf eine bestimme Ressource zugegriffen wird. Sonst kann es zu Problemen kommen, wenn z.B. ein Thread etwas aus einem File löscht, während ein anderer liest.