LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Nebenläufigkeit mit Java

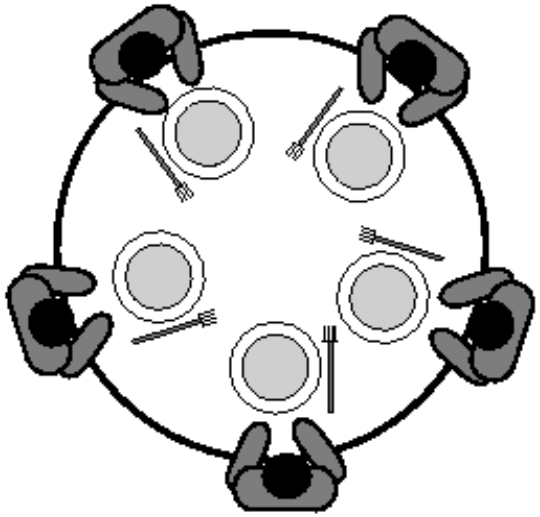
Einheit 04: Übungen

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Das Philosophen-Problem (letzte Stunde)

Implementieren Sie das bekannte Philosophen-Problem in Java (zunächst ohne Deadlock-Vermeidung):

- 5 Philosophen hocken an einem Tisch
- Es gibt genau 5 Gabeln
- Jeder Philosoph soll abwechselnd schlafen und essen.
- Will ein Philosoph essen, so versucht er zunächst seine linke Gabel zu nehmen und dann seine rechte.
  - Wenn das klappt, dann isst er und legt anschließend das Besteck wieder zurück
  - Wenn die Gabel nicht vorhanden ist, dann wartet er auf die Gabel, bis sie wieder greifbar ist.
- Das Philosophen-Problem führt höchstwahrscheinlich irgendwann zu einem Deadlock. Wie könnte dieser vermieden werden?
  - Implementieren Sie nun eine Deadlockfreie Variante

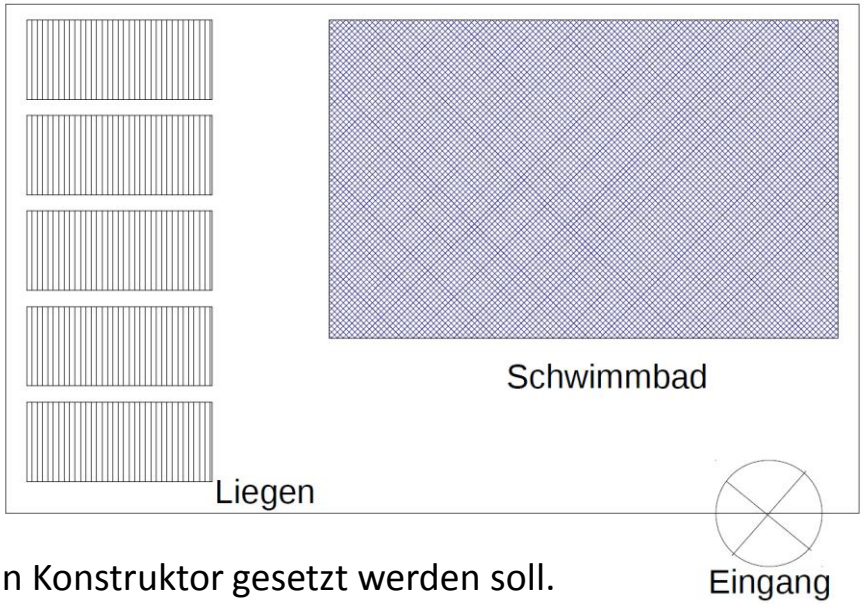


Quelle: https://www.dpunkt.de/java/Programmieren_mit_Java/Multithreading/11.html

Übung 01

Synchronisieren Sie die Badegäste eines kleinen Schwimmbads:

- Badegäste sollen dabei als Java `Threads` realisiert werden.
- Betrachten Sie dazu folgendes Szenario:
  - Das kleine Schwimmbad besitzt maximal `maxLiegen`, die von den Badegästen genutzt werden können.
  - Die Badegäste können das Schwimmbad über ein Drehkreuz betreten und verlassen
  - Zu jedem Zeitpunkt kann immer nur eine Person durch das Drehkreuz gehen.
  - Es dürfen sich stets auch nur maximal so viele Personen im Schwimmbad befinden (inklusive Drehkreuz), wie Liegen vorhanden sind!
- Schreiben Sie für dieses Szenario ein Java-Programm, welches 50 Badegäste bei `maxLiegen=5` den synchronisierten Einlass gewährt, so dass obigen Bedingungen nicht verletzt werden!
- Schreiben Sie folgende Klassen:
  - `Badegast extends Thread`
    - Will Schwimmbad betreten und nach einer zufälligen Zeit wieder verlassen
  - `Simulation`
    - Erzeugt das Schwimmbad, die Badegäste und lässt diese beginnen
    - Beinhaltet Main-Methode
  - `Schwimmbad`
    - Stellt Methode zum betreten bereit
    - Stellt Methode zum verlassen bereit
    - Besitzt eine Variable `maxLiegen`, die über den Konstruktor gesetzt werden soll.



Übung 02

Simulieren Sie folgenden Sachverhalt in Java:

- In einer Uni gibt es einen Hörsaal mit begrenzter Kapazität (`MAX_CAPACITY`).
- Studenten benutzen diesen Hörsaal gerne, um Ihre Hausaufgaben zu machen
  - Es können immer nur so viele Studenten im Hörsaal sein, wie die Kapazität hergibt.
  - Die Studenten müssen als ggf. warten bis der Hörsaal wieder Plätze hat, um ihn betreten zu können.
- Von Zeit zu Zeit kommt Herr Professor Albert in den Hörsaal.
  - Wenn er im Anmarsch ist (man sieht ihn schon von weitem auf dem Flur), dürfen keine Studenten mehr in den Hörsaal gehen.
  - Genau so wenig, wenn Professor Albert selbst im Hörsaal ist.
  - Der Professor selbst ist eitel und betritt den Hörsaal natürlich erst, wenn der letzte Student aus dem Hörsaal gegangen ist, sprich er muss vor dem Betreten warten, bis alle Studenten die sich noch im Hörsaal befinden, rausgegangen sind.
- Ein Teil des Codes ist schon vorgegeben (KursWebseite)
  - Implementieren Sie die abstrakte Klasse `HoersaalAbstrakt` mit den vorgegebenen Methoden unter Verwendung der gegebenen Klassen:
    - `Simulation`, `Professor` und `Student`
  - Achten Sie auf eine genaue Synchronisation der beteiligten Gruppen, so dass obige Bedingungen nicht verletz werden!
  - Erzeugen Sie genug Konsolen-Ausgaben, so dass Sie den Verlauf nachvollziehen können!