Java-Grundlagenschulung
Tag 7

SHD Einzelhandelssoftware GmbH

Agenda

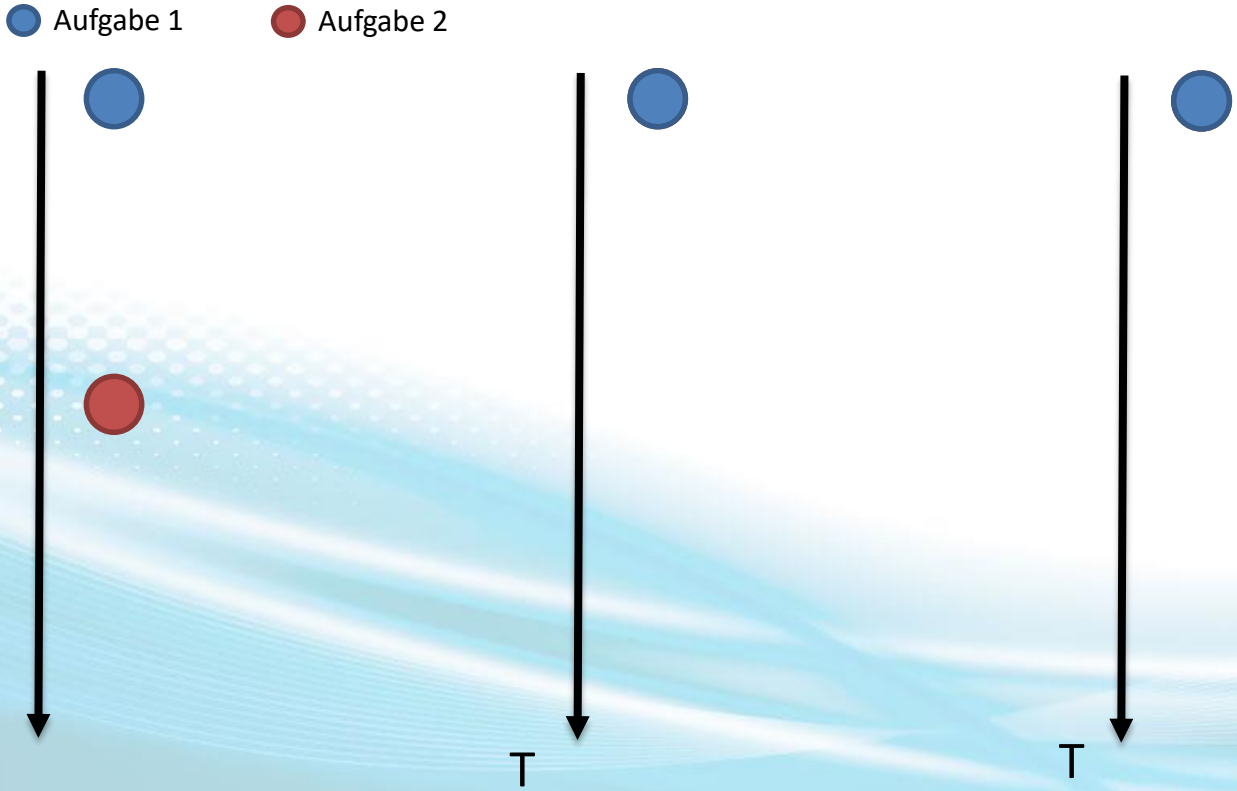
- Threads in Java
- Aufgabe 1 - Einen Thread starten
- Listener in Java
- Aufgabe 2 – ThreadFinishedListener
- Logging mit Log4j
- Aufgabe 3 – Arbeiten mit dem Logger
- Empfehlung
- Hausaufgabe

Threads in Java

- Threads dienen der asynchronen (gleichzeitigen) Verarbeitung
- Normaler Quellcode im Hauptthread läuft weiter
- Runnable Interface zum definieren des asynchronen Codes
- Mit `start()` startet man die `run()` Methode des Threads
- Ziel: Performance Optimierung

```
new Thread(new Runnable()
 {
   @Override
   public void run()
   {
     //Do your Stuff here
   }
 }).start();
```

Threads in Java



Aufgabe 1 - Einen Thread starten

1. Erstelle die Methoden `aufgabe1()` & `aufgabe2()`

2. Füge in `aufgabe1()` ein `Thread.sleep(1000)` und in `aufgabe2()` `Thread.sleep(2000)` ein.

3. Führe beide Methoden aus und stoppe die Laufzeit

4. Lagere die Logik in einzelne Threads aus `new Thread(new Runnable())`

5. Führe beide Methoden aus und stoppe erneut die Laufzeit

Listener in Java

- Helfen bei der Programmierung von Events
- Listener werden mit Interface Klassen definiert
- Üblicherweise meldet man sich bei einem Listener an
- In unserem Beispiel dienen sie uns zum mithorchen wenn ein Thread fertiggestellt wurde

Aufgabe 2 - ThreadFinishedListener

1. Erstelle das Interface `ThreadFinishedListener`
2. Definiere die Methode `void onFinished(String aufgabenName)`
3. Deklariere den Listener als statisches Attribut mit `new ThreadFinishedListener()` und gebe den Aufgabennamen auf der Konsole in der anonymen Ausimplementierung aus.
4. Gebe jeder Aufgaben-Methode über einen Parameter das Listener Objekt mit
5. Führe am Ende jeder Aufgabe die `onFinished()` Listener Methode aus

Logging mit Log4j

- Ein Logger übernimmt die Protokollierung des Programmablaufs
- Der Entwickler muss selbst entscheiden, an welcher Stelle mit welchem Level geloggt werden soll -> jede Klasse sollte sein eigenes Log-Objekt haben.
- Es gibt die folgenden LOG-Level in hierarchischer Reihenfolge:

  OFF -> FATAL -> ERROR -> WARN -> INFO -> DEBUG -> TRACE -> ALL
- Üblicherweise wird der Log in einem separatem log-Verzeichnis in eine separate log-Datei geschrieben
- Die Konfiguration erfolgt über eine Konfigurationsdatei (log4j.properties) im resources Verzeichnis der Java Projekts.
- Nicht Bestandteil des JDK -> externe Lib

```
private static final Logger LOG = Logger.getLogger(Main.class);
```

Aufgabe 3 – Logging mit Log4j

1. Lade dir die log4j.jar aus dem Internet runter
2. Binde die Jar in dein Projekt als externe Lib ein
3. Lege neben den `src` und `test` Order auch einen `resources` Order in deiner Projektstruktur an und markiere diesen in den Projekteinstellungen als solchen.
4. Lege in dem neuen Ordner eine Datei ‚log4j.properties' an und recherchiere im Internet nach dem Inhalt -> Ziel ist es die Logausgaben gleichzeitig auf der Konsole und in einer extra Datei auf dem LogLevel ‚Info' auszugeben.
5. Bau deinen `ThreadFinishListener` dahingehend um, dass er den Logger benutzt und kontrolliere deine Logs.

Empfehlung

1. Vermeidet Threads.
2. Threads killen ist nicht möglich, da kein eigener Prozess vorhanden ist.
3. Racing Conditions möglich -> Synchronisierung empfehlenswert
4. Synchronisierung kann zu „DeadLocks“ führen.

Hausaufgabe

1. Erzeuge eine Klasse Konto mit kontoStand und name als Attribut und den Methoden einzahlen und auszahlen
2. Erzeuge eine Klasse Bankprogramm mit den Methoden
   - geldAbheben(Konto konto, int betrag, String threadName)
   - geldUeberweisen(Konto senderKonto,Konto empfaengerKonto, int betrag, String threadName)
3. Beide Methoden sollen in einem eigenen Thread gestartet werden und dürfen den Kontostand nicht ins negative bringen.
4. Simuliere **nach** der Prüfung ob genug Geld vorhanden ist einen Arbeits/Netzwerkzugriff mit Thread.sleep(x)
5. Protokolliere mit dem Logger jeden einzelnen Vorgang in den Methoden, vor allem die Kontostände vor jeder Aktion und von welchem Thread er kommt.
6. Erzeuge zwei Konten für Hans und Peter mit jeweils 1000€ und lasse Hans gleichzeitig 1000€ abheben und 1000€ an Peter überweisen. Was fällt auf?
7. Informiere dich über das keyword synchronize und löse damit das Problem aus Aufgabe 6 (Tipp: synchronized(konto){})