Referat nebenläufige Programmierung Alex Hepting 1998917

Threads sind Abschnitte bei denen der Programmcode nebenläufig ausgeführt werden soll

- `Thread`: Jeder laufende Thread ist ein Exemplar dieser Klasse.
- `Runnable`: Beschreibt den Programmcode, den die JVM parallel ausführen soll.
- `Lock`: Dient zum Markieren von kritischen Abschnitten, in denen sich nur ein Thread befinden darf.
- `Condition`: Threads können auf die Benachrichtigung anderer Threads warten.

Threads erzeugen

über eine Schnittstelle wird dem Thread mitgeteilt, dass der Code parallel ausgeführt werden soll.

Diese Schnittstelle heißt Runnable und schreibt nur die Methode run() vor.

Parallele Verarbeitung.

Damit ein Thread gestartet wird reicht run() nicht aus. Man startet einen thread mit start().

Diese Methode übergibt den Aufruf an einen Konstruktor, der ein neues Thread aus der Klasse Thread erzeugt.

In der Standardbibliothek von Java ist die Klasse Thread bereits integriert.

Um einfachse Threads zu erzeugen, genügt es die Methode Runnable zu einer beliebigen Klasse mit „implements Runnable“ zu erweitern.

Das was im Thread gestartet werden soll, steht alles in der Methode run(), die zusätzlich zu der Klassenmethoden hinzugefügt wird.

Um den Thread letzendlich zu starten, wird die methode start() aufgerufen, die in der Klasse von Thread enthalten ist.

Diese führt dann die Methode run() im Thread aus. Der Code wird somit nebenläufig ausgeführt.

Der Klou ist, dass run() allein nur eine Methode ist und nicht nebenläufig laufen würde.

Weil der Konstruktor einen neuen Thread erzeugt mit der Methode start(), wird eine so implementierte Methode auch im Thread ausgeführt und läuft somit parallel ab.

Zustände von Threads

Man kann anfangs Threads einen festen Namen zuweisen. `Thread(String name)`

Standardmäßig ist der Name »Thread-x«, wobei x eine eindeutige Nummer ist.

Mit getName() kann man den Namen zurückbekommen. Mit setName() lässt dich der name des Threads ändern.

Läuft ein Thread erst mal, so hat dieser beim Aufruf zusätzliche Methoden, die für den Thread nützlich sind.

Ein Beispiel für einer dieser Methoden ist der Schlafmodus, der den Thread für eine bestimmte Zeit schlafen legen kann.

```
try {
   Thread.sleep( 2000 );
} catch ( InterruptedException e ) { }
```

Die Alternative zu sleep ist die yield Methode, die über die Thread-Verwaltung funktioniert.

Dabei wird der Thread für eine Runde ausgesetzt und wieder eingeschleift, wenn die Warteschlange bei den Prozessen wieder frei wird. `static void yield()`

Threads beenden

In der `run()`-Methode tritt eine `RuntimeException` auf, die die Methode beendet. Das beendet weder die anderen Threads noch die JVM als Ganzes.

Beenden ist nicht nötig, weil die Methode run() ständig läuft, und nach dem Programmaufruf automatisch von der JVM Garbagekollektor weggeräumt wird.

Veraltete methoden wie stop() sind daher nicht zu empfehlen.

Der Vorteil von Java ist, dass die JVM dem Programmierer das Schreiben von „Beenden methoden“ erpart.

Mit Interrupt kann man den Thread bitten sich zu beenden. Dabei wird dem Thread mitgeteilt seine Arbeit periodisch aufzugeben.

Den aktuellen Status von diesem Aufruf kann man mit isInterrupted() überprüfen oder mit interrupt() einleiten

Quelle: Galileo Computing „Java ist auch eine Insel“