

High-Level Concurrency

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

Explizite Lock-Objekte

```
// Synchronisierung eines Teils einer Methode über ein  
// Lock-Objekt (seit Java 5)  
// Package `java.util.concurrent.locks`  
public int incrVal() {  
    Lock waechter = new ReentrantLock();  
    ...  
    waechter.lock();  
    ... // Geschützter Bereich  
    waechter.unlock();  
    ...  
}
```

Thread-Management: Executor-Interface und Thread-Pools

```
MyThread x = new MyThread();    // Runnable oder Thread
```

```
ExecutorService pool = Executors.newCachedThreadPool();
```

```
pool.execute(x);    // x.start()
```

```
pool.execute(x);    // x.start()
```

```
pool.execute(x);    // x.start()
```

```
pool.shutdown();    // Feierabend :)
```

Fork/Join-Framework: Teile und Herrsche

```
public class RecursiveTask extends ForkJoinTask<V> {  
    protected V compute() {  
        if (task klein genug) {  
            berechne task sequentiell  
        } else {  
            teile task in zwei subtasks:  
            left, right = new RecursiveTask(task)  
            rufe compute() auf beiden subtasks auf:  
            left.fork();           // starte neuen Thread  
            r = right.compute();   // nutze aktuellen Thread  
            warte auf ende der beiden subtasks: l = left.join()  
            kombiniere die ergebnisse der beiden subtasks: l+r  
        }  
    }  
}
```

Swing und Threads

- Implementieren:
 - `SwingWorker#doInBackground`: Für die langwierige Berechnung (muss man selbst implementieren)
 - `SwingWorker#done`: Wird vom EDT aufgerufen, wenn `doInBackground` fertig ist
- Aufrufen:
 - `SwingWorker#execute`: Startet neuen Thread nach Anlegen einer Instanz und führt dann automatisch `doInBackground` aus
 - `SwingWorker#get`: Return-Wert von `doInBackground` abfragen

Letzte Worte :-)

- Viele weitere Konzepte
 - Semaphoren, Monitore, ...
 - Leser-Schreiber-Probleme, Verklemmungen, ...
- => Verweis auf LV “Betriebssysteme” und “Verteilte Systeme”

- **Achtung:** Viele Klassen sind nicht Thread-safe!

Beispiel Listen:

- `java.util.ArrayList` ist **nicht** Thread-safe
- `java.util.Vector` ist Thread-sicher

=> Siehe Javadoc in den JDK-Klassen!

- Thread-safe bedeutet **Overhead** (Synchronisierung)!

Multi-Threading auf höherem Level: Thread-Pools und Fork/Join-Framework

- Feingranulareres und flexibleres Locking mit Lock-Objekten und Conditions
- Wiederverwendung von Threads: Thread-Management mit Executor-Interface und Thread-Pools
- Fork/Join-Framework zum rekursiven Zerteilen von Aufgaben und zur parallelen Bearbeitung der Teilaufgaben
- `SwingWorker` für die parallele Bearbeitung von Aufgaben in Swing



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.