

Konzepte der Asynchronität in modernen Programmiersprachen

Dr. Peter Dillinger
peter.dillinger@email.de

- Synchron und asynchron Methodenaufrufe
- Ergebnisabholung
 - Geteilter Speicher (*Shared Memory*)
 - Future
 - Rückruf (*Callback*)
- Über dem Tellerrand
 - C#
 - JavaScript
 - ABAP
- Zusammenfassung
- Literatur

Folien, Quellcode, Übungen
<https://github.com/phd4S/async>



Motivation

Synchroner Programmablauf

- blockierend
- arbeitet sequentiell

Asynchroner Programmablauf

- nicht-blockierend
- arbeitet parallel
- Erhöht die
Reaktionsfähigkeit

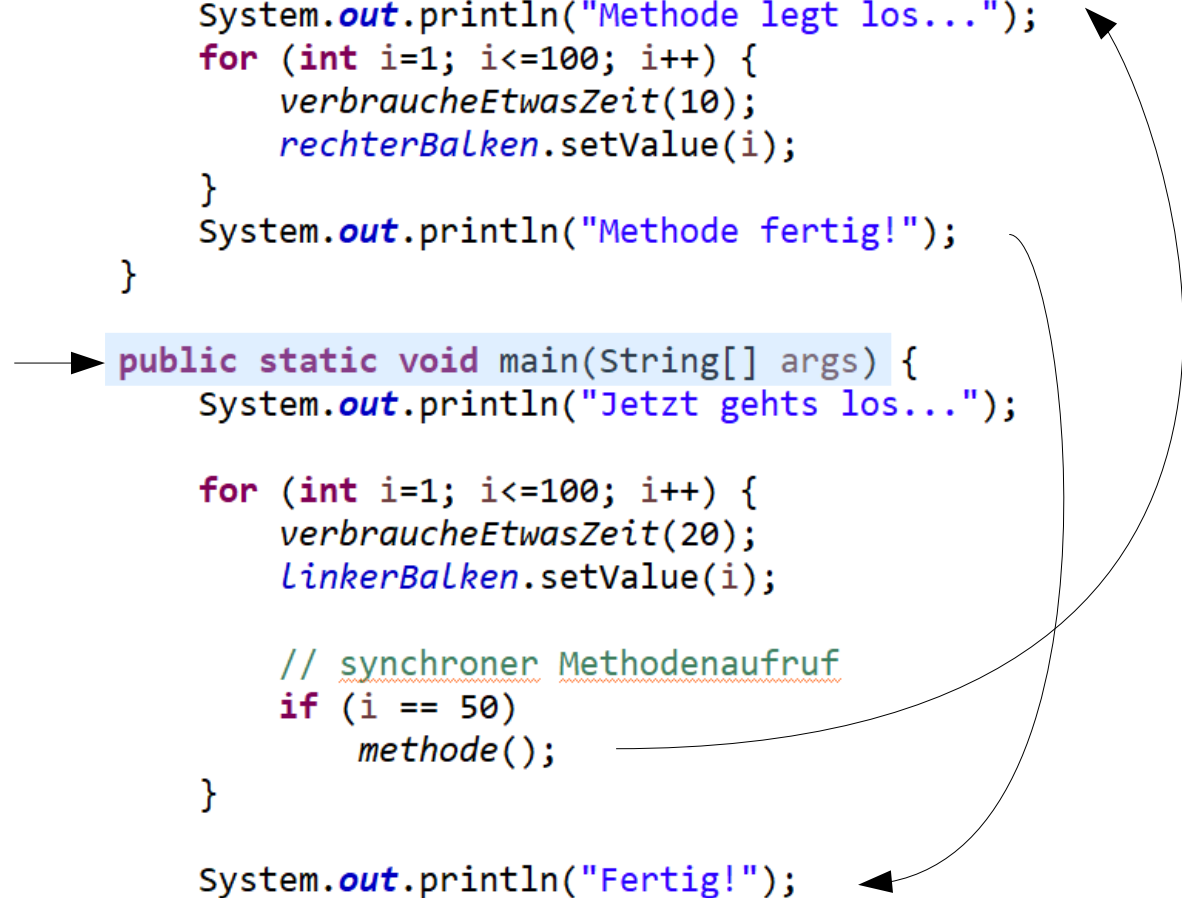
```
// eine normale Methode
public static void methode() {
    System.out.println("Methode legt los...");
    for (int i=1; i<=100; i++) {
        verbraucheEtwasZeit(10);
        rechterBalken.setValue(i);
    }
    System.out.println("Methode fertig!");
}

→ public static void main(String[] args) {
    System.out.println("Jetzt gehts los...");

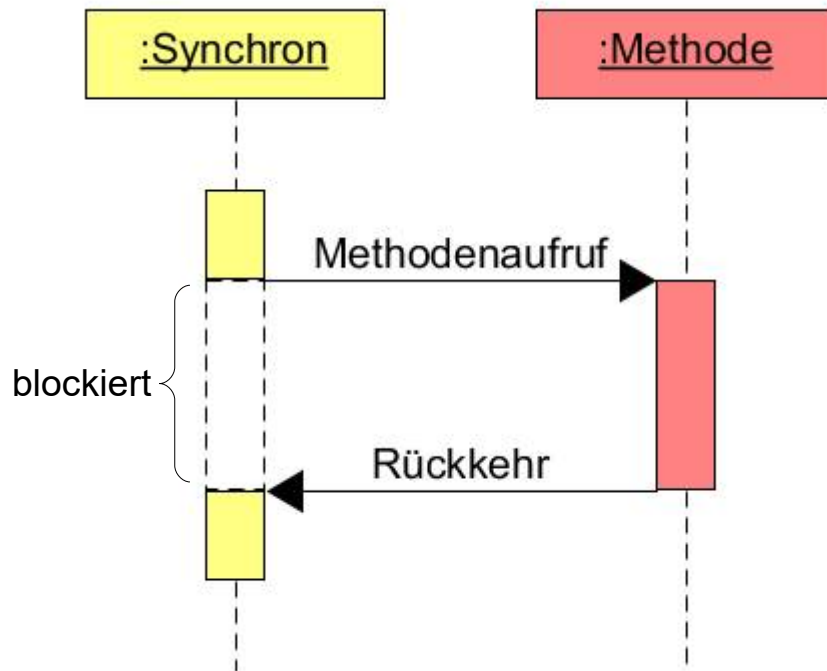
    for (int i=1; i<=100; i++) {
        verbraucheEtwasZeit(20);
        linkerBalken.setValue(i);

        // synchroner Methodenaufruf
        if (i == 50)
            methode();
    }

    System.out.println("Fertig!");
}
```

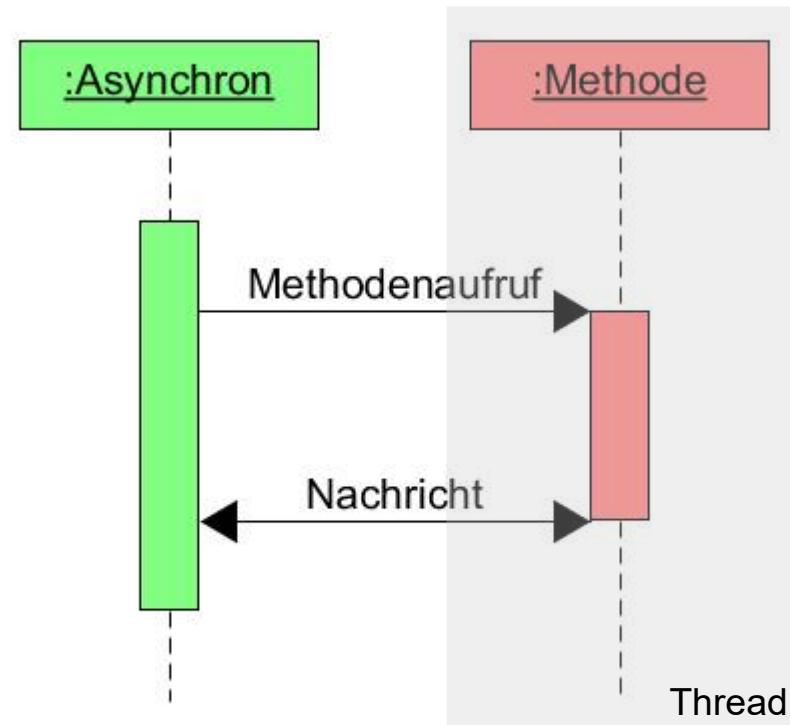


Methodenaufrufe



Synchroner Methodenaufruf

- Abgabe der Kontrolle
- nur eine Ausführungseinheit notwendig (Single- oder Multi-Threading)



Asynchroner Methodenaufruf

- Kontrolle wird beibehalten
- mehrere Ausführungseinheiten notwendig (Multi-Threading)
- Ergebnis muss abgeholt/übermittelt werden (Synchronisation)

- **Geteilter Speicher (*Shared Memory*)**
 - Ergebnis wird in einer Variable (Attribut) hinterlegt, auf dem beide Methoden Zugriff haben
 - Kann zu Synchronisationsproblemen führen: Wettlaufbedingung (*Race Condition*) und Verlorenes Update (*Lost Update*)
 - Lösung: Sperrung der Variable (Attribut) per Mutex, Semaphoren
 - Kann zu weiteren Synchronisationsproblemen führen: Verklemmung (*Deadlock*) oder Verhungern (*Starvation*)

- Future (*Promises, Delay, Deferred*)
= Platzhalter (Proxy) für ein Ergebnis
 - noch nicht bekannt bzw.
Berechnung noch nicht abgeschlossen

```
public interface Future<V> {  
    boolean cancel(boolean mayInterruptIfRunning);  
    boolean isCancelled();  
    boolean isDone();  
    V get() throws InterruptedException, ExecutionException;  
    V get(long timeout, TimeUnit unit)  
        throws InterruptedException, ExecutionException, TimeoutException;  
}
```

- get(...) ist ein blockierender Aufruf
- Führt meist zum Polling:
 while (!isDone()) ...

- Rückruf (Callback, Delegate)
 - nach Abschluss der Methode wird eine andere designierte Methode aufgerufen
 - Aufrufer muss nicht mehr um die Abholung des Ergebnis kümmern, sondern stellt eine gesonderte Methode bereit

```
class CompletableFuture<T> implements Future<T>
```

```
runAsync(...)
```

```
supplyAsync(...)
```

```
thenApply(...)
```

```
thenAccept(...)
```

```
thenRun(...)
```

Über dem Tellerrand

- C#
- JavaScript
- ABAP

Zusammenfassung

Konzepte der Asynchronität in modernen Programmiersprachen

- Lernziele

Quellen / weiterführende Literatur

- **Lane, Hobson; Howard, Cole; Hapke, Hannes Max (2019)**
Natural Language Processing in Action
- **Goyal, Palash; Pandey, Sumit; Jain, Karan (2018)**
Deep Learning for Natural Language Processing
- **Vasiliev, Yuli (2020)**
Natural Language Processing with Python and spaCy
- ***Diverse Autoren***
Natural Language Processing with Java / R / ... / Python
(<http://www.nltk.org/book/>)
- **Russell, Stuart; Norvig, Peter (2016)**
Artificial Intelligence - A Modern Approach
- **Ertel, Wolfgang (2017)**
Introduction to Artificial Intelligence

Zum Schluss ...