Teil XI

Parallelität

Threads

Ablauf einer Berechnung bei einer rechnenden Einheit.

- 3 Programme: rot, grün, blau
- Jeder Programm bekommt **fair** kurze Rechenzeit.
- Für Anwender sieht es so aus, als würden alle drei Programm **gleichzeitig** ablaufen.
- Unter anderem nachdem die Beschleunigung einer einzelnen rechnenden Einheit nicht mehr ohne weiteres möglich war, wurden CPU's mit **mehreren** rechnenden Einheiten konstruiert.

Threads

- Auf Betriebssystemebene entspricht zunächst jedes Programm einem **Prozess**.
- Prinzipiell können soviele Prozesse **gleichzeitig** (parallel) ausgeführt werden, wie es rechnende Einheiten gibt.

Threads

- Auf Betriebssystemebene entspricht zunächst jedes Programm einem **Prozess**.
- Prinzipiell können soviele Prozesse **gleichzeitig** (parallel) ausgeführt werden, wie es rechnende Einheiten gibt.

- Threads ermöglichen die parallele Ausführung von **Teilen** eines Programmes.
- Nun können soviele Threads **gleichzeitig** (parallel) ausgeführt werden, wie es rechnende Einheiten gibt.
- Anstelle des Programmes als ein Prozess gibt es nun soviele Prozesse, wie das Programm Threads hat.

Threads

Hierzu muss eine Thread gestartet werden können (Java: start).

Zudem muss auf das Ende des Threads gewartet werden können (Java: join).

t1	start	···	end	
t2	start	···	end	
t3	start	···	end	join
t4	start	···	end	
t5	start	···	end	

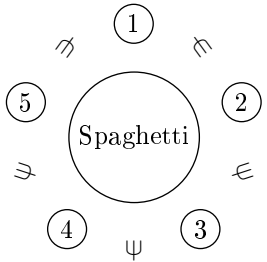
Threads: Deadlocks

- Die Verwendung von Threads ist nicht unproblematisch.
- Bei nicht ausreichender Konzeption kann es zu sogenannten **deadlocks** kommen:
  der Prozess hängt und wird nie wieder fortgesetzt.

Threads: Deadlocks

- Die Verwendung von Threads ist nicht unproblematisch.
- Bei nicht ausreichender Konzeption kann es zu sogenannten **deadlocks** kommen:
  der Prozess hängt und wird nie wieder fortgesetzt.
- Solche deadlocks gilt es zu vermeiden
  $\rightarrow$ Theoretische Informatik.

Threads: Dining Philosophors



Threads: Dining Philosophors

- Gegeben:
  - n Philosophen
  - Teller mit Spaghetti (wird immer nachgefüllt)
  - n Gabeln zwischen den Philosophen

Threads: Dining Philosophors

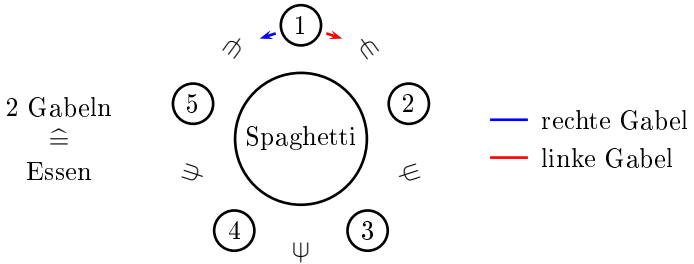
- Gegeben:
  - n Philosophen
  - Teller mit Spaghetti (wird immer nachgefüllt)
  - n Gabeln zwischen den Philosophen
- Aufgabe der Philosophen
  - Nachdenken
  - Essen:
    Philosoph benötigt 2 Gabeln, die rechts und links neben ihm liegen

Threads: Dining Philosophors

- Gegeben:
  - n Philosophen
  - Teller mit Spaghetti (wird immer nachgefüllt)
  - n Gabeln zwischen den Philosophen
- Aufgabe der Philosophen
  - Nachdenken
  - Essen:
    Philosoph benötigt 2 Gabeln, die rechts und links neben ihm liegen
- Constraint:
  - Will ein Philosoph essen, so nimmt er immer zuerst die rechte und dann die linke Gabel

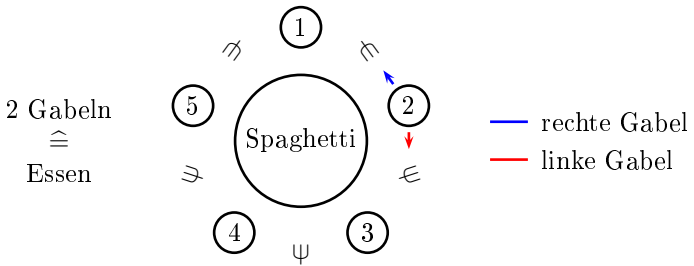
Threads: Dining Philosophors

Sequentiell: Philosoph 1



Threads: Dining Philosophors

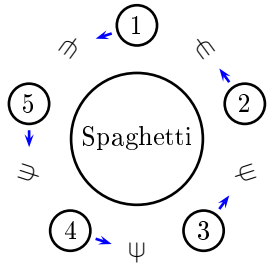
Sequentiell: Philosoph 2



Threads: Dining Philosophors

Parallel: alle gleichzeitig

2 Gabeln
$\widehat{=}$
Essen

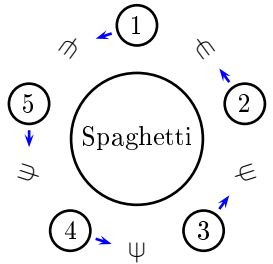


rechte Gabel
linke Gabel

Threads: Dining Philosophors

Parallel: alle gleichzeitig

2 Gabeln
$\widehat{=}$
Essen



$\rightarrow$ Keiner kann die linke Gabel nehmen, da sie der links sitzende Philosoph schon als seine rechte Gabel benutzt.

Example: Counting Thread

```java
package threadExample;

import java.util.List;
import java.util.Map;

/**
 * @author zeckzer
 */
public class CounterThread
  extends Thread {

  private final List<Integer> values;
  private final Map<Integer, Integer>
      counts;

  /**
   * @param values
   * @param counts
   */
  public CounterThread(
    List<Integer> values,
    Map<Integer, Integer> counts
  ) {
    super();
    this.values = values;
    this.counts = counts;
  }

  @Override
  public void run() {
    for (Integer value : values) {
      synchronized (counts) {
        if (counts.containsKey(value)) {
          int count = counts.get(value);
          ++count;
          counts.put(value, count);
        } else {
          counts.put(value, 1);
        }
      }
    }
  }
}
```

Example: Counting Thread

```
1     private static void printContent(
2       Map<Integer, Integer> counts
3     ) {
4       for (Map.Entry<Integer, Integer> pair : counts.entrySet()) {
5         System.out.print(pair.getKey() + ":" + pair.getValue() + " ");
6       }
7       System.out.println("");
8     }
```

Example: Counting Thread

```
1     private static void testThread(
2       List<Integer> values,
3       Map<Integer, Integer> counts
4     ) {
5       Thread t1 = new CounterThread(values, counts);
6       t1.start();
7       Thread t2 = new CounterThread(values, counts);
8       t2.start();
9       Thread t3 = new CounterThread(values, counts);
10      t3.start();
11      Thread t4 = new CounterThread(values, counts);
12      t4.start();
13      Thread t5 = new CounterThread(values, counts);
14      t5.start();
15      Thread t6 = new CounterThread(values, counts);
16      t6.start();
17      Thread t7 = new CounterThread(values, counts);
18      t7.start();
19      Thread t8 = new CounterThread(values, counts);
20      t8.start();
21      Thread t9 = new CounterThread(values, counts);
22      t9.start();
23      Thread t10 = new CounterThread(values, counts);
24      t10.start();
25
26      try {
27        t1.join();
28        t2.join();
29        t3.join();
30        t4.join();
31        t5.join();
32        t6.join();
33        t7.join();
34        t8.join();
35        t9.join();
36        t10.join();
37      } catch (Throwable thr) {
38      }
39
40      printContent(counts);
41    }
42
```

Example: Counting Thread

```
1  package threadExample;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.List;
6  import java.util.Map;
7
8  /**
9   * @author zeckzer
10  */
11 public class ThreadExample {
12
13   /**
14    * @param args the command line arguments
15    */
16   public static void main(String[] args) {
17     List<Integer> values = new ArrayList<>();
18     for (int i = 1; i <= 50; ++i) {
19       values.add(i);
20     }
21     Map<Integer, Integer> counts = new HashMap<>();
22
23     counts = new HashMap<>();
24     System.out.println("testThread HashMap");
25     testThread(values, counts);
26   }
```