Übungsstunde 2

Programm für heute

- Assignment2 besprechen
- Threads allgemein
- Synchronisation
- Assignment3 vorbesprechen

Assignment2

- Gut gelöst
- Viele Wege führen zum Ziel

Threads

Die Klasse Thread

- Gehört zur Standardbibliothek von Java
- Kreiere neues Objekt mit
  - Thread worker = new Thread();
- Mit worker.start() wird der Thread gestartet und run() aufgerufen
- run() muss implementiert werden (da „deferred“)
- Code nach dem start() wird parallel zum run() ausgeführt.

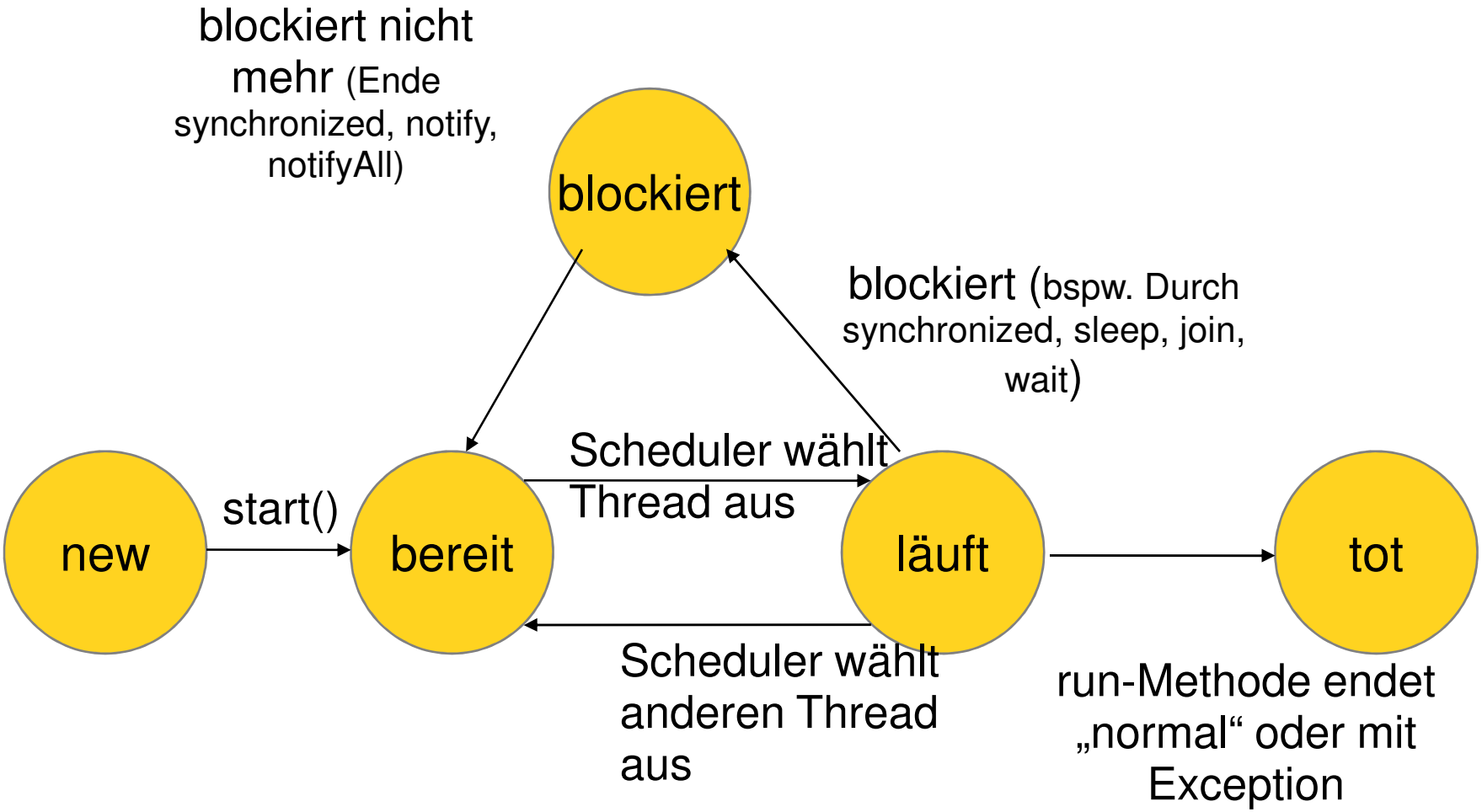
```
public static void main(String[] args)
{
        MyThread thread1 = new MyThread();
        thread1.start();
        System.out.println("Hello World");
}

public void run() {
        System.out.println("I'm a
        Thread");
}
```

Das Interface Runnable

- Funktioniert ähnlich wie beim erben von Thread
- Vorteil: man kann erben

- Beispiele hat es auf meiner Webseite
  - www.mymoutains.ch → PP

Zustand eines Threads



Synchronisation

Warum brauchen wir Synchronisation?

- Threads kommunizieren über gemeinsame Daten
- Gibt es nur Leseoperationen kein Problem
- Was wenn mehrere Threads gleichzeitig auf eine Variable schreiben (und lesen)?
  - Fehler können passieren wenn Operationen nicht atomar sind
- → Zugriff muss also geregelt werden in den meisten Fällen

Beispiel

Counter = 0; // wird von Thread A und B geteilt

Thread A: couter++

Thread B: System.out.println(counter)

→ Output unklar, 1 oder 0?

Schlüsselwort: synchronized

- Es gibt zwei Möglichkeiten:
  - Eine Methode synchronisieren

```
synchronized void foo() {}
```

  - Einen Codeblock synchronisieren

```
public void foo() {
        //nicht synchronisiert
        synchronized(this) {
                //kritischer Bereich
        }
        //nicht synchronisiert
}
```

Was passiert in synchronisiertem Code?

- Beim Eintritt in eine synchronisierte Methode oder einen Codeblock wird eine Sperre gesetzt
  - In dieser Zeit kann kein anderer Thread auf den Codeteil zugreifen, sie werden blockiert
- Beim Austritt wird die Sperre entfernt
  - Entweder durch Freigabe der Sperre
  - Oder wenn eine Exception ausgelöst wurde

„Reentrant“ Synchronisation

- Threads können “locks” verlangen, die sie bereits besitzen
- Beschreibt Situation, in der in synchronisiertem Code eine Methode aufgerufen wird, welche auch synchronisierten Code enthält und beide das selbe „lock“ benützen
- Ohne „reentrant synchronization“ währen weitere Vorsichtsmassnahmen nötig, so dass sich Threads nicht selber blocken.

Assignment3

Um was geht es?

- Implementieren von Erzeuger/Verbraucher Beziehung
  - Beide laufen als Thread
- Erzeuger produziert konstant Integers, die der Verbraucher jeweils 1x abfragen soll
- Verwendung eines Puffers zum kommunizieren
  - Erzeuger schreibt Wert in Puffer
  - Verbraucher liest Wert aus Puffer
- Der Prozess muss synchronisiert werden.

Output zu Beginn

```
Producer produced: 0
          Consumer consumed: 0
          Consumer consumed: 0
Producer produced: 1
          Consumer consumed: 1
Producer produced: 2
          Consumer consumed: 2
Producer produced: 3
Producer produced: 4
          Consumer consumed: 4
          Consumer consumed: 4
```

Korrekter Output

```
Producer produced: 0
            Consumer consumed: 0
Producer produced: 1
            Consumer consumed: 1
Producer produced: 2
            Consumer consumed: 2
Producer produced: 3
            Consumer consumed: 3
Producer produced: 4
            Consumer consumed: 4
Producer produced: 5
            Consumer consumed: 5
```

Oder auch so:

```
Producer produced: 0
          Consumer consumed: 0
Producer produced: 1
          Consumer consumed: 1
          Consumer consumed: 2
Producer produced: 2
Producer produced: 3
          Consumer consumed: 3
Producer produced: 4
          Consumer consumed: 4
```

Eine Bitte

- Kommentiert euren Code, das erleichtert mir die Arbeit massiv
- Danke!!!