Nebenläufige Programmierung - Threads

Programmierpraktikum 2012

Christian Lovato

Prozesse und Threads

- Jeder Prozess besitzt eigenen Speicher
- Prozesse können mehrere Threads beinhalten
- Threads „teilen“ sich den Speicher

Prozesse und Threads

- Prozesse sind „stur“
  - Strikte Abfolge von Befehlen
- Threads sind „flexibel“
  - Laufen im Hintergrund und die Ausführungs-reihenfolge wird vom System bestimmt

Threads

- Verwendung von Threads:
  - Interface:

```
public class Blololol implements Runnable{
        @overide public void run() { … }
}
```

Anschließend:

```
Thread t = new Thread(new Blololol);
t.start();
```

Threads

- Vererbung:

```
public class MeinThread extends Thread(){
        @overide public void run() { … }
}
```

Dann:

```
Thread t = new MeinThread();
t.start();
```

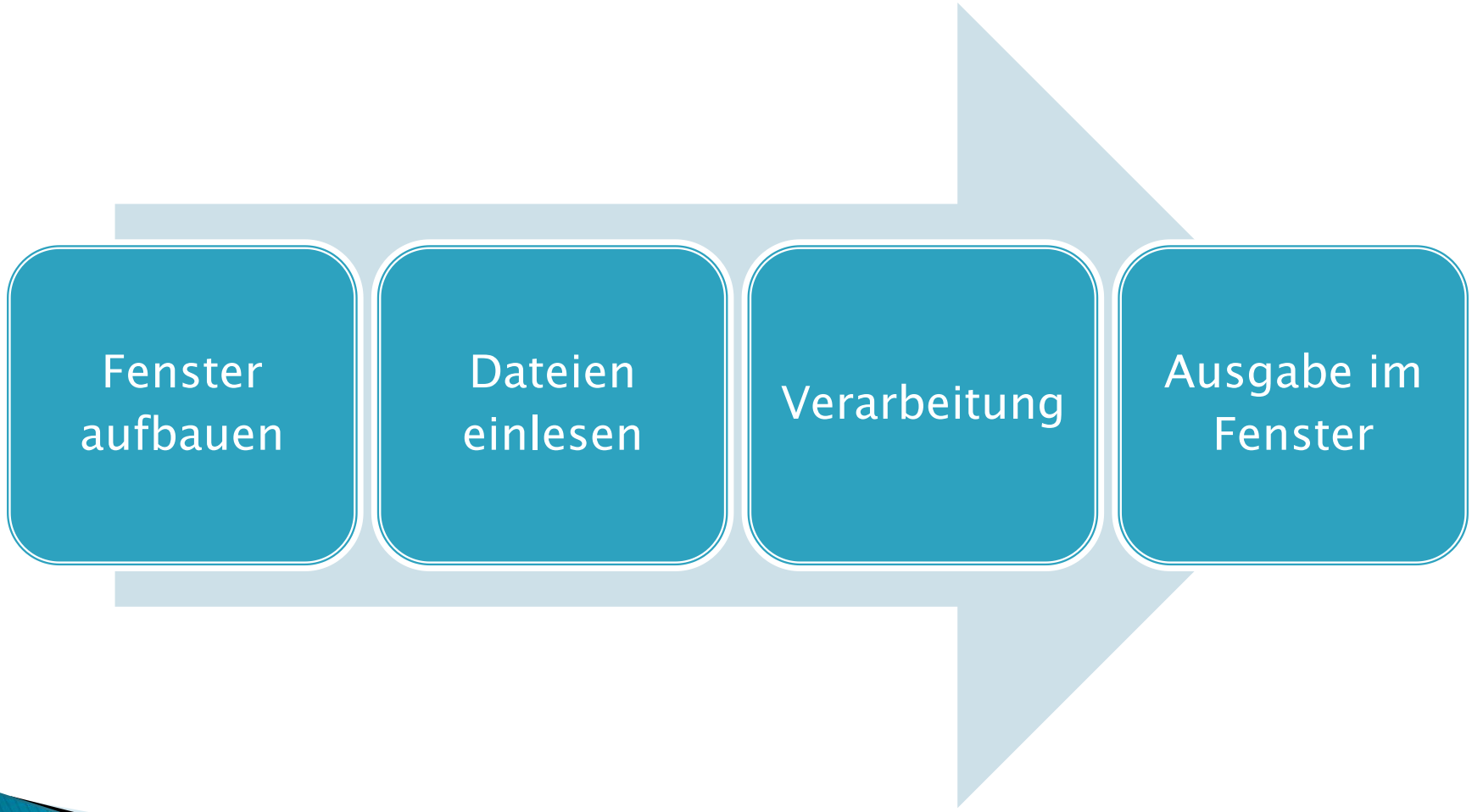
Threads

- Weitere Anwendungsgebiete:
  - Timer
  - ActionListener
  - …

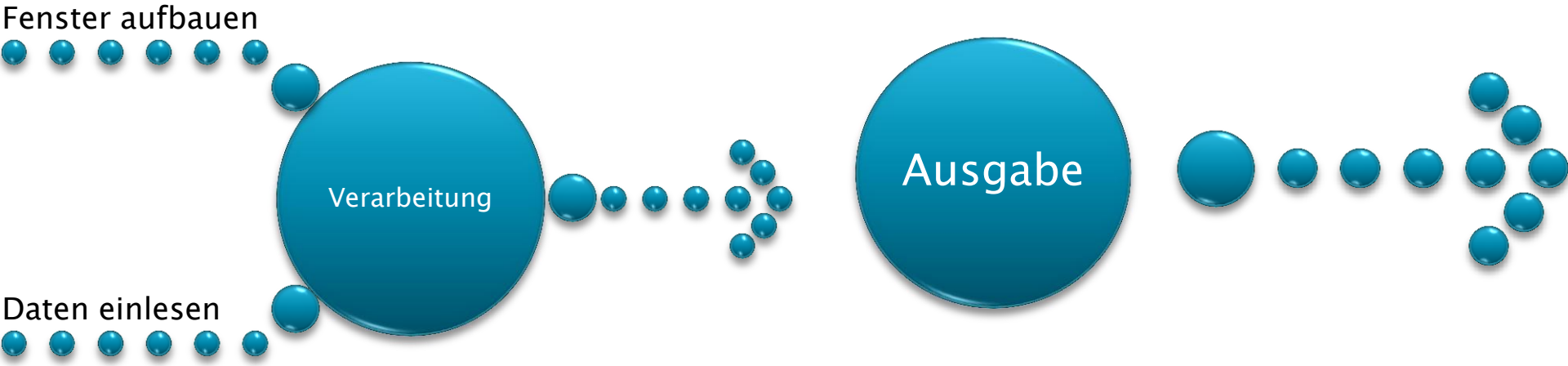
Vorteile

- Latente Ausführung im Hintergrund
- Parallele Ausführung von Anweisungen
- Steigerung der Geschwindigkeit

Beispiel


Fenster aufbauen
Dateien einlesen
Verarbeitung
Ausgabe im Fenster

Beispiel



Vorsicht ist geboten…

Vorsicht ist geboten…

- Kritische Abschnitte im Programm

- Funktionen die nicht kritisch erscheinen:
  - i++;

Lock

- Zugriff auf Objekte kann gesperrt werden
- Thread muss ggf. auf Freigabe warten
- Prinzip eines Tors: lock() und unlock()

Lock Beispiel

```
final Lock lock = new ReentrantLock();
final Point punkt = new Point();

Runnable r = new Runnable(){

    @Override
    public void run() {

            int x = (int)(Math.random() * 1000), y = x; //z.B x = 3,   y = 8

            while ( true ) {


                    punkt.x = x;            punkt.y = y;          //   P(3|8)
                    int xc = punkt.x,       yc = punkt.y;         //xc = 3,   yc = 8
                    if ( xc != yc )
                            System.out.println( "Aha: x=" + xc + ", y=" + yc );

            }
    }
};
new Thread( r ).start();
new Thread( r ).start();
```

Ausgabe:

Aha: x=5, y=8
Aha: x=8, y=5
Aha: x=5, y=8
Aha: x=5, y=8
...

Lock Beispiel

```
final Lock lock = new ReentrantLock();
final Point punkt = new Point();

Runnable r = new Runnable(){

    @Override
    public void run() {

            int x = (int)(Math.random() * 1000), y = x;

            while ( true ) {


                    punkt.x = x; punkt.y = y;
                    int xc = punkt.x, yc = punkt.y;


            }
    }
};
new Thread( r ).start();
new Thread( r ).start();
```

Lock Beispiel

```
final Lock lock = new ReentrantLock();
final Point punkt = new Point();

Runnable r = new Runnable(){

    @Override
    public void run() {

            int x = (int)(Math.random() * 1000), y = x;

            while ( true ) {

                    lock.lock();
                    punkt.x = x; punkt.y = y;
                    int xc = punkt.x, yc = punkt.y;
                    lock.unlock();

            }
    }
};
new Thread( r ).start();
new Thread( r ).start();
```

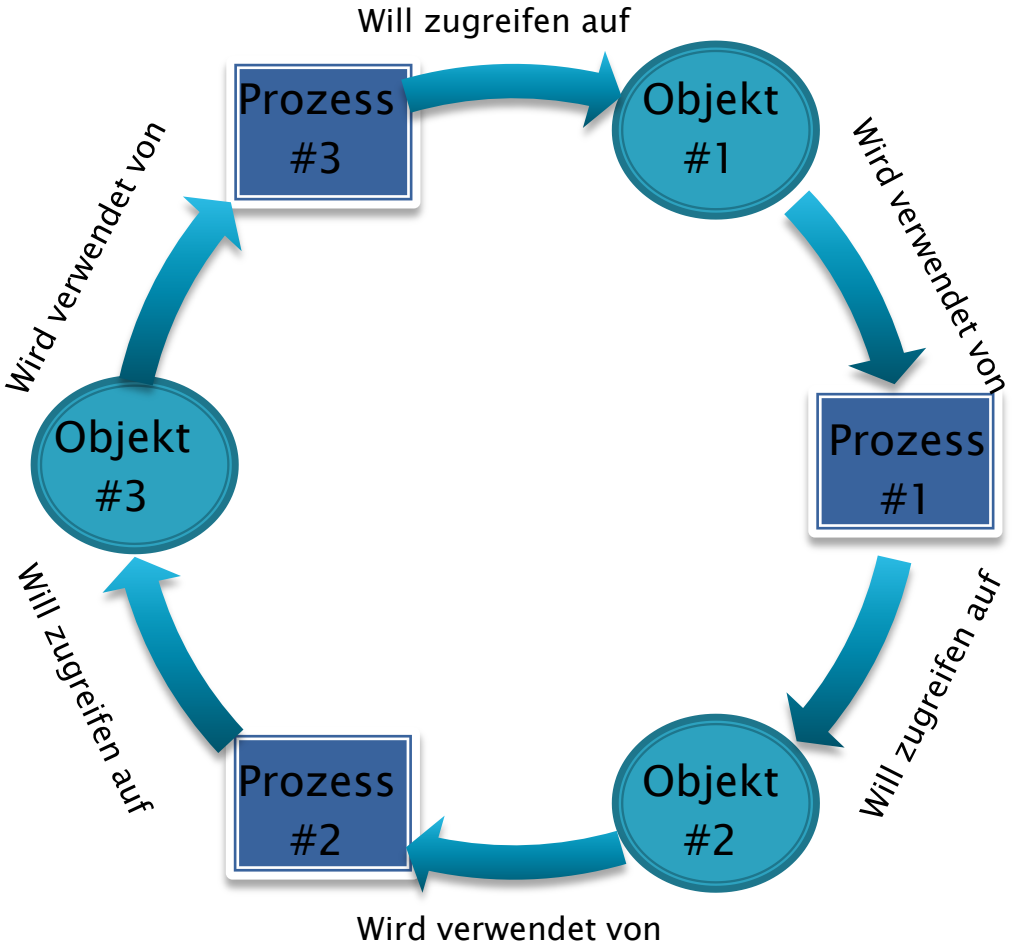
Synchronisation

- Synchronisierte Methoden verhindern den Fehler von eben:
  - synchronized void iErhoehen() { i++; }
- „Automatisiertes Lock-Verfahren“
- Endlosschleifen sperren Objekt für immer

Lock und Synchronized

- Unachtsames verwenden kann zu „Deadlock“ führen

Deadlock


Will zugreifen auf
Prozess #3
Objekt #1
Wird verwendet von
Prozess #1
Will zugreifen auf
Objekt #2
Wird verwendet von
Prozess #2
Will zugreifen auf
Objekt #3
Wird verwendet von

Beispiele in Eclipse

Quellen

- http://download.oracle.com/javase/6/docs/api/javax/swing/Timer.html
- http://openbook.galileocomputing.de/javainsel9/
- http://de.wikipedia.org/wiki/Thread_%28Informatik%29
- http://de.wikipedia.org/wiki/Deadlock

Danke für Eure Aufmerksamkeit!