

Nachtrag zu Annotationen

Annotation

```
@Retention(Retention.Policy=RUNTIME)
public @UnsereAnno {
    String wert();
    int[] noten;
    //verboten: Student student();
    Jahreszeit jahreszeit();
}
```

```
public enum Jahreszeit {
    Sommer, Winter
}
```

Klasse die Annotation verwendet

```
@UnsereAnno(wert="Fritz",
             noten={1, 3, 2, 1},
             jahreszeit=Jahreszeit.Sommer)
public void doSth() {
}
```

Auslesen von Annotationen (Klassen, Methoden, Attributen)

```
Annotation anno =
    Studentin.class.getAnnotation(UnsereAnno.class);
```

```
Method methode = Studentin.class.getMethod("doSth");
Annotation anno =
    methode.getAnnotation(UnsereAnno.class);
```

```
Field field = Studentin.class.getDeclaredField("ein Attribut");
Annotation anno =
    field.getAnnotation(UnsereAnno.class);
```

Nebenläufigkeit

Auf einem Rechner können mehrere Programme parallel ausgeführt werden (-> Multi-tasking). Das Betriebssystem verwaltet dazu mehrere Prozesse. Jeder Prozess

- hat eine Prozess-ID (PID)
- hat einen eigenen Adressraum
- hat eigene Ressourcen (I/O (Dateien), Netzwerk, Interfaces)
- mindestens einen Thread

Threads

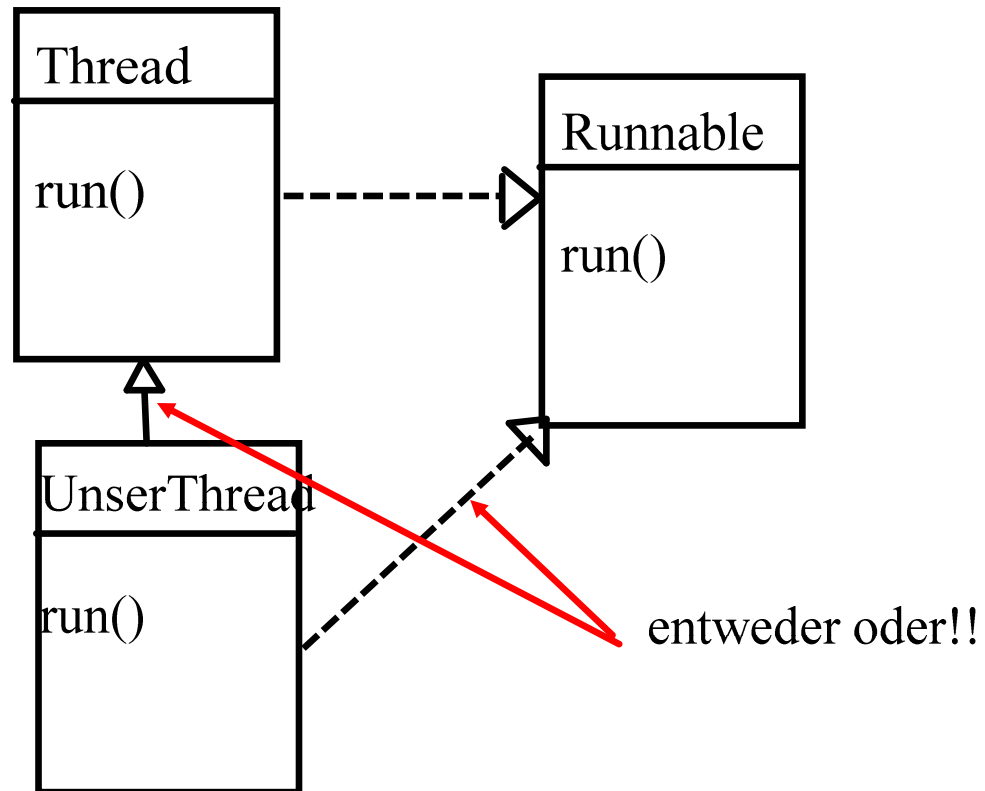
mit mehreren Threads kann Nebenläufigkeit innerhalb eines Prozesses (Programms) erreicht werden. z.B. um

- Drucken im Hintergrund
- Zugriff auf Netzwerk
- Berechnungen im Hintergrund
-

Threads mit Java realisieren

Zwei Möglichkeiten

1. Von Thread erben
2. Interface Runnable implementieren



```
package threads;  
  
public class StartThreads {  
    public static void main(String[] args) {  
        Thread slowThread = new Thread(new SlowThread(), "Ich bin der langsame Thread");  
        Thread fastThread = new Thread(new FastThread(), "Ich bin der schnellere Thread");  
  
        slowThread.start();  
        fastThread.start();  
  
        System.out.println("Bin fertig");  
    }  
}
```

Achtung:
- run() wird nicht parallel ausgeführt
- start() wird parallel ausgeführt

Beenden von Threads

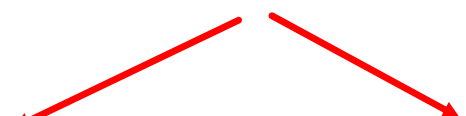
- Methode run() ist abgearbeitet
- Methode run() bricht mit Fehler ab
- Methode wird mit interrupt() unterbrochen und dies vom Thread auch abgefragt wird
- Hauptthread endet und "unser" Thread wurde mit daemon(true) gestartet.

```
public void run(){  
    while(!interrupted()){  
        System.out.println("Ich bin im Einsatz");  
    }  
    System.out.println("Ich wurde interrupted");  
}
```

```
infinityThread.setDaemon(true); //Flag, ob Thread mit dem Hauptprogramm enden soll
```

Wenn ein Thread (z.B. der Hauptthread) warten soll, bis ein anderer Thread fertig ist, so nutzt man die Methode `join`:

```
try {  
    Thread.sleep(10000);  
    agentThread.start();  
    agentThread.interrupt();  
    agentThread.join(); //Hauptthread wartet auf AgentThread  
} catch (InterruptedException e) {  
    e.printStackTrace();  
}  
  
System.out.println("Bin fertig");
```



Mehrfachausführung von Threads

Um einen Thread mehrfach ausführen zu können, bedarf eines Executors bzw. des ExecutorServices. Mit dieser Klasse kann man einen ThreadPool aufbauen:

`new CachedThreadPool()`: Pool mit variabler Größe

`new FixedThreadPool()`: Pool mit fixer Größe

```
Thread slowThread = new Thread(new SlowThread(), "Ich bin der langsame Thread");  
Thread fastThread = new Thread(new FastThread(), "Ich bin der schnellere Thread");
```

```
ExecutorService executor = Executors.newCachedThreadPool();  
executor.execute(slowThread);  
executor.execute(fastThread);
```

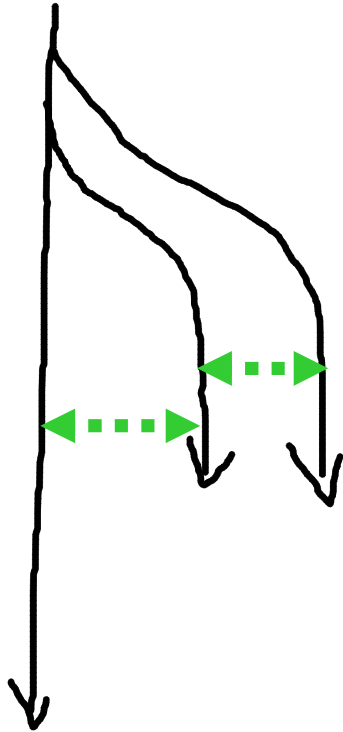
```
Thread.sleep(2000);
```

```
executor.execute(slowThread);  
executor.execute(fastThread);
```

```
executor.shutdown();
```

Die beiden Threads werden erstmalig gestartet

Die beiden (identischen) Threads werden zum zweiten Mal gestartet (ohne diese neu zu instanzieren)



In der nächsten Stunde werden wir untersuchen, wie Threads untereinander kommunizieren können.