OOP: Nebenläufigkeiten Threads

Dipl.-Inform. Arnold Willemer

arnold.willemer@hs-flensburg.de

Schlafen für Profis

- Die C-64-Lösung kocht und blockiert den Prozessor

```
while (!fertig)
{
        // nichts tun: busy waiting oder polling! Böse!
}
```

- Sleep sorgt für entspanntes Warten

```
try
{
        Thread.sleep(1000); // 1000ms!
}
catch (InterruptedException e)
{
}
```

- Gelegentliches Schlafen entspannt Prozesse

Countdown

- Das Programm zählt sekundenweise herunter

```
public class CountDown
{
	public static void main(String[] args)
	{
		for (int i = 10; i > 0; --i)
		{
			System.out.println(i);
			try
			{
				Thread.sleep(1000);
			}
			catch (InterruptedException e)
			{
			}
		}
	}
}
```

Thread, der Hintergrundarbeiter

- Die Klasse Thread implementiert die Methode run. Darin wird der parallel laufende Code geschrieben.

```
public class GlockenThread extends Thread
{
	@Override
	public void run()
	{
		for (int i=0; i<anzahl; i++)
		{
			try
			{
				Thread.sleep(zeit);
				System.out.println(meldung);
			} catch(InterruptedException e) {
			}
		}
	}
```

Thread laufen lassen

- Das Thread-Objekt wird durch Aufruf der Methode start gestartet

```
Thread thread1 = new GlockenThread("bing", 10, 600);
thread1.start();
Thread thread2 = new GlockenThread("  bang", 15, 400);
thread2.start();
```

- Der Thread endet beim Erreichen der Methode run
- Die Methode main bildet den Haupt-Thread.

Das Interface Runnable

- Alternative zum Erweitern von Thread:
  - Interface Runnable implementieren, insbesondere dessen Methode run
  - Runnable-Implementierung als Parameter dem Konstruktor von Thread übergeben.
  - Thread-Objekt über dessen Methode start starten

Parallellauf per Thread

```
public class Glocken implements Runnable {

    public Glocken(String meldung, int anzahl, long zeit) {
     … }

    @Override
    public void run() {
        for (int i=0; i<anzahl; i++) {
            try {
                Thread.sleep(zeit);
                System.out.println(meldung);
            } catch(InterruptedException e) {
            }
        }
    }

    public static void main(String[] args) {
        new Thread(new Glocken("bing", 10, 6)).start();
        new Thread(new Glocken("bang", 15, 4)).start();
        new Thread(new Glocken("bong", 8, 8)).start();
    }
}
```

Gemeinsames Spielen

- Alle gemeinsam zur Oma ohne Quengeln
  - Lösung: Einsammeln, wenn fertig gespielt (join)
- Spielzeug, das nur einer gleichzeitig nutzen kann
  - Lösung: Aussperren der Geschwister (synchronized)
- Mama muss das kaputte Spielzeug reparieren.
  - Lösung: warten bis Mama ruft (wait, notify)
- Bettgehzeit: Schluss mit Spielen!
  - Lösung: Kind an den Ohren ins Bett tragen (interrupt)

Auf Glockenende warten (join)

```
static List<Thread> gelaeut = new ArrayList<Thread>();

public static void main(String[] args)
{
	gelaeut.add(new Thread(new Glocken("bing", 10, 600)));
	gelaeut.add(new Thread(new Glocken("  bang", 15, 400)));
	gelaeut.add(new Thread(new Glocken("    bong", 8, 800)));
	for (Thread thread : gelaeut)
	{
		thread.start();
	}
	for (Thread thread : gelaeut)
	{
		try
		{
			thread.join();
		} catch (InterruptedException e) {
		}
	}
	System.out.println("Geläute zu Ende");
}
```

Kritischer Bereich

- Vasenfabrik hat mehrere Öfen, aber nur eine Verpackungsmaschine.
- Ziel: Eine Vase pro Karton
- Der Zugang zur Verpackung ist ein kritischer Bereich: Es darf nur ein Thread gleichzeitig den Bereich betreten.
- Zauberwort: synchronized

```
public synchronized void verpacke()
{
        // ...
}
```

- Programm ThreadVasen

Auf Objekte warten

- Object hat die Methoden wait und notify
- Der Aufruf von wait legt den Aufrufer schlafen bis ein anderer Thread notify aufruft.
- Das Warte-Object muss synchronized sein!
- In der Vasenfabrik muss der Verpacker auf neue Lieferungen von Kartons warten. Solange stoppt die Produktion.

Statt warten auf Godot: Time-Out

- Wenn es während der Wartezeit die Uhr Feierabend anzeigt, unterbricht der Verpacker seine Wartezeit und geht nach Hause.
- Die Uhr unterbricht (interrupt) die Tätigkeit des Verpackers
- Zeitbombe - ZeitbombeThread

Fensterprobleme

- Fensterprogramme warten immer auf Ereignisse. Was passiert, wenn ein Vorgang länger dauert?
- Die grafische Oberfläche eines Programms ist (in der Regel) nicht threadsafe.
- Swing stellt einen Timer zur Verfügung.
- Timer meldet sich wie Button beim ActionListener
- Keine echte Nebenläufigkeit!

Zwischenstopp Button

- Wie fängt man ein Button-Event?

```
public class Zeitbombe extends Jframe
                        implements ActionListener  {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (scharf) {
            button.setText("Start");
        } else {
            button.setText("Stopp");
        }
        scharf = !scharf;
    }
        // ...
        button.addActionListener(this);
        // ...
}
```

Swings eigener Timer

- Timer timer = new Timer(zeit, listener);
- timer.setRepeats(true);
- timer.start();
- timer.stop();
- Wir spielen James Bond!