

1 Beispiel: Bank

1.1 Ausgangssituation – Package `UE19.nothread`

Wir brauchen mehrere Klassen:

- Konto
 - verwaltet ein Konto mit Kontostand
 - Attribut: `kontostand` (Vereinfachung: `int`)
 - Konstruktor (Startwert ist 0)
 - getter *und* setter
 - `add(int betrag)` – Einzahlung (umständlich mit Zwischenschritten):

```
int wert = getKontostand();  
wert = wert + betrag;  
setKontostand(wert);
```

- Ueberweiser
 - eine Klasse die *vieler* Überweisungen von einem Konto auf ein anderes Konto vornimmt.
 - zwei **Konten** als Attribute: von, nach
 - zwei Konstanten:


```
* ANZAHL = 10_000_000
* BETRAG = 10
```
 - eine Methode `run()`:


```
for (int i=0; i < ANZAHL; i++) {
    von.add(-BETRAG);
    nach.add(BETRAG);
}
```
- `main()`
 - Erzeuge drei Konten a, b und c.
 - Und drei Überweiser – wir wollen das Geld im Kreis schicken: a nach b, b nach c, c nach a.
 - Bestimme die Zeit für alle Überweisungen mit `System.currentTimeMillis()`.
 - Zeitdauer als Kommentar unten in der Datei speichern.
 - Kontrolle: Kontostände ausgeben.

Gib diese Version ab!

1.2 Schneller mit Threads – Package `UE19.thread`

Wir wollen die Überweisungen `parallel` ausführen – jeder Ueberweiser läuft in einem eigenen Thread.

```
class Ueberweiser extends Thread {
    ...
}
```

- `main()`
 - **Achtung:** der Start erfolgt jetzt mittels `ab.start()` – nicht `run()` aufrufen!
 - Die Überweisungen laufen jetzt parallel in eigenen Threads. Für die Zeitmessung müssen wir das Ende der einzelnen Threads abwarten: `einThread.join();`.
 - * alle Threads erzeugen
 - * alle Threads starten
 - * alle Threads joinen
 - Wie lange dauert es jetzt? Wieder als Kommentar speichern.
 - Kontostände korrekt?

Gib diese Version ab!

1.3 Threads – aber richtig

Damit nicht mehrere Änderungen gleichzeitig passieren:

```
public synchronized void add(int betrag)
```

- Wie lange dauert es jetzt?

1.4 Alternative – Package `UE19.runnable`

Statt unsere Klasse von `Thread` abzuleiten:

- implementieren des Interfaces `Runnable` (braucht Methode `run()`)
- einen `Thread` mit diesem `Runnable` erzeugen

```
public class Ueberweiser implements Runnable {  
    public void run() {  
        ...  
    }  
}  
  
...  
Runnable r = new Ueberweiser(a, b);  
Thread myThread = new Thread(r);  
myThread.start();  
...
```

Implementiere auch diese Variante.

1.5 Wirklich parallel

Änderung:

- `anzahl` auf 1000
- jeder `Ueberweiser` braucht eine kurze Pause nach jedem Schritt (eine Überweisung, ein Schleifendurchgang)
 - `Thread.sleep(1)`

Wie schauen die Laufzeiten jetzt aus?