F.7 Nebenläufigkeit in Java

- Thread-Konzept und Koordinierungsmechanismen sind in Java integriert
  ➥ nicht-orthogonale, nicht-uniforme Sprache bzgl. Nebenläufigkeit
- Erzeugung von Threads über Thread-Klassen
  - Die auszuführenden Methode wird in einer speziellen Klassen, die ein Interface **`Runnable`** implementiert als Methode **`run`** implementiert.
    - Thread erzeugen = “run-Methoden”-Objekt instantiieren, Thread-Objekt instantiieren und dem Konstruktor das "run-Methoden"-Objekt übergeben
    - Thread starten = Methode **`start`** an Thread-Objekt aufrufen, → Methode **`run`** wird nebenläufig ausgeführt
  - Es wird eine Subklasse von Thread definiert, die die **`run`**-Methode redefiniert. Eine Instanz der Subklasse wird erzeugt und an ihr die Methode **`start`** aufgerufen.
    - Thread erzeugen = Objekt der Subklasse instantiieren
    - Thread starten = Methode **`start`** an dem Objekt aufrufen → Methode **`run`** wird nebenläufig ausgeführt
- Beispiel

```
class MyClass implements Runnable {
        public void run()   {
                System.out.println("Hello\n");
        }
}

....
MyClass o1 = new MyClass();     // create object
Thread t1 = new Thread(o1);     // create thread to run in o1

t1.start();                     // start thread

Thread t2 = new Thread(o1);     // create second thread to run in o1

t2.start();                     // start second thread
```


F.7 Nebenläufigkeit und Java (2)

★ **Koordinierungsmechanismen**

- Monitore: exclusive Ausführung von Methoden eines Objekts
  - Methoden oder Code-Blöcke können `synchronized` deklariert werden
  - Alle synchronized-Methoden und Code-Blöcke eines Objekts (einer Instanz, nicht aller Instanzen einer Klassen gemeinsam!) bilden gemeinsam einen Monitor: d.h. es kann immer nur einThread zu einem Zeitpunkt in einer der Methoden bzw. Code-Blöcke aktiv sein.
  - Ein Objekt kann weitere Methoden besitzten, die nicht als synchronized deklariert wurden. Solche Methoden (z.B. Methoden, die nur auf dem Objektzustand lesen, aber keine kritischen, koordinierungsbedürftigen Operationen ausführen) können auch mehrfach parallel ausgeführt werden.

◆ Beispiel:

```
class Bankkonto {
    int value;
    public synchronized void AddAmmount(int v) {
        value=value+v;
    }
    public synchronized void RemoveAmmount(int v) {
        value=value-v;
    }
}
...
Bankkonto b=....
b.AddAmmount(100);
```

- Conditions: gezieltes Freigeben des Monitors und Warten auf ein EreignisInnerhalb eines Monitors kann "wait" aufgerufen werden
  ⇒ Monitor wird freigegeben und Thread legt sich schlafen.
- Jemand anderes kann innerhalb des Monitors "notify" oder "notifyAll" aufrufen
  ⇒ Einer der bzw. alle blockierten Threads werden aufgeweckt

◆

F.7 Nebenläufigkeit und Java (3)

- Beispiel: Implementierung einer Klasse Semaphore

```
class Semaphore {
    private int count;

    public Semaphore (int n) {
        this.count = n;
    }
// ...
```


F.7 Nebenläufigkeit und Java (4)

- … Beispiel: Implementierung einer Klasse Semaphore

```
//...
    public synchronized void P (int n) {
        while((count - n) <= 0) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }

        count -= n;
    }

    public synchronized void V (int n) {
        count += n;
        notify ();
    }
}
```