Verändere die Klasse ZaehlRauf so, dass die Zählvariable i ein Attribut wird.

```
private int i=0;
public void run(){
        for (   ; i<=1000; i++) {…
```

Starte zwei Threads, die gemeinsam bis 1000 zählen.

```
 public void test(){
        ZaehlRauf rauf1 = new ZaehlRauf();
        Thread rauf2 = new Thread(rauf1, "zweiter");
        rauf2.start();
        rauf1.run();
    }
```

```
Thread[main,5,main]: 11
Thread[main,5,main]: 12
Thread[main,5,main]: 13
Thread[main,5,main]: 14
Thread[zweiter,5,main]: 14
Thread[zweiter,5,main]: 16
Thread[zweiter,5,main]: 17
Thread[zweiter,5,main]: 18
Thread[zweiter,5,main]: 19
```

```
Thread[zweiter,5,main]: 300
Thread[zweiter,5,main]: 301
Thread[main,5,main]: 98
Thread[main,5,main]: 303
Thread[main,5,main]: 304
Thread[main,5,main]: 305
Thread[main,5,main]: 306
Thread[main,5,main]: 307
Thread[main,5,main]: 308
Thread[main,5,main]: 309
Thread[zweiter,5,main]: 302
Thread[zweiter,5,main]: 311
Thread[zweiter,5,main]: 312
Thread[zweiter,5,main]: 313
Thread[zweiter,5,main]: 314
Thread[main,5,main]: 310
Thread[main,5,main]: 316
```

Der Thread soll so lange weiter zählen, bis er von einem interrupt (z.B. durch Aufruf der Methode hoerAuf) unterbrochen wird.

```
public void run(){
        Thread t = Thread.currentThread();
        while (!t.isInterrupted()){
            System.out.println( t + ": " + i);
            i++;
        }
}
```

```
public class ThreadTest {
    Thread rauf = new Thread(new ZaehlRauf());

    public void zaehl(){
            rauf.start();
    }

    public void hoerAuf(){
            rauf.interrupt();
    }
}
```

Semaphore:

- unübersichtlich
- viele Fehlermöglichkeiten

➔ **Monitorkonzept** (nutzt intern Semaphore)
(Sir CAR Hoare)

Alle Methoden eines Objekts, deren Vereinbarung mit dem Schlüsselwort synchronized beginnt, bilden einen Monitor. Für ein Objekt kann zu einer bestimmten Zeit nur ein einziger Thread eine oder mehrere dieser zum Monitor gehörigen Methoden ausführen. Die anderen müssen warten.

run-Methode synchronized setzen - Gute Idee?

Zwei Threads sollen nun korrekt gemeinsam (bis 1000) zählen.
Lagere dazu das Hochzählen in eine Methode
public **synchronized** … erhoehen() aus (Monitor).

Zwischenlösung:

```
public void run(){
      while (i<1000){
         erhoehen();
      }
}
public synchronized void erhoehen(){
       System.out.println(Thread.currentThread() + ": " + i);
       i++;
}
```

Tipp zum perfekten Zählen bis 1000:
public synchronized **boolean** erhoehen()

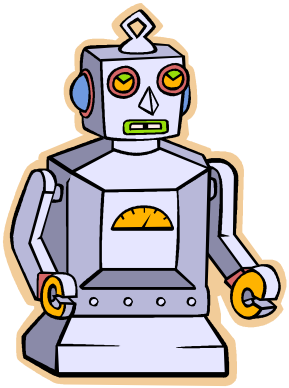
```
public void run(){
      boolean weitermachen = true;
      while (weitermachen){
            weitermachen = erhoehen();
      }
}
```

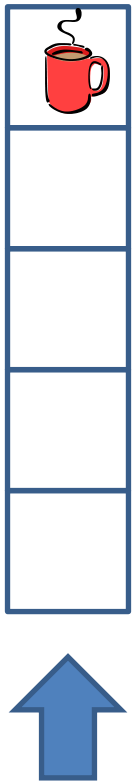
```
public synchronized boolean erhoehen(){
        boolean nichtfertig = true;
        if (i<=1000) {
            System.out.println(Thread.currentThread() + ": " + i);
            i++;
        }
        else nichtfertig = false;
        return nichtfertig;
    }
```

Erzeuger-Verbraucher-Problem

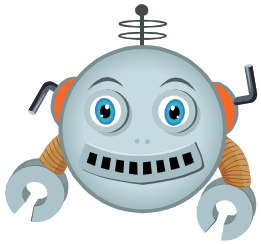
Puffer

Erzeuger





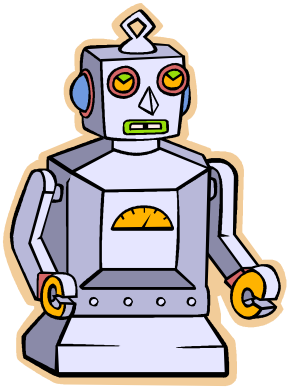
Verbraucher

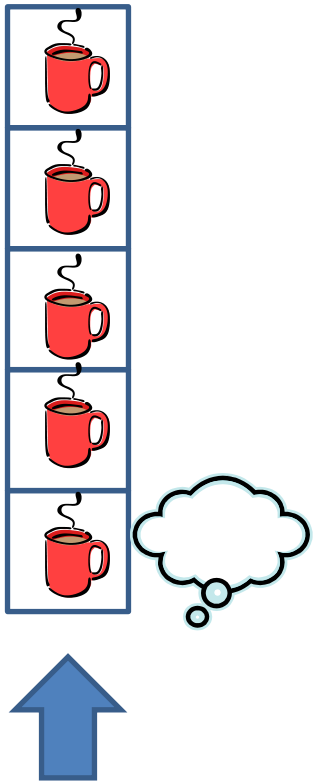


Erzeuger-Verbraucher-Problem

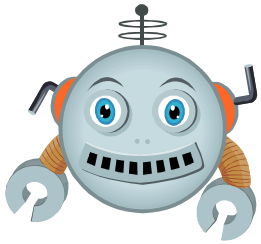
Puffer

Erzeuger





Verbraucher

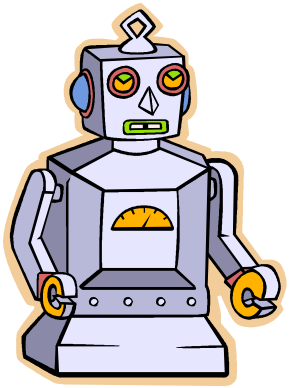


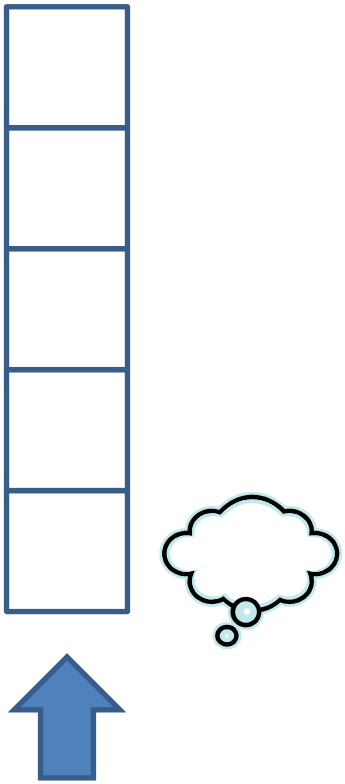
Problem1: Speicher voll, Erzeuger will einfüllen

Erzeuger-Verbraucher-Problem

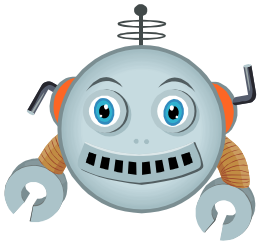
Puffer

Erzeuger





Verbraucher

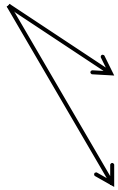


Problem2: Speicher leer, Verbraucher will abholen

Lösung (z.B. Problem2):

V verlässt kritischen Bereich.

- **aktives** Warten:
  wartet, wiederholt Versuch

  lange: Zeit vergeudet

  kurz: evtl. viele Versuche nötig, behindert E

- **passives** Warten: **effizienter!**
  E teilt V mit, dass Lagerplatz wieder besetzt ist.
  Methoden:
  wait(): Monitor freigeben und in Warteschlange einreihen
  notify(): ein auf den aktuellen Monitor wartender Thread wird benachrichtigt

notify

```
public final void notify()
```

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object's monitor by calling one of the `wait` methods.