erster Thread

```
class ThreadA extends Thread {
  public void run() {
    while (true) {
      System.out.println("A");
    }
  }
}

public class ThreadAufruf {
  public static void main(String[] args) {
    ThreadA threadA = new ThreadA();
    threadA.start();
  }
}
```

Thread-Abbruch

```
public class TInterrupt extends Thread {
  int cnt = 0;
  public void run() {
    while (true) {
      System.out.print(cnt + ": ");
      for (int i = 0; i < 30; ++i) {
        System.out.print(i == cnt % 30 ? "* " : ".");
      }
      System.out.println();
    }
  }
  public static void main(String[] args) {
    TInterrupt th = new TInterrupt();
    th.start();
  }
}
```

Abbruch mitten in Zeile!!!

Thread-Abbruch 2

```
public class TInterrupt extends Thread {
  int cnt = 0;
  public void run() {
    while (!isInterrupted()) {
      System.out.print(cnt + ": ");
      for (int i = 0; i < 30; ++i) {
        System.out.print(i == cnt % 30 ? "* " : ".");
      }
      System.out.println();
    }
  }
```

Thread-Abbruch 3

```
public static void main(String[] args) {
   TInterrupt th = new TInterrupt();
   th.start();
   try {
      Thread.sleep(2000);
   } catch (InterruptedException e) {
   }
   th.interrupt();
}
```

erster Thread als Impl. von Runnable

```
class ThreadA implements Runnable {
  public void run() {
    while (true) {
      System.out.println("A");
    }
  }
}

public class ThreadAufruf {
  public static void main(String[] args) {
    Thread threadA = new Thread(new ThreadA());
    threadA.start();
  }
}
```

allgemeiner Thread entspricht ThreadA

Synchronisation

```
public class ZählerFolie6 extends Thread {
  static int cnt = 0;

  public void run() {
    while (true) {
      for (int i = 0; i < 1000000; i++) {
        cnt++;
        cnt--;
      }
      System.out.println(cnt++);
    }
  }
  public static void main(String[] args) {
    (new ZählerFolie6("t1: ")).start();
    (new ZählerFolie6("t2: ")).start();
  }
}
```

Synchronisation einer Methode

```
class Counter {

  private int cnt = 0;

  public  synchronized  int nextNumber() {
    int ret = cnt;

//ein paar zeitaufwändige Berechnungen
// ...

    cnt++;
    return ret;
  }
}
```

Synchronisation eines Blocks

```
public class Zähler extends Thread {
  static int cnt = 0;

  public void run() {
    while (true) {
      synchronized (getClass()) {
        for (int i = 0; i < 1000000; i++) {
          cnt++;
          cnt--;
        }
        System.out.println(cnt++);
      }
    }
  }
```

beschafft Klassenobjekt als Monitor
zum Schutz von Klassenattribut cnt

Producer

```
class Producer extends Thread {
  private Vector v;
  public Producer(Vector v) {
    this.v = v;
  }

  public void run() {
    String s;
    while (true) {
      synchronized (v) {
        s = "Wert "+Math.random();
        v.addElement(s);
        v.notify();
      }
    }
  }
}
```

Monitor

Block

Mitteilen

Consumer

```
class Consumer extends Thread {
  private Vector v;
  public Consumer(Vector v) {
    this.v = v;
  }
  public void run() {
    while (true) {
      synchronized (v) {
        if (v.size() < 1) {
          try {
            v.wait();
          } catch (InterruptedException e) {
          }
        }
        v.removeElementAt(0);
      }
    }
  }
}
```

Producer/Consumer-main

```
public class PCMain {

  public static void main(String[] args) {

    Vector v = new Vector();

    Producer p = new Producer(v);
    Consumer c = new Consumer(v);

    c.start();
    p.start();
  }
}
```

Monitor erzeugen

Monitor übergeben

Threads starten