Warten durch *join*():

```
static void test2() {
  List<Thread> threads = new ArrayList<Thread>();
  for(int i = 1; i <= 10; i++) {
    ZahlenZeiger z = new ZahlenZeiger(i);
    Thread t = new Thread(z);
    t.start();
    threads.add(t);
  }
  try {
    for(Thread t : threads) {t.join();}
  }
  catch(InterruptedException e) {System.out.println(e);}
}
```

join() wartet, bis der Thread endet.

Unterklasse von *Thread* statt *Runnable*:

```
class QueensSolver extends Thread {
  private int n; // Groesse des Bretts
  private volatile Brett brett;
  private volatile boolean finished = false;
  public QueensSolver(int n) {this.n = n;}
  public void run() {
    brett = new RandomBrett(n);
    brett.setzeDamenAbZeile(0);
    finished = true;
  }
  public void printSolution() {brett.print();}
  public boolean finished() {return finished;}
  public void finish() {brett.geloest = true;}
}
```

```
static void test3() {
  final int n = 250;
  List<QueensSolver> solvers = new ArrayList<QueensSolver>();
  for(int i = 0; i < 4; i++) {
    QueensSolver solver = new QueensSolver(n);
    solver.start(); solvers.add(solver);
  }
  while(true) { // Polling!
    for(QueensSolver solver : solvers) {
      if(solver.finished()) {
        solver.printSolution();
        for(QueensSolver s : solvers) {s.finish();}
        return;
      }
    }
  }
}
```

Überblick

6 Nebenläufigkeit

- Threads
- Synchronisation

Es ist schwer, Daten konsistent zu halten, wenn mehrere Threads auf sie zugreifen:

```
class WortPaarVersuch implements Runnable {
  private String deutsch;
  private String englisch;
  public void run() {
    while(true) {
      setNames("Hallo","hello");
      setNames("Hund","dog");
      setNames("Katze","cat");
    }
  }
  public void setNames(String d,String e) {
    deutsch = d; englisch = e;
  }
  public String getNames() {
    return deutsch + "=" + englisch;
  }
}
```

Deklarieren wir Methoden **synchronized**, dann können keine von ihnen zeitlich überlappt ablaufen.

```
class WortPaar implements Runnable {
  private volatile String deutsch;
  private volatile String englisch;
  public void run() {
    while(true) {
      setNames("Hallo","hello");
      setNames("Hund","dog");
      setNames("Katze","cat");
    }
  }
  public synchronized void setNames(String d,String e) {
    deutsch = d; englisch = e;
  }
  public synchronized String getNames() {
    return deutsch + "=" + englisch;
  }
}
```

Neues Kapitel

Fehlerbehandlung

Unter einem Laufzeitfehler verstehen wir eine aussergewöhnliche Situtation, die normalerweise nicht auftreten sollte.
Beispiel:
Wir öffnen eine Datei, die eigentlich existieren muß.

```
DataInputStream stream = null;
try {
  stream = new FileInputStream("dateiname");
}
catch(FileNotFoundException exception) {
  ... // Fehlerbehandlung, z.B. Abbruch mit Fehlermeldung
}
... // Lese aus der Datei etc.
```

Gegenbeispiel:
Wie lesen Instruktionen aus einer optionalen Datei.
Falls diese nicht existiert, liegt kein Fehler vor.

Beispiel: Menschen, die heiraten können.

```
class Mensch {
  private int alter;
  private String name;
  private Mensch partner;
  public Mensch(String name, int alter) {
    this.alter = alter;
    this.name = name;
    this.partner = null;
  }
  public int getAlter() {return alter;}
  public String getName() {return name;}
  public Mensch getPartner() {return partner;}
```

Mit **throw** können wir ein Objekt vom Typ *Exception* „werfen“, welches von jemanden mit **catch** gefangen werden muß.

Annahme: Heiraten ist nur erlaubt, wenn das Durchschnittsalter mindestens 18 beträgt.

```
public void heiraten(Mensch partner) throws ZuJungException {
  if(this.alter + partner.alter < 36) {
    throw new ZuJungException();
  }
  this.partner = partner;
}
```

Für das geworfene Objekt erstellen wir eine eigene Klasse:

```
class ZuJungException extends Exception {}
```

So können wir die Methode *heiraten* jetzt aufrufen:

```
public static void main(String args[ ]) {
   Mensch karl = new Mensch("Karl Ranseier",17);
   Mensch sue = new Mensch("Sue Winter",17);
   try {
      karl.heiraten(sue);
   } catch (ZuJungException e) {
      System.out.println("Zu jung. Sie haben nicht geheiratet.");
   }
}
```