Beispiel

Wir berechnen die größte Zahl in einem Array.

```
static int maximum(int a[ ]) throws EmptyArrayException {
  int n = a.length;
  if(n == 0) {
    throw new EmptyArrayException();
  }
  int max = a[0];
  for(int i = 0; i < n; i++) {
    if(max < a[i]) {
      max = a[i];
    }
  }
  return max;
}
```

Neues Kapitel

Refactoring

Ein (komplexes) Programm muß sich während seiner Lebenszeit Erweiterungen und Veränderungen unterwerfen.

Oft zeigt sich bei einer Erweiterung, daß die derzeitige Struktur ungeeignet ist.

Bestes Vorgehen:

1. Die Struktur des Programms verändern.
2. Die neue Funktionalität hinzufügen.

Den ersten Schritt nennen wir *Refactoring*.

Die Struktur des Programms wird verbessert, ohne Neues hinzuzufügen.

Typische Refactoringschritte (es gibt viel mehr):

- Doppelten Code durch einfachen ersetzen.
- Zu lange Methoden durch kürzere ersetzen.
- Lokale Variablen abschotten (getter und setter).
- Funktionalität verallgemeinern für besseres Wiederbenutzen.
- Neue Unterklassen einführen.
- Unterklassen zusammenfassen.
- Variablen oder Methoden in Ober- oder Unterklassen verschieben.
- Variablen oder Methoden in andere Klassen verschieben.
- . . . und sogar einfache Umbenennungen in sinnvollere Namen.

Neues Kapitel

Wie können wir Daten dauerhaft speichern?

Häufigste Methoden:

- In einer Datei speichern.
- In einer Datenbank speichern.

Lesen aus einer Datei:

```
double readDouble(String name) {
  FileInputStream is = null;
  DataInputStream dataStream = null;
  double result;
  try {
    is = new FileInputStream(name);
    dataStream = new DataInputStream(is);
    result = dataStream.readDouble();
  }
  catch (Exception e) {e.printStackTrace();}
  return result;
}
```

Serializable

Implementiert eine Klasse das Interface *Serializable*, dann kann ihr Inhalt direkt in eine Datei geschrieben und aus ihr gelesen werden.

```
class Person implements Serializable {
  private String vorname;
  private String nachname;
  public Person(String vorname, String nachname) {
    this.vorname = vorname;
    this.nachname = nachname;
  }
  public String toString() {
    return vorname + "  " + nachname;
  }
}
```

So schreiben wir eine *Person* in eine Datei:

```
Person karl = new Person("Karl","Ranseier");
try {
  FileOutputStream stream = new FileOutputStream("karl");
  ObjectOutputStream objstream =
    new ObjectOutputStream(stream);
  objstream.writeObject(karl);
  stream.close();
}
catch(IOException e) {
  System.out.println("Oje: " + e);
}
```

So lesen wir eine *Person*:

```
Person unbekannt = null;
try {
  FileInputStream stream = new FileInputStream("karl");
  ObjectInputStream objstream =
    new ObjectInputStream(stream);
  unbekannt = (Person) objstream.readObject();
  stream.close();
}
catch(ClassNotFoundException e) {
  System.out.println("Falsche Klasse gelesen: " + e);
}
catch(IOException e) {
  System.out.println("Oje: " + e);
}
System.out.println(unbekannt);
```