JAVA und die grafische Benutzeroberfläche

GUI-Objekte und ihre Anordnung

Alfred Nussbaumer

Nicht alle Aktionen, die „mit der Maus ausgeführt“ werden, sind an MouseEvents gebunden (vgl. [5]). Für den Klick auf eine Schaltfläche verwenden wir beispielsweise einen ActionListener, einen ItemListener für die Auswahl eines Auswahlkästchens oder einen AdjustmentListener für die Position eines Scrollbalkens. Weiters stellt sich die Frage, wie die GUI-Objekte am Bildschirm – oder besser gesagt – im jeweiligen Anwendungsfenster positioniert werden können. Während in früheren Artikeln für GUI-Ausgaben meistens das AWT verwendet wurde, soll in diesem Artikel „Swing“ verwendet werden.

1. AWT und Swing

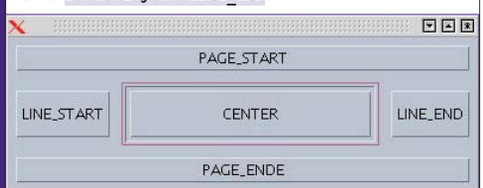
Das Abstract Window Toolkit erlaubt das Programmieren von GUI-Anwendungen. Zahlreiche Grafikobjekte, eine stabile Ereignisbehandlung und Layoutmanager sind Kennzeichen dieser Klassensammlung. Swing erweitert diese Komponenten um zahlreiche weitere Objekte und ermöglicht das so genannte Pluggable Look-and-Feel. Damit ist es möglich, dass eine Anwendung verschiedenes Aussehen haben kann, etwa das „Java“-, „Metal“-,„Windows“-, „GTK“- oder „Motif“-Look-and-Feel. Andererseits ist dabei die Darstellung der Objekte nicht durch die nativen Möglichkeiten des Betriebssystems eingeschränkt. Eine detaillierte Beschreibung aller Swing-Objekte ist beim Hersteller online erreichbar ([2], [3]). Aktuelle JAVA-Anwendungen sind im Allgemeinen mit Swing realisiert.

2. Layout, Look & Feel und Komponenten

GUI-Objekte werden grundsätzlich dem freien Platz entsprechend der Reihe nach im Anwendungsfenster platziert (*„FlowLayout“*). Um sie an eine bestimmte Stelle zu heften, verwendet man verschiedene Layouts, von denen wir hier zwei verwenden werden, das BorderLayout und GridLayout.

Mit dem BorderLayout können 5 Bereiche eines Fensters verwaltet werden, die mit den folgenden BorderLayout-Konstanten adressiert werden:

`BorderLayout.PAGE_START`, `BorderLayout.PAGE_END`, `BorderLayout.CENTER`, `BorderLayout.LINE_START` und `BorderLayout.LINE_END`.



Die Definitionen eines Layouts stehen im Konstruktor der jeweiligen Klasse. Für das obige Beispiel sind dies folgende Angaben:

```
public BL() {
   Container c = getContentPane();
   c.setLayout(new BorderLayout(3,3));
   JButton north = new JButton("PAGE_START");
   JButton east = new JButton("LINE_END");
   JButton south = new JButton("PAGE_ENDE");
   JButton west = new JButton("LINE_START");
   JButton center = new JButton("CENTER");
   c.add(north, BorderLayout.PAGE_START);
   c.add(east, BorderLayout.LINE_END);
   c.add(south, BorderLayout.PAGE_END);
   c.add(west, BorderLayout.LINE_START);
   c.add(center, BorderLayout.CENTER);
}
```

Die Parameter in `BorderLayout(3,3)` legen den frei bleibenden Abstand zwischen den Bereichen fest. Der Abstand von 3 Pixeln wird im Beispiel noch um den Randbereich erweitert, den die Button-Objekte im „Motif“-Look&Feel von vornherein aufweisen.

Das GridLayout erlaubt, die GUI-Objekte in Reihen und Spalten anzuordnen. So legt die Definition

```
c.setLayout(new GridLayout(2,3,15,5));
```

ein GridLayout fest, das die Objekte in 2 Reihen zu je 3 Feldern anordnet. Zwischen den Feldern ist ein freier Abstand von 15 Pixel, zwischen aufeinander folgenden Reihen ein Abstand von 5 Pixel:


```
public GL() {
   Container c = getContentPane();
   c.setLayout(new GridLayout(2,3,15,5));
   JButton button1 = new JButton("Button1");
   JButton button2 = new JButton("Button2");
   JButton button3 = new JButton("Button3");
   JButton button4 = new JButton("Button4");
   JButton button5 = new JButton("Button5");

   c.add(button1);
   c.add(button2);
   c.add(button3);
   c.add(button4);
   c.add(button5);
}
```

Layouts können miteinander kombiniert werden: Um in einem Bereich eines Layouts ein anderes Layout verankern zu können, verwendet man so genannte Panels.

Eine Besonderheit von Swing stellt das Look and Feel dar. Damit kann das Aussehen einer Anwendung leicht an das Betriebssystem oder an den Windowmanager angepasst werden, in dessen Umgebung sie eingebettet ist. Geschieht dies zur Laufzeit, spricht man auch vom *„Pluggable Look & Feel“*, PLAF). Die Objekte sind jeweils gleich angeordnet, nur ihre Darstellung, z.B. die Textgestaltung oder die Markierung der gedrückten Schaltfläche hängen von der jeweiligen Look&Feel-Klasse ab:

Java Basic

Java GTK

Java Motif

Java Motif, Variante

Von den zahlreichen verschiedenen Swing-Komponenten sollen hier nur einige besprochen werden:

- `JLabel` – dient zur Darstellung eines Textes.
- `Button` – dient zur Darstellung von Schaltflächen.
- `CheckBox` – spezieller Button, dient zur Eingabe von Wahrheitswerten. Sein Wahrheitswert ändert sich mit jedem Anklicken.
- `RadioButton` – wie `JCheckBox`, allerdings können RadioButtons zu einer Gruppe zusammengefasst werden: Innerhalb dieser Gruppe kann jeweils nur ein Button ausgewählt werden (d.h. Er liefert den Wahrheitswert `„true"`).

3. Ereignisquellen und Listener

Verschiedene GUI-Komponenten können bestimmte Ereignisse auslösen. Sie bilden eine so genannte Ereignisquelle. Alle Ereignisse sind in bestimmte Ereignisklassen eingeteilt, die jeweils von einem bestimmten Interface (Listener) entgegengenommen werden (vgl. [4], [5]). Für die Beispiele verwenden wir folgende Events:

- `ActionEvent`, wird beispielsweise durch `JButton`, `JCheckBox`, `JTextField`, `JmenuItem` ausgelöst.
- `ItemEvent`, wird beispielsweise durch `JCheckbox`, `JCheckboxMenuItem` ausgelöst.
- `AdjustmentEvent`, beschreibt `JScrollBar`-Ereignisse.

Die zugehörigen Interfaces müssen jeweils implementiert werden. Damit die GUI-Objekte Ereignisse auslösen können, müssen sie die Verbindung mit dem gewünschten Interface mit einer `add`-Methode herstellen, beispielsweise:

```
JCheckBox chG = new JCheckBox("Gitter",true);
chG.addActionListener(this);
```

In diesem Fall sendet das `JCheckBox`-Objekt `chG` Action-Events an einen ActionListener. Die entsprechende Listener- Methode zur Ereignisbehandlung, `actionPerformed()`, löst hier eine Neuausgabe des Fensterinhaltes aus:

```
public void actionPerformed(ActionEvent e) {
    repaint();
}
```

Der Status der Auswahlbox `chG` wird beispielsweise innerhalb der `paint()`-Methode mit der Funktion `isSelected()` ermittelt:

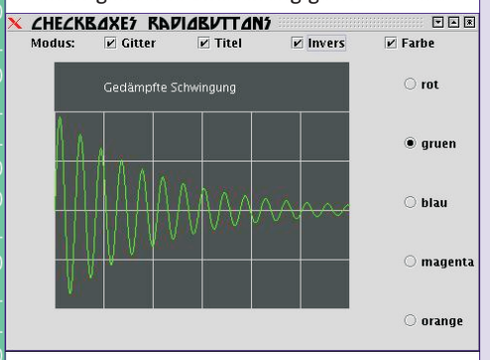
```
if (chG.isSelected()) {
    ...
}
```

Die besprochene, elementare Vorgangsweise ermöglicht einfache Anwendungen mit Benutzereingaben über Auswahlfelder, Schaltflächen und Schieberegler.

4. Beispiel: Ausgabe eines Funktionsgraphen

Eine vorgegebene Funktion wird in einem rechteckigen Bereich ausgegeben. In der Anwendung kann ein bestimmter Ausgabemodus durch entsprechende Schaltflächen gewählt werden. Da verschiedene Merkmale kombiniert werden, sind dafür `JCheckBox`-Ob-

jekte vorgesehen. Für die Zeichenfarbe ist nur eine eindeutige Auswahl sinnvoll; hier werden `JRadioButton`-Eingaben verwendet. Der Bildschirm wird jedes Mal neu ausgegeben, wenn irgendeine Einstellung geändert wird:



Der Programmcode ist durch zwei Teile gekennzeichnet: Im Konstruktor werden ein bestimmtes Layout und alle Schaltflächen definiert und angeordnet. Die Ausgabe der Funktion wird durch die `paint()`-Methode geregelt, in der alle erforderlichen Grafikobjekte definiert werden.

Im Konstruktor legt man zunächst den so genannten Container fest, der alle weiteren Objekte enthalten soll. Im Beispiel ist dies der Container `c`. Eine korrekte Referenz darauf liefert die Methode `getContentPane()`. Mit der `add`-Methode können alle weiteren Panels auf die ContentPane „geheftet" werden. Für jedes Panel wird ein eigenes Layout bestimmt. In seine Bereiche werden letztlich alle Komponenten eingefügt.

Um das Zusammenspiel von GUI-Objekten und der Ereignisbehandlung verfolgen zu können, ist der Programmcode des Beispiels hier vollständig angegeben: Jedes GUI-Objekt ist an den ActionListener gebunden. Daher löst jede Änderung eines Auswahlfeldes ein Neuzeichnen der Funktion aus. Innerhalb der `paint()`-Methode wird mit der `isSelected()`-Methode überprüft, welche Schaltfläche ausgewählt wurde.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Plot extends JFrame
   implements ActionListener {

  JCheckBox chF;
  JCheckBox chB;
  JCheckBox chG;
  JCheckBox chD;
  JRadioButton chr;
  JRadioButton chb;
  JRadioButton chg;
  JRadioButton chm;
  JRadioButton cho;

  public Plot() {
    Container c = getContentPane();
    c.setLayout(new BorderLayout());

    JLabel hinweis = new JLabel
      ("Modus:");

    JPanel grafik = new JPanel();
    grafik.setPreferredSize
      (new Dimension(400, 300));
    c.add(grafik, BorderLayout.CENTER);

    JPanel modus = new JPanel();
    c.add(modus, BorderLayout.PAGE_START);
    modus.setLayout(new GridLayout(1,5));

    JPanel farbe = new JPanel();
    c.add(farbe, BorderLayout.LINE_END);
    farbe.setLayout(new GridLayout(5,1));

    chG = new JCheckBox("Gitter",true);
    chB = new JCheckBox("Titel",true);
    chD = new JCheckBox("Invers", false);
    chF = new JCheckBox("Farbe",true);

    chG.addActionListener(this);
    chB.addActionListener(this);
    chD.addActionListener(this);
    chF.addActionListener(this);

    modus.add(hinweis);
    modus.add(chG);
    modus.add(chB);
    modus.add(chD);
    modus.add(chF);

    ButtonGroup Farbgruppe = new ButtonGroup();
    chr = new JRadioButton("rot", false);
    chg = new JRadioButton("gruen", false);
    chb = new JRadioButton("blau", true);
    chm = new JRadioButton("magenta", false);
    cho = new JRadioButton("orange", false);

    Farbgruppe.add(chr);
    Farbgruppe.add(chg);
    Farbgruppe.add(chb);
    Farbgruppe.add(chm);
    Farbgruppe.add(cho);

    chr.addActionListener(this);
    chg.addActionListener(this);
    chb.addActionListener(this);
    chm.addActionListener(this);
    cho.addActionListener(this);

    farbe.add(chr);
    farbe.add(chg);
    farbe.add(chb);
    farbe.add(chm);
    farbe.add(cho);
  }

  double f(double x) {
    return 200 -
      2 * Math.pow(2.71828, -0.01 * x)
          * Math.sin(0.3 * x) * 50;
  }

  public void actionPerformed(ActionEvent e) {
    repaint();
  }

  public void paint (Graphics g) {
    super.paint(g);
    if (chD.isSelected())
   g.setColor(Color.darkGray);
    else g.setColor(Color.white);
    g.fillRect(50,50, 300,250);
    g.setColor(Color.black);

    if (chG.isSelected()) {
      if (chD.isSelected())
        g.setColor(Color.lightGray);
      for (int i=1;i<8;i++) {
        g.drawLine(i*50,100,i*50,300);
      }
      for (int i=2;i<7;i++) {
        g.drawLine(50,i*50,350,i*50);
      }
    }

    if (chB.isSelected()) {
      if (chD.isSelected())
        g.setColor(Color.white);
      g.drawString("Gedämpfte
Schwingung",100,80);
    }

    if (chF.isSelected()) {
      if (chr.isSelected())
g.setColor(Color.red);
      if (chb.isSelected())
        g.setColor(Color.blue);
      if (chg.isSelected())
        g.setColor(Color.green);
      if (chm.isSelected())
        g.setColor(Color.magenta);
      if (cho.isSelected())
        g.setColor(Color.orange);
    }

    for (int x=0;x<300;x++) {
      g.drawLine
        (x+50,(int)f(x),x+51,(int)f(x+1));
    }
  }

  public static void main(String args[]) {
    Plot proggi = new Plot();
    proggi.addWindowListener
      (new WindowAdapter() {
      public void windowClosing
        (WindowEvent evt) {
          System.exit(0);
      }
    });
    proggi.setTitle
      ("CheckBoxes & RadioButtons");
      proggi.setSize(500, 300);
      proggi.setLocation(100,100);
      proggi.pack();
      proggi.show();
  }
}
```

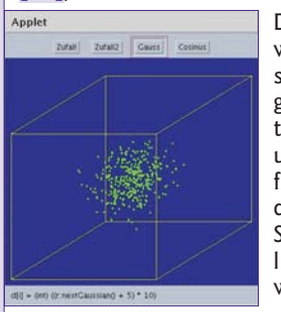
Auf eine Besonderheit soll hier extra hingewiesen werden: Alle GUI-Objekte werden unter Swing mit Hilfe einer `paint()`-Methode ausgegeben. Da bei der Grafikausgabe genau diese `paint()`-Methode neu aufgerufen wird, erhält man schließlich alle eigenen Zeichnungen, aber von den ursprünglichen Komponenten ist wenig bis gar nichts mehr zu sehen. Die Lösung liegt darin, die ursprüngliche `paint()`-Methode in der überschriebenen `paint()`-Methode mit Hilfe der Anweisung

```
super.paint(g);
```

aufzurufen. Dazu setzt man einfach diese Befehlszeile an den Anfang der eigenen `paint()`-Methode.

5. Beispiel: Lorenz-Plot

Zahlenreihen können auf vielfältige Weise miteinander verglichen werden. Eine interessante Möglichkeit besteht darin, jeweils drei aufeinander folgende Werte als die drei Raumkoordinaten eines Punktes aufzufassen. Stellt man diese Punkte etwa in einem Schrägriss dar, so sieht man, wie stark aufeinander folgende Werte voneinander abweichen (vgl. Auswertung eines Langzeit-EKG, **[12]**).



Der Programmcode wird hier nur ausschnittweise angegeben (der gesamte Quelltext steht unter **[13]** zur Verfügung, insbesonders kann dort die Schrägrissdarstellung nachgelesen werden).

Im Konstruktor wird neben den erforderlichen Schaltflächen auch die gewünschte *„Look-and-Feel"*-Klasse (LAF-Klasse, hier Motif) festgelegt: Innerhalb eines `„try-catch"`-Blockes versucht die Methode `UIManager.setLookAndFeel()` die LAF-Klasse zu laden. Anschließend werden alle Grafikelemente in der gewünschten Form mit `SwingUtilities.updateComponentTreeUI()` initialisiert.

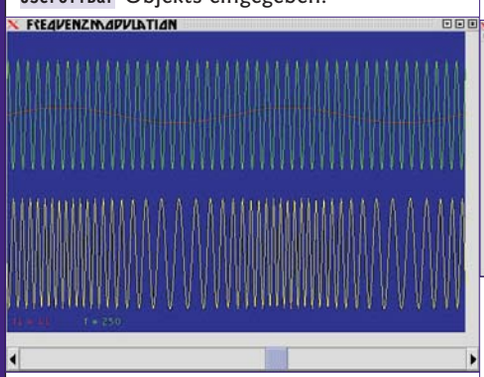
```
import java.awt.*;
import java.awt.event.*;
import java.util.Random;
import javax.swing.*;

public class Zahlenreihe extends JApplet
   implements ActionListener {

  private int d[] = new int[3000];
```

```
  private Random r = new Random();
  private JButton Zufall;
  private JButton Zufall2;
  private JButton Gauss;
  private JButton Cosinus;
  private String reihe =
    "Zahlenwerte d[i] < 100,
     i zwischen 0 und 3000";

  public void init() {
    Container c = getContentPane();
    Zufall = new JButton("Zufall");
    Zufall2 = new JButton("Zufall2");
    Gauss = new JButton("Gauss");
    Cosinus = new JButton("Cosinus");
    Zufall.addActionListener(this);
    Zufall2.addActionListener(this);
    Gauss.addActionListener(this);
    Cosinus.addActionListener(this);
    c.setLayout(new FlowLayout());
    c.add(Zufall);
    c.add(Zufall2);
    c.add(Gauss);
    c.add(Cosinus);

    try {
      UIManager.setLookAndFeel
        ("com.sun.java.swing.plaf.motif.
          MotifLookAndFeel");
    }
    catch ( Exception e ) {
      System.err.println(e.toString());
    }
    SwingUtilities.updateComponentTreeUI(this);

  }
  ...
  public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == Zufall)
      zahlenreihe(1);
    if (ae.getSource() == Zufall2)
      zahlenreihe(2);
    if (ae.getSource() == Gauss)
      zahlenreihe(3);
    if (ae.getSource() == Cosinus)
      zahlenreihe(4);
    repaint();
  }
  ...
```

Sobald eine Schaltfläche gedrückt wurde, stellt die Methode `actionPerformed()` fest, von welcher Quelle das Ereignis stammt. Davon abhängig wird das Datenfeld `reihe` mit entsprechenden Werten initialisiert. Abschließend wird noch die `repaint()`-Methode aufgerufen, die alle Objekte, den Würfel und alle Datenpunkte ausgibt.

6. Beispiel: Schieberegler verwenden

Im folgenden Beispiel soll die Periodenlänge einer Sinuskurve sinusförmig variiert werden (vgl. „Frequenzmodulation“). Die erste Sinusfunktion („Trägerschwingung“), in der folgenden Abbildung grün gezeichnet, hat eine feststehende Periodenlänge. Eine zweite, rot dargestellte Sinusfunktion hat eine deutlich kleinere Amplitude. Ihre Frequenz zwischen 0 und 14 wird mit Hilfe eines `JScrollBar`-Objekts eingegeben.



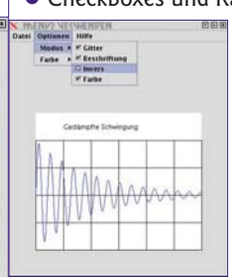
Man beachte den horizontal liegenden Schieberegler unterhalb der resultierenden, gelb gezeichneten Schwingung! Seine Stellung wird mit einem `AdjustmentListener` an die Anwendung mitgeteilt.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JFM extends JFrame {

  JScrollBar sb1;
  int f1;

  public JFM() {
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    JPanel panel = new JPanel();
    panel.setPreferredSize
      (new Dimension(600,30));
    c.add(panel, BorderLayout.PAGE_END);

    JPanel grafik = new JPanel();
    grafik.setPreferredSize(
        new Dimension(600, 400));
    c.add(grafik, BorderLayout.PAGE_START);
        aktion.pdf
    panel.setLayout(new GridLayout(1,1));

    sb1 = new JScrollBar
      (JscrollBar.HORIZONTAL,0, 1, 0, 20);
    sb1.addAdjustmentListener(
      new AdjustmentListener() {
        public void adjustmentValueChanged
            (AdjustmentEvent ade) {
        f1 = ade.getValue();
        repaint();
      }
    });
    panel.add(sb1);
  }

  public void paint(Graphics bs) {
    super.paint(bs);
    /* Ausgabe aller Funktionen */
    ...
    }

    public static void main(String args[]) {
        JFM proggi = new JFM();
        proggi.setTitle("Frequenzmodulation");
        proggi.setSize(600,450);
        proggi.setLocation(100,100);
        proggi.pack();
        proggi.show();
    }
}
```

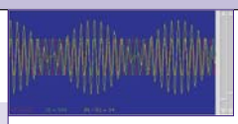
Der komplette Quelltext kann von der Webseite des Autors bezogen werden ([13]).

7. Ausblick, Übungen

Eine Übersicht über Swing-Objekte geht über einen einzelnen Artikel weit hinaus. In der Java-Dokumentation von SUN ([1]), in den dazu gehörenden Tutorials ([2], [3]) und in einer Reihe ausgezeichneter Webseiten sind zahlreiche Details, auch in praktischen Beispielen, angegeben. Einige mögliche Erweiterungen der hier beschriebenen Beispiele sind in folgenden Anregungen zusammengefasst:



- CheckBoxes und RadioButtons auf der Anwendungsfläche sind nicht zumutbar. Es ist besser an ihrer Stelle Menüleisten zu verwenden. Ihre Einträge werden mit Ereignissen zum Ausgeben der Funktion gebunden. Im Vergleich zu dem im Artikel angegebenen Beispiel ist daher nur der Konstruktor anzupassen.
- Zwei Sinuskurven sollen addiert werden (Schwebungen, Amplitudenmodulation). Bei gleichbleibender Amplitude sollen ihre Frequenzen mit Hilfe zweier senkrecht stehender Schieberegler gewählt werden:



Die gewählten Frequenzwerte und ihre Differenz werden als Zeichenketten ausgegeben.

- Die Schrägrissdarstellung des Lorenz-Plots soll mit Hilfe eines horizontalen und vertikalen Schiebereglers gedreht werden können.
- Statt `JScrollBar` soll `JSlider` verwendet werden.
- Die Programmierung von GUI-Elementen wird zunehmend komplexer. Es ist daher üblich, GUI-Anwendungen mit einer geeigneten Entwicklungsumgebung zu erstellen.
- Für umfangreichere Anwendungen sind zusätzliche Elemente (Werkzeugleisten, Kontextmenüs, Datei-, Druck-, Farbauswahl- und Bearbeitungsdialoge, Tooltips und Online-Hilfe, etc.) erforderlich.
- Für bestimmte Anwendungen sind spezielle Swing-Komponenten entwickelt worden. Tabelleninhalte werden beispielsweise mit `JTable`-Objekten, XML-Dateiinhalte oder Verzeichnis-Bäume mit `JTree`-Objekten ausgegeben.
- Der jeweils sichtbare Bereich einer Anwendung wird mit Hilfe einer speziellen Fenstertechnik gesteuert.

8. Literatur, Weblinks

[1] *http://java.sun.com/j2se/1.4.2/docs/index.html* (Dokumentation aller verfügbaren Packages)

[2] *http://java.sun.com/docs/books/tutorial/* (Tutorials zu JAVA, auch in ZIP-Dateien zum Download erhältlich)

[3] *http://java.sun.com/docs/books/tutorial/uiswing/* (Tutorial zu Swing)

[4] PCNEWS-92, S. 24 („JAVA und die grafische Benutzeroberfläche, KeyEvents“)

[5] PCNEWS-93, S. 40 („JAVA und die grafische Benutzeroberfläche, Mouse-Events“)

[6] *http://www.javareference.com/* (Zahlreiche Anleitungen zu verschiedenen JAVA-Themen, JSP-Referenz)

[7] **Guido Krüger**, „Handbuch der JAVA-Programmierung“, Addison & Wesley, ISBN 3-8273-2201-4

[8] *http://www.javabuch.de* („Handbuch der JAVA-Programmierung“, freier Download für schulische Zwecke)

[9] **Herbert Schildt**, „JAVA – Grundlagen der Programmierung“, mitp, IBSN 3-8266-1524-7

[10] **Udo Müller**, „JAVA – das Lehrbuch“, mitp, ISBN 3-8266-1333-3

[11] **Christian Ullenboom**, „Java ist auch eine Insel“, Galileo Computing, ISBN 3-89842-365-4

[12] *http://www.heise.de/ct/01/01/036/*, (c't 2001/1, „Forschung“ - Beschreibung des Lorenz-Plots in Zusammenhang mit der Untersuchung von Herz-Rhythmus-Störungen).

[13] *http://www.gymmelk.ac.at/nus/informatik/wpf/JAVA* (Unterrichtsbeispiele zum Programmieren mit JAVA, Quelltexte zum Downloaden)