HANSER

Grundkurs Programmieren in Java

Dietmar Ratz, Detlef Seese, Jan Wiesenberger, Jens Scheffler

Band 2: Einführung in die Programmierung kommerzieller Systeme

ISBN 3-446-40494-5

Leseprobe

Kapitel 10

Applets

Nachdem wir in Kapitel 5 gelernt haben, Programme zu entwickeln, die grafische Benutzungsoberflächen besitzen, werden wir uns in diesem Kapitel damit beschäftigen, solche grafische Oberflächen nicht nur lokal, sondern über das Internet zur Verfügung zu stellen. Dazu werden wir unsere Programme nicht mehr als Frames, sondern als **Applets** schreiben. Diese können über Computer-Netzwerke verbreitet und innerhalb eines Browsers (wie zum Beispiel *Netscape Navigator* oder *Microsoft Internet Explorer*) ausgeführt werden.
Prinzipiell unterscheidet sich der Aufbau einer grafischen Oberfläche in einem Applet nicht von dem in einem Frame. Einzige Besonderheit beim Applet ist, dass es kein eigenes Fenster bereitstellt. Ein Applet kann daher nicht direkt ausgeführt werden und muss in der Umgebung eines Browsers oder des Appletviewers (ein Hilfsprogramm aus dem JDK) ablaufen, die den Fensterrahmen dafür liefert. Dazu muss das Applet in eine HTML-Datei eingebettet werden. **HTML** steht für **Hyper Text Markup Language** und ist eine Seitenbeschreibungs-Sprache, die zur Darstellung von Webseiten eingesetzt wird.

10.1 Erstellen und Ausführen von Applets

In diesem Abschnitt erläutern wir zunächst anhand des einfachen Beispiels aus Abschnitt 5.4, wo die grundsätzlichen Unterschiede beim Erstellen und Ausführen von Applikationen bzw. Applets liegen.

10.1.1 Vom Frame zum Applet am Beispiel

Wir wollen uns daran erinnern, wie wir in Abschnitt 5.4 unser erstes einfaches Fenster mit einem kleinen Text ausgestattet haben. Unser Java-Programm haben wir damals wie folgt geschrieben:

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /** Erzeuge ein einfaches Swing-Fenster mit einem Textlabel */
5  public class FrameMitText extends JFrame {
6    Container c;          // Container dieses Frames
7    JLabel beschriftung;  // Label, das im Frame erscheinen soll
8
9    // Konstruktor fuer unseren Frame mit Textlabel
10   public FrameMitText() {
11     // Bestimme die Referenz auf den eigenen Container
12     c = getContentPane();
13     // Setze das Layout
14     c.setLayout(new FlowLayout());
15     // Erzeuge das Labelobjekt mit Uebergabe des Labeltextes
16     beschriftung = new JLabel("Label-Text im Frame");
17     // Fuege das Label dem Frame hinzu
18     c.add(beschriftung);
19   }
20
21   public static void main(String[] args) {
22     FrameMitText fenster = new FrameMitText();
23     fenster.setTitle("Frame mit Text im Label");
24     fenster.setSize(300,150);
25     fenster.setVisible(true);
26     fenster.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27   }
28 }
```

Die Klasse `FrameMitText` erbt von `JFrame`. In ihrem Konstruktor wird zunächst die Referenz auf die Content-Pane (den eigentlichen Container des Frames) bestimmt. Danach wird das Layout des Containers festgelegt. Schließlich wird ein Objekt der Klasse `JLabel` mit dem darzustellenden Text erzeugt und dem Container hinzugefügt. In der `main`-Methode wird eine Instanz der Klasse `FrameMitText` erzeugt, die Beschriftung der Titelleiste und die Größe des Frames festgelegt, der Frame sichtbar geschaltet und schließlich dafür gesorgt, dass das Fenster ordnungsgemäß beendet werden kann.
Nach dem Compilieren konnten wir unser Programm mit dem Java-Interpreter starten, indem wir das Kommando

Konsole

```
java FrameMitText
```

im Konsolenfenster eingaben, wodurch mit der Ausführung der `main`-Methode begonnen wurde. Optisch präsentierte sich unser Programm wie in Abbildung 5.4 dargestellt.
Wollen wir dieselbe Funktionalität nicht in einem Frame, sondern in einem Applet realisieren, müssen wir etwas anders vorgehen:

```
1  import java.awt.*;
2  import javax.swing.*;
3
```

```
4  /** Erzeuge ein einfaches Swing-Applet mit einem Textlabel */
5  public class AppletMitText extends JApplet {
6    Container c;          // Container dieses Applets
7    JLabel beschriftung;  // Label das im Applet erscheinen soll
8
9    // init-Methode fuer unser Applet mit Textlabel
10   public void init() {
11     // Bestimme die Referenz auf den eigenen Container
12     c = getContentPane();
13     // Setze das Layout
14     c.setLayout(new FlowLayout());
15     // Erzeuge das Labelobjekt mit Uebergabe des Labeltextes
16     beschriftung = new JLabel("Label-Text im Applet");
17     // Fuege das Label dem Applet hinzu
18     c.add(beschriftung);
19   }
20
21 }
```

Unsere Klasse `AppletMitText` erbt nun von `JApplet` und wird dadurch mit den vorgefertigten Eigenschaften eines Applets ausgestattet. Wie bei unserer Frame-Klasse verwenden wir auch im Applet die Instanzvariablen `c` vom Typ `Container` und `beschriftung` vom Typ `JLabel`. Da unser Java-Programm selbst nicht für die Erzeugung eines Objekts unserer Applet-Klasse zuständig sein wird, müssen wir keinen Konstruktor programmieren. Der Aufbau unserer einfachen grafischen Oberfläche, der beim Frame noch im Konstruktor untergebracht war, wird in unserer Applet-Klasse in die Methode `init`, eine Methode, die später automatisch beim Start des Applets aufgerufen wird, verlagert. Wiederum wird (wie beim Frame) die Referenz auf die Content-Pane bestimmt, das Layout des Containers festgelegt und ein Objekt der Klasse `JLabel` mit dem darzustellenden Text erzeugt und dem Container hinzugefügt.
Markantester Unterschied zu unserer Frame-Klasse ist sicherlich das Fehlen der `main`-Methode. Unsere Klasse `AppletMitText` kann daher nach dem Compilieren auch nicht direkt mit dem Java-Interpreter gestartet werden. Wir müssen noch ein wenig mehr tun.

10.1.2 Applet in HTML-Datei einbetten

Um unser Applet tatsächlich ausführbar zu machen, müssen wir die Klasse zunächst noch in eine HTML-Seite wie z. B.

```
1  <html>
2    <head>
3      <title>
4        AppletMitText
5      </title>
6    </head>
7    <body>
8      <b>
9        Das Applet AppletMitText
10     </b>
```

```
11      <hr></hr>
12      <applet  code="AppletMitText.class"  width="300"  height="150">
13        Hier sollte eigentlich ein Applet laufen
14      </applet>
15      <hr></hr>
16      Mit normalem Text geht es weiter ...
17    </body>
18  </html>
```

einbetten, die wir dann in der Datei `AppletMitText.html` speichern. Ganz allgemein gesehen, dient die Seitenbeschreibungssprache HTML der Definition von Inhalt, Struktur und Format von Texten und Bildern, die auf Webseiten im Internet dargestellt werden sollen. Dazu werden so genannte **Tags** (deutsch: Markierungen) in Form von Schlüsselwörtern, die jeweils von `<` und `>` eingeklammert werden, verwendet. Diese Tags markieren den Beginn und das Ende (die Tags enthalten dann ein vorangestelltes `/`-Zeichen) eines Text-Abschnitts, der auf eine bestimmte Art und Weise dargestellt werden soll. Beispielsweise leitet `<b>` einen fettgedruckt erscheinenden Text-Abschnitt ein und `</b>` beendet diesen.
Außerdem besteht eine HTML-Datei, die vollständigerweise jeweils mit dem Tag `<html>` beginnt und mit `</html>` endet, in der Regel aus einem Kopf (zwischen `<head>` und `</head>`), der zum Beispiel einen Titel (zwischen `<title>` und `</title>`) enthält, und einem Rumpf (zwischen `<body>` und `</body>`), der die eigentlichen Text- und Bild-Inhalte des Dokuments beinhaltet (in unserem Beispiel insbesondere das Applet). Darüber hinaus stellt die Sprache HTML, die übrigens nicht wie Java zwischen Groß- und Kleinschreibung unterscheidet, natürlich ein umfangreiches Sortiment an Tags zur Verfügung, auf die wir im Rahmen dieses Buchs nicht genauer eingehen können. Hierzu sei auf die zahlreiche Literatur (wie z. B. [7]) und das breite Web-Angebot (wie z. B. [17]) verwiesen. In obigem Beispiel haben wir übrigens noch den Tag `<hr>` eingesetzt, der eine horizontale Linie zeichnet.
Für uns als Applet-Programmierer entscheidend ist ein ganz spezieller HTML-Tag, der Applet-Tag `<applet ...>` in Zeile 12. Dieser ist zuständig für den Aufruf unseres Applets und erfordert zusätzliche Angaben, so genannte **Attribute**, nämlich

- `code="AppletMitText.class"`
  legt den Namen der auszuführenden Java-Applet-Klasse fest.
- `width="300"`
  legt die Breite der Applet-Fläche in Pixel fest.
- `height="150"`
  legt die Höhe der Applet-Fläche in Pixel fest.

Sollte aufgrund eines Fehlers das Applet in einem Web-Browser nicht ausgeführt werden können, so wird lediglich der Text, der zwischen `<applet ...>` und `</applet>` steht (Zeile 13), angezeigt. Ein Teil dessen, was wir im Frame-Beispiel in der `main`-Methode programmiert haben, steckt also nun bei unserem Applet innerhalb des Applet-Tags der HTML-Datei.

Abbildung 10.1: Das Applet im Appletviewer

10.1.3 Applet über HTML-Datei ausführen

Nun sind wir in der Lage, unser Applet auszuführen. Wir benutzen dazu zunächst den Appletviewer, ein Hilfsprogramm, das in jeder JDK-Installation enthalten ist.[1] Geben wir in einem Konsolenfenster das Kommando

Konsole
```
appletviewer AppletMitText.html
```

ein, so wird ein Fenster erzeugt, in dem das Applet dargestellt werden kann. In diesem Fenster wird nun unser Applet angezeigt (siehe Abbildung 10.1) – und zwar nur das Applet, die restlichen Inhalte der HTML-Datei werden ignoriert.
Der vollständige Inhalt unserer HTML-Datei kann nur von einem Internet-Browser wie z. B. *Netscape Navigator* oder *Microsoft Internet Explorer* angezeigt werden. Dazu müssen wir den Browser unserer Wahl dazu veranlassen, die HTML-Datei zu öffnen. Dies kann z. B. über dessen Datei-Menü und den Eintrag „Seite Öffnen“ oder durch Doppelklick auf die HTML-Datei erfolgen. Auf der angezeigten Seite (siehe 10.2) wird nun genau zwischen den beiden horizontalen Linien, die wir mit den beiden `<hr>`-Tags in unsere HTML-Seite eingefügt haben, der im Applet-Tag festgelegte Bereich für das Applet reserviert und dieses dort angezeigt.
Sollte das Applet in Ihrem Browser nicht angezeigt werden, so kann dies daran liegen, dass Ihr Browser nicht mit der aktuellen Version des Java-Laufzeitsystems arbeitet. Dies können Sie leicht beheben, indem Sie das entsprechende **Java-Plug-in** aktivieren. Unter Windows genügt es zum Beispiel, das gleichnamige Hilfsprogramm, das sich unter *Start/Einstellungen/Systemsteuerung* findet, zu starten und darin die mit *Browser* beschriftete Karteikarte zu wählen und darin das Java-Plug-

[1] Jedes neu geschriebene Applet sollte zum Test auf jeden Fall einmal mit dem Appletviewer gestartet werden.

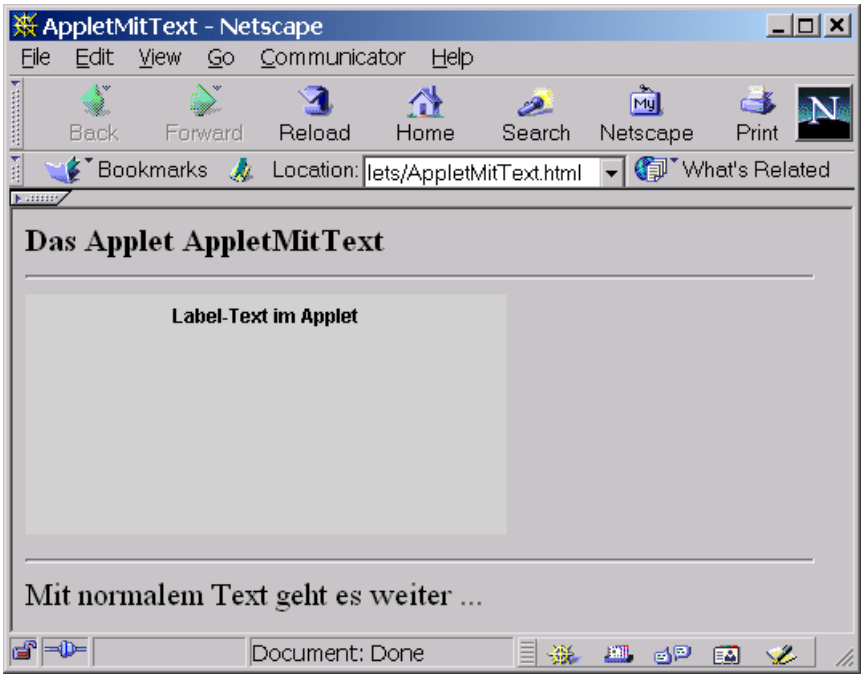

Abbildung 10.2: Das Applet im Browser

in als Standard-Java-Laufzeitprogramm für Ihren Browser einzustellen.[2] Wenn diese Einstellungen übernommen wurden, funktioniert nach einem Neustart des Browsers alles wie gewünscht.

10.2 Die Methoden der Klasse `JApplet`

Ruft man sich die Hierarchie der AWT- und Swing-Klassen (siehe Abbildung 5.6 auf Seite 140) in Erinnerung, so erkennt man, dass die Klasse `JApplet` genau wie `JFrame` und andere Swing-Komponenten von den AWT-Klassen `Component` und `Container` erbt. Somit stehen natürlich – wie allen anderen Komponenten – auch jedem `JApplet`-Objekt die in Abschnitt 6.1 beschriebenen Methoden `getBackground`, `setBackground`, `getForeground`, `setForeground` `getFont`, `setFont`, `getHeight`, `getWidth`, und `setSize` sowie die in Abschnitt 6.2 beschriebenen Methoden `add`, `remove`, `getComponents` und `setLayout` mit entsprechender Funktionalität zur Verfügung. Darüber hinaus

[2] Dies funktioniert natürlich nur, wenn das Java-Plug-in auf Ihrem Rechner installiert ist. Dies ist in der Regel der Fall, wenn Sie das JDK installiert haben. Für Rechner, die über keine komplette JDK-Installation verfügen (sollen), kann das Plug-in von der entsprechenden Webseite [22] heruntergeladen werden.

sind natürlich auch die ebenfalls geerbten Methoden `addXxxListener` und `removeXxxListener` zur Registrierung von Listener-Objekten (vergleiche Abschnitt 7.4) verfügbar.
Genau wie ein `JFrame`-Objekt kann auch ein `JApplet`-Objekt neben der Content-Pane, auf der Komponenten eingefügt werden können, eine eigene Menüleiste erhalten. Daher stellt auch die Klasse `JApplet` die Instanzmethoden `getContentPane` und `setJMenuBar` bereit.
Wichtige Methoden erbt die Klasse `JApplet` von der Klasse `Applet`, wobei den Methoden `init`, `start`, `stop` und `destroy` besondere Bedeutung zukommt. Diese werden nämlich automatisch vom Appletviewer bzw. vom Browser zur Kommunikation mit dem Applet aufgerufen, je nachdem in welchem Zustand („Lebensabschnitt“) sich das Applet gerade befindet.

- **`public void`** `init()`
  wird aufgerufen, wenn das Applet erstmals geladen wird und dient der Initialisierung des Applet-Objekts (sozusagen als Ersatz für entsprechende Anweisungen in einem Konstruktor).
- **`public void`** `start()`
  wird aufgerufen, wenn das Applet gestartet wird (unmittelbar nach `init` und jedes Mal wenn die HTML-Seite, in die das Applet eingebettet ist, erneut angezeigt wird).
- **`public void`** `stop()`
  wird aufgerufen, wenn das Applet vorübergehend angehalten wird (jedes Mal wenn die HTML-Seite, in die das Applet eingebettet ist, nicht mehr angezeigt wird und unmittelbar vor `destroy`).
- **`public void`** `destroy()`
  wird aufgerufen, wenn das Applet zerstört wird (weil der Appletviewer oder der Browser beendet wird).

Diese vier Methoden sind gleichermaßen mit einem leeren Rumpf implementiert, so dass sie zwar prinzipiell aufgerufen werden, aber natürlich nicht aktiv werden. Wenn wir nun aber eine Subklasse von `JApplet` schreiben, so können und sollten wir diese Methoden überschreiben, um in den Methoden entsprechende Aufgaben erledigen zu lassen:

- In der Methode `init` sollten alle für das Appletobjekt wichtigen Initialisierungen durchgeführt werden. Zum Beispiel sollten hier alle benötigten Objekte (wie z. B. Swing-Komponenten oder Listener-Objekte) erzeugt werden.
- In der Methode `start` sollten alle notwendigen Aktionen bzw. Prozesse (z. B. Animationen) des Applets gestartet werden.
- In der Methode `stop` sollten alle Aktionen, die bei nichtangezeigtem Applet nicht benötigt werden, vorübergehend (bis zum nächsten `start`-Aufruf) angehalten werden.

- In der Methode `destroy` sollten alle Aktionen bzw. Prozesse des Applets endgültig beendet und alle Abschlussarbeiten, die vor der endgültigen Zerstörung des Applets notwendig sind, ausgeführt werden.

Wir wollen uns das Zusammenspiel von Browser- bzw. Appletviewer und Applet anhand eines Beispiels verdeutlichen. In unserem Applet

```
1  import javax.swing.*;
2  import java.awt.*;
3  public class AppletMethoden extends JApplet {
4    public void init() {
5      System.out.println("init");
6    }
7    public void start() {
8      System.out.println("start");
9    }
10   public void stop() {
11     System.out.println("stop");
12   }
13   public void destroy() {
14     System.out.println("destroy");
15   }
16 }
```

haben wir den Rumpf der vier Methoden `init`, `start`, `stop` und `destroy` so gestaltet, dass dort jeweils der Methodenname auf das Konsolenfenster ausgegeben wird. Beim Ausführen des Applets können wir somit genau verfolgen, welche der vier Methoden gerade ausgeführt wird. Wenn wir also (natürlich nach Einbettung unseres Applets in eine entsprechende HTML-Seite) beispielsweise nacheinander

- das Applet im Appletviewer starten,
- das Appletviewer-Fenster minimieren,
- das Appletviewer-Fenster wiederherstellen,
- das Appletviewer-Fenster minimieren,
- das Appletviewer-Fenster wiederherstellen und
- das Appletviewer-Fenster schließen (beenden),

so erhalten wir die Konsolenausgaben

Konsole

```
init
start
stop
start
stop
start
stop
destroy
```

10.3 Zwei Beispiele

Zu Beginn dieses Kapitels haben wir bereits angedeutet, dass sich der Aufbau einer grafischen Oberfläche im Applet nicht von dem in einem Frame unterscheidet. Gleiches gilt auch für die Ereignisverarbeitung. Nachfolgend wollen wir dies an zwei Beispielen, die wir bereits mit Hilfe von Frames programmiert haben, demonstrieren, indem wir diese „Web-fähig“ machen, also in ein entsprechendes Applet verwandeln.

10.3.1 Auf die Plätze, fertig, los!

In Abschnitt 7.6 haben wir eine grafische Oberfläche für unser Stoppuhr-Programm aus Kapitel 3 erzeugt. Auch wenn wir diese in einem Applet generieren wollen, gilt es,

- vier `JLabel`-Objekte für die Anzeige der Start-, der Stopp- und der Laufzeit sowie der Status-Information,
- drei `JButton`-Objekte für den Start-, den Stopp- und den Neu-Knopf sowie
- ein `JComboBox`-Objekt für die Auswahl des Anzeigeformats für die gemessene Laufzeit

einzubauen. Die entsprechenden Anweisungen wandern in die `init`-Methode unserer Applet-Klasse.
Um die gewohnte Funktionalität sicherzustellen, benötigen wir natürlich auch jetzt wieder Listener-Objekte für die drei Buttons und für die Combo-Box, so dass wir die Listener-Klassen `KnopfListener` und `BoxListener` direkt aus dem Frame-Programm übernehmen können. Allerdings müssen wir sie nun erneut innerhalb der Frame-Klasse programmieren, und so „rächt“ sich nun die Tatsache, dass wir diese Klassen als innere Klassen unserer Frame-Klasse realisiert haben. Wären es eigenständige Klassen, könnten wir sie auch in unserer Applet-Klasse direkt verwenden. Natürlich müssen wir auch die beiden Hilfsmethoden `differenzString` und `anzeigeAktualisieren` erneut programmieren.
Unser Applet hat somit die Gestalt

```
1  import java.util.*;
2  import java.text.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.border.*;
7
8  /** Swing-Applet mit Stoppuhrfunktion */
9  public class StoppuhrApplet extends JApplet {
10   Container c;
11   JButton   startButton, stoppButton, neuButton;
12   JLabel    startZeit, stoppZeit, differenz, status;
13   JComboBox ergebnisFormat;
14   Date      startZeitObj = null, stoppZeitObj = null;
```

```
15    Font      schriftGross = new Font("SansSerif",Font.BOLD,20),
16              schriftKlein = new Font("SansSerif",Font.BOLD,12);
17    SimpleDateFormat
18              form  = new SimpleDateFormat("dd.MM.yy, HH:mm:ss:SS");
19
20    // Initialisierung des Applets
21    public void init() {
22      c = getContentPane();
23      c.setLayout(new GridLayout(4, 2, 5, 10));
24
25      startZeit = new JLabel("--", JLabel.CENTER);
26      startZeit.setFont(schriftKlein);
27      startZeit.setBorder(new TitledBorder("Startzeit"));
28      stoppZeit = new JLabel("--", JLabel.CENTER);
29      stoppZeit.setFont(schriftKlein);
30      stoppZeit.setBorder(new TitledBorder("Stoppzeit"));
31      differenz = new JLabel("--", JLabel.CENTER);
32      differenz.setFont(schriftGross);
33      differenz.setBorder(new TitledBorder("Laufzeit"));
34
35      KnopfListener kL = new KnopfListener();
36
37      startButton = new JButton("START");
38      startButton.setToolTipText("startet die Stoppuhr");
39      startButton.addActionListener(kL);
40      stoppButton = new JButton("STOPP");
41      stoppButton.setToolTipText("stoppt die Stoppuhr");
42      stoppButton.addActionListener(kL);
43      neuButton = new JButton("NEU");
44      neuButton.setToolTipText("loescht alle Felder");
45      neuButton.addActionListener(kL);
46
47      ergebnisFormat = new JComboBox();
48      ergebnisFormat.addItem("Laufzeit in ms");
49      ergebnisFormat.addItem("Laufzeit in min:sec:ms");
50      ergebnisFormat.addItemListener(new BoxListener());
51
52      status = new JLabel("START druecken!", JLabel.CENTER);
53      status.setFont(schriftGross);
54
55      stoppButton.setEnabled(false);
56      neuButton.setEnabled(false);
57
58      c.add(startZeit);
59      c.add(startButton);
60      c.add(stoppZeit);
61      c.add(stoppButton);
62      c.add(differenz);
63      c.add(neuButton);
64      c.add(ergebnisFormat);
65      c.add(status);
66    }
67
68    // Bestimmung der Laufzeit in ms oder in min:sec:ms als String
69    public String differenzString() {
```

```
70      long diffZeit = (stoppZeitObj.getTime() - startZeitObj.getTime());
71      if (ergebnisFormat.getSelectedIndex() == 0)
72        return (diffZeit + " ms");
73      else {
74        long ms = diffZeit % 1000;
75        diffZeit = diffZeit / 1000;
76        long s = diffZeit % 60;
77        diffZeit = diffZeit / 60;
78        long m = diffZeit % 60;
79        return (m + ":" + s + ":" + ms);
80      }
81    }
82
83    // Aktualisierung aller Anzeige-Labels und Buttons
84    public void anzeigeAktualisieren() {
85      if ((startZeitObj != null) && (stoppZeitObj != null)) {
86        startButton.setEnabled(false);
87        stoppButton.setEnabled(false);
88        neuButton.setEnabled(true);
89        startZeit.setText(form.format(startZeitObj));
90        stoppZeit.setText(form.format(stoppZeitObj));
91        differenz.setText(differenzString());
92        status.setText("NEU fuer neuen Stoppvorgang!");
93        status.setFont(schriftKlein);
94      }
95      else if (startZeitObj != null) {
96        startButton.setEnabled(false);
97        stoppButton.setEnabled(true);
98        neuButton.setEnabled(false);
99        startZeit.setText(form.format(startZeitObj));
100       status.setText("Uhr laeuft!");
101       status.setFont(schriftGross);
102     }
103     else {
104       startButton.setEnabled(true);
105       stoppButton.setEnabled(false);
106       neuButton.setEnabled(false);
107       startZeit.setText("--");
108       stoppZeit.setText("--");
109       differenz.setText("--");
110       status.setText("START druecken!");
111       status.setFont(schriftGross);
112     }
113   }
114
115   // Listener fuer die Buttons
116   class KnopfListener implements ActionListener {
117     public void actionPerformed(ActionEvent e) {
118       if (e.getSource() == startButton)
119         startZeitObj = new Date();
120       else if (e.getSource() == stoppButton)
121         stoppZeitObj = new Date();
122       else if (e.getSource() == neuButton) {
123         startZeitObj = null;
124         stoppZeitObj = null;
```

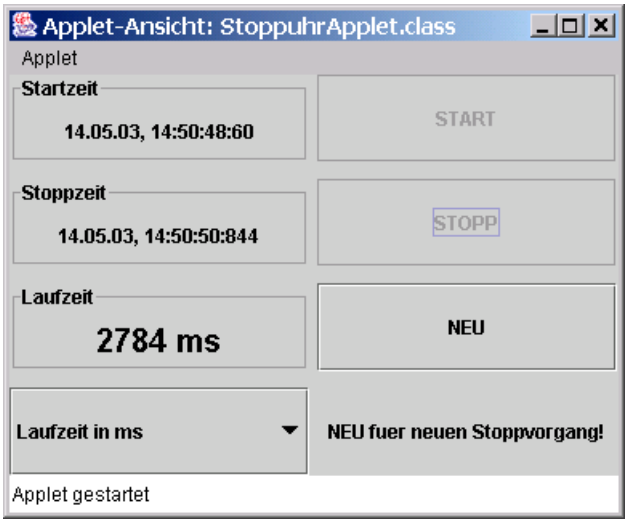

Abbildung 10.3: Die Applet-Variante unserer Stoppuhr

```
125         }
126         anzeigeAktualisieren();
127       }
128     }
129
130     // Listener fuer die Combo-Box
131     class BoxListener implements ItemListener {
132       public void itemStateChanged(ItemEvent e) {
133         anzeigeAktualisieren();
134       }
135     }
136   }
```

und beim Start der HTML-Seite, in die wir das Stoppuhr-Applet einbetten, mit dem Appletviewer bietet sich das gewohnte Bild (siehe Abbildung 10.3).

10.3.2 Punkte verbinden im Applet

Beim Verwandeln unserer Klasse `PunkteVerbinden` aus Abschnitt 9.1.4 in ein Applet haben wir es dank unseres damaligen Designs besonders leicht. Wir lassen natürlich unser Applet

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /** Erzeuge ein Swing-Applet mit einem Zeichenbrett */
6  public class PunkteVerbindenApplet extends JApplet {
7    Container c;           // Container dieses Applets
```

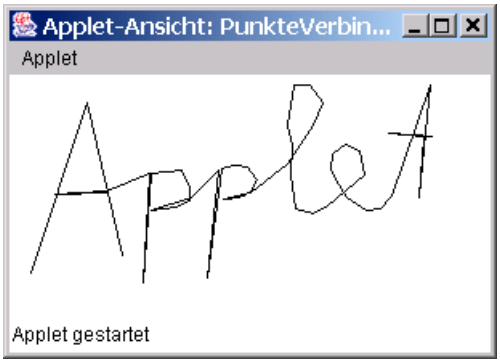

Abbildung 10.4: Maus-Klicks durch Linien verbinden im Applet

```
8     Zeichenbrett z;        // Zeichenbrett zum Linien-Malen
9
10    public void init() {
11      // Bestimme die Referenz auf den eigenen Container
12      c = getContentPane();
13      // Erzeuge neues Zeichenbrett und fuege es dem Frame hinzu
14      z = new Zeichenbrett();
15      c.add(z);
16    }
17  }
```

von `JApplet` erben und übernehmen in der `init`-Methode einfach die drei Anweisungen aus dem Konstruktor von `PunkteVerbinden`. Dabei greifen wir auf die Klasse `Zeichenbrett` (siehe Abschnitt 9.1.4) zurück, die wir hier unverändert zum Einsatz bringen können.
Bei Aufruf unseres Applets bzw. der zugehörigen HTML-Seite mit dem Appletviewer können wir wieder, wie von der Frame-Variante gewohnt, einfache Linienzeichnungen erstellen (siehe Abbildung 10.4).

10.4 Details zur HTML-Einbettung

In diesem Abschnitt wollen wir uns noch kurz mit einigen Details zur Einbettung eines Applets in HTML-Seiten beschäftigen. Dabei werden wir uns einerseits den Applet-Tag nochmals etwas genauer ansehen und andererseits noch eine Methode kennen lernen, mit der das Applet die vom Browser dargestellte Seite festlegen kann.

10.4.1 Der Applet-Tag

Neben `width` und `height` haben wir im Applet-Tag bisher lediglich das Attribut `code` verwendet, mit dem die auszuführende Applet-Klasse angegeben wer-

den kann. Damit das Laden des Applets auch funktioniert, muss sie im gleichen Verzeichnis wie die HTML-Datei liegen. Will man das Applet in einem anderen Verzeichnis ablegen, so kann das Attribut `codebase` dazu verwendet werden, den Pfad zum Applet anzugeben. Eine solche Pfadangabe kann dabei absolut in Form einer vollständigen Web-Adresse oder relativ zum Verzeichnis, in dem die HTML-Seite liegt, erfolgen.
Arbeitet ein Applet mit vielen verschiedenen Klassen, was bei einem größeren Programmierprojekt durchaus der Fall sein kann, so müssen prinzipiell natürlich alle diese Klassen bei Bedarf über das Internet geladen werden. In solchen Fällen bietet es sich an, alle diese Klassen zunächst in einer Archiv-Datei zusammenzufassen. Jede JDK-Installation bietet dafür ein Tool namens `jar`, mit dessen Hilfe solche Archivdateien erzeugt werden können [25]. Bei der Einbettung solcher Applets in die HTML-Datei kann durch das Attribut `archive` diese Archiv-Datei spezifiziert werden. Dadurch lässt sich bei Darstellung der HTML-Seite die komprimierte Archiv-Datei über das Internet übertragen, was natürlich weniger aufwändig ist als die Übertragung der einzelnen Klassen in unkomprimierter Form. Gestartet wird dann wie gewohnt das durch das `code`-Attribut gekennzeichnete Applet, dessen Bytecode der Interpreter direkt aus der Archiv-Datei auslesen kann.
Haben wir beispielsweise die drei compilierten Class-Dateien (die Applet-Klasse `PunkteVerbindenApplet.class`, die Panel-Klasse `Zeichenbrett.class` und die Listener-Klasse `Zeichenbrett$ClickBearbeiter.class`), die in unserem Applet `PunkteVerbindenApplet` benötigt werden in ein Archiv namens `pva.jar` gepackt, und legen wir die Archiv-Datei nicht in das Verzeichnis, in dem die HTML-Datei liegt, sondern in das Unterverzeichnis `keller`, so muss die HTML-Datei folgende Gestalt haben:

```
<html>
  <head>
    <title>
      PunkteVerbindenApplet
    </title>
  </head>
  <body>
    <b>
      Hier kommt das Applet PunkteVerbindenApplet aus dem
      jar-Archiv 'pva.jar' im Unterverzeichnis 'keller'
    </b>
    <hr></hr>
    <applet  code="PunkteVerbindenApplet.class"
             archive="pva.jar"
             codebase="keller"
             width="300"  height="150">
      Hier sollte eigentlich ein Applet laufen
    </applet>
  </body>
</html>
```

Zwischen den beiden Tags `<applet ...>` und `</applet>` können wir nicht nur Text schreiben, den der Browser im Fehlerfall anstelle des Applets anzeigt.

Wir können hier auch beliebige Parameter – allerdings stets als Strings dargestellt – an das Applet übergeben. Dazu steht der HTML-Tag `<param ...>` zur Verfügung, der mehrfach auftreten kann und jeweils mit den Attributen `name` und `value` versehen wird, um den Namen und den Wert eines Parameters festzulegen. In der HTML-Datei

```
1  <HTML>
2  <HEAD>
3     <TITLE>ParameterApplet</TITLE>
4  </HEAD>
5  <BODY>
6    <APPLET code="ParameterApplet.class" width=200 height=100>
7       <param name="north"  value="ich">
8       <param name="east"   value="du">
9       <param name="south"  value="er">
10      <param name="west"   value="sie">
11      <param name="center" value="es">
12   </APPLET>
13 </BODY>
14 </HTML>
```

legen wir beispielweise in den Zeilen 7 bis 11 die fünf Parameter `north`, `east`, `south`, `west` und `center` fest, die wir mit den `String`-Werten `ich`, `du`, `er`, `sie` und `es` belegen.
Innerhalb des eingebetteten Applets können die Werte dieser Parameter mit Hilfe der Methode

- **`public`** `String getParameter(String name)`
  liefert den Wert des Parameters mit Namen `name` aus der Parameterliste innerhalb der beiden Applet-Tags, die das Applet in die HTML-Seite einbetten.

ausgelesen werden. Im Applet `ParameterApplet` würde somit ein Aufruf der Form `getParameter("north")` die Zeichenkette `ich` zurückliefern.
Wollen wir beispielsweise in unserer Klasse `ParameterApplet` dafür sorgen, dass die fünf Parameter zur Beschriftung der fünf Regionen eines Border-Layouts benutzt werden, so könnten wir folgendermaßen vorgehen:

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.net.*;
5
6  /** Erzeuge ein Applet, das die Beschriftungen des
7      Border-Layouts als Parameter uebergeben bekommt */
8  public class ParameterApplet extends JApplet {
9    Container c;
10   JLabel lab;
11   String[] param = {"north", "east", "south", "west", "center"};
12   String[]   ort = { BorderLayout.NORTH, BorderLayout.EAST,
13                      BorderLayout.SOUTH, BorderLayout.WEST,
14                      BorderLayout.CENTER };
15   public void init() {
16     c = getContentPane();
```

Abbildung 10.5: Parameterübergabe an ein Applet

```
17      c.setLayout(new BorderLayout());
18      for (int i=0; i<5; i++) {
19        lab = new JLabel(getParameter(param[i]),JLabel.CENTER);
20        c.add(lab,ort[i]);
21      }
22    }
23  }
```

In der Instanzvariablen `param` (ein Feld mit Komponenten vom Typ `String`) haben wir zunächst die Namen der Parameter festgelegt, die aus der HTML-Datei übernommen werden sollen. In einem weiteren Feld namens `ort` haben wir zusätzlich die entsprechenden „Himmelsrichtungen“ des Border-Layouts abgelegt. Wenn wir uns also mit `param[i]` beschäftigen, wissen wir, dass der aktuelle Wert dieses Parameters im Bereich `ort[i]` des Border-Layouts abgelegt werden soll. In der `init`-Methode können wir daher innerhalb einer Schleife jeweils mit `getParameter` den Wert des `i`-ten Parameters bestimmen und diesen mit `add` im Bereich `ort[i]` als Label platzieren. Mit den Parametern der obigen HTML-Datei ergibt sich somit die in Abbildung 10.5 gezeigte Beschriftung.

10.4.2 Die Methode `showDocument`

Ein Applet ist in der Lage, den Browser, in dem es gerade ausgeführt wird, eine andere Webseite laden und anzeigen zu lassen. Dazu benötigt das Applet natürlich Zugriff auf den umgebenden Kontext (den Web-Browser bzw. den Appletviewer, der das Applet ausführt). In der Klasse `JApplet` steht daher die Methode

- **`public`** `AppletContext getAppletContext()`
  liefert eine Referenz auf den Applet-Kontext.

zur Verfügung. Dabei ist `AppletContext` ein Interface, das unter anderem die Methode

- **`public void`** `showDocument(URL url)`
  veranlasst die Browser-Umgebung, die durch die URL `url` spezifizierte

Internet-Seite anzuzeigen. Falls der Applet-Kontext kein Browser ist, wird der Methodenaufruf ignoriert.

bereitstellt. Um diese Methode einsetzen zu können, müssen wir die Adresse der darzustellenden Webseite in Form eines `URL`-Objekts angeben. Unter einer **URL** (**Uniform Resource Locator**) versteht man eine spezielle Darstellung einer Internet-Adresse. Sie legt genau fest, welche Datei von welchem Rechner mit welchem Dienst aufgerufen werden soll. In Kapitel 14 werden wir uns noch genauer mit dieser Art von Internet-Adressen beschäftigen. Im Moment wollen wir unter URL einfach das verstehen, was wir üblicherweise als Web-Adresse verwenden, wenn wir mit unserem Browser eine bestimmte Seite im Internet ansteuern wollen.
Wollen wir eine bestimmte Webseite mit der Methode `showDocument` anzeigen lassen, so können wir uns mit dem Konstruktor

- **`public`** `URL(String spec)` **`throws`** `MalformedURLException`
  erzeugt eine URL aus der angegebenen String-Darstellung `spec`.

der Klasse `URL`, die im Paket `java.net` bereitgestellt wird, ganz einfach ein `URL`-Objekt erzeugen, müssen allerdings dabei beachten, dass der Konstruktor eine spezielle Ausnahme wirft, wenn die angegebene Zeichenkette rein formal nicht als URL zulässig ist.[3]
In Anwendung der eben beschriebenen Methoden haben wir unser Beispiel-Applet

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.net.*;
5
6  /** Erzeuge ein Applet mit einem Button, der in der Lage
7      ist, die vom Browser angezeigte Seite zu wechseln */
8  public class GoogleButtonApplet extends JApplet {
9    Container c;           // Container dieses Frames
10   JButton button;        // Knopf
11
12   public void init() {
13     c = getContentPane();
14     c.setLayout(new FlowLayout());
15     button = new JButton("Zu Google surfen");
16     c.add(button);
17     ButtonListener bL = new ButtonListener();
18     button.addActionListener(bL);
19   }
20
21   // Innere Button-Listener-Klasse
22   class ButtonListener implements ActionListener {
23     public void actionPerformed(ActionEvent e) {
24       // URL festlegen und anzeigen lassen
25       try {
```

[3] Ob die angegebene Adresse tatsächlich existiert, wird dabei jedoch nicht überprüft.

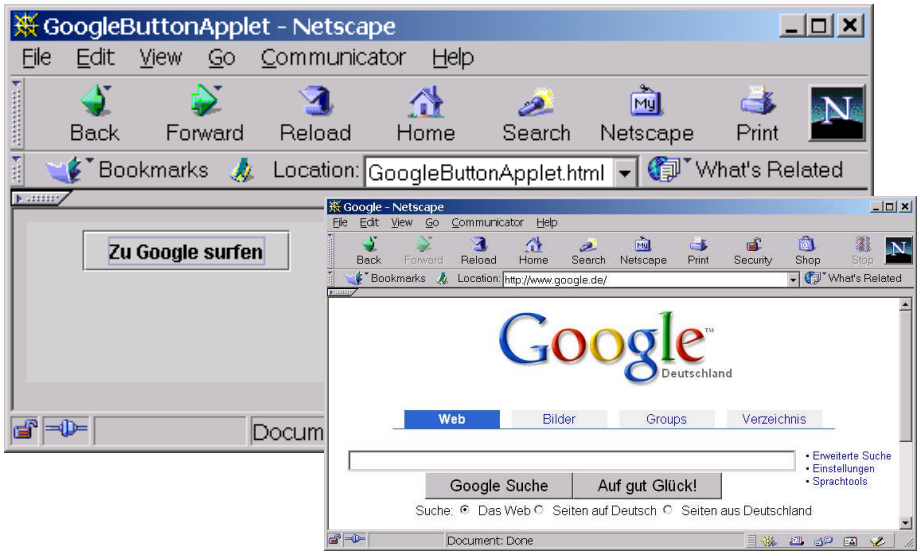

Abbildung 10.6: Browser-Steuerung aus dem Applet

```
26            URL google = new URL("http://www.google.de");
27            getAppletContext().showDocument(google);
28          }
29          catch(MalformedURLException mfue) {
30            System.err.println(mfue);
31          }
32        }
33      }
34    }
```

so gestaltet, dass sich das Applet mit einem Button präsentiert (siehe Abbildung 10.6), durch den wir den ausführenden Browser dazu veranlassen können, zum Beispiel die Suchmaschine „Google“ anzusteuern.

10.5 Sicherheitseinschränkungen bei Applets

Als Nutzer bzw. Nutzerin des Internet mit seinen zahlreichen Diensten wissen Sie sicherlich auch von den Gefahren, die das Herunterladen von Dateien und Programmen im Hinblick auf mögliche Angriffe von außen auf Ihr eigenes Computer-System darstellt. Vor diesem Hintergrund ist natürlich auch die Einbettung eines Applets in eine HTML-Seite kritisch zu betrachten, da beim Surfen auf die entsprechende Seite die Class-Datei des Applets automatisch auf den eigenen Rechner geladen und gestartet wird. Aus genau diesem Grund unterliegen Applets – im Gegensatz zu Applikationen – gewissen Sicherheitseinschränkungen, die den Einsatz gefährlicher Operationen nicht gestatten. Es wird in diesem

Zusammenhang häufig vom so genannten **Sandkasten-Prinzip** gesprochen, weil man Applets in ihrer Ausführung sozusagen in einen Sandkasten verbannt, in dem nur ungefährliche „Spielzeuge“ (Operationen) verfügbar sind. Java-Applets dürfen daher

- nicht lesend oder schreibend auf Dateien des lokalen Rechners zugreifen,
- keine Programme starten,
- keine Bibliotheken laden,
- keine Verbindungen zu einem anderen Rechner (mit Ausnahme des Rechners, von dem das Applet geladen wurde) aufnehmen,
- keine System-Eigenschaften des lokalen Rechners auslesen,
- keine Top-Level-Fenster ohne Warnhinweis erzeugen,
- die virtuelle Maschine (JVM) nicht beenden,

wenn sie im Browser ausgeführt werden. Im Appletviewer sind einige der genannten Aktionen jedoch gestattet.
Mit Hilfe von so genannten **signierten Applets** ist es möglich, diese Beschränkungen für die Applet-Ausführung im Browser ganz oder teilweise aufzuheben. Dazu wird der ausführbare Applet-Code mit einer digitalen Signatur versehen, anhand derer nachgeprüft werden kann, ob das Applet von einer bekannten und zuverlässigen (vertrauenswürdigen) Stelle kommt und daher mehr Rechte bekommen darf. Details zum Thema Sicherheit in Java finden Sie in [23].
Explizit erlaubt ist es Applets jederzeit, Verbindung zu dem Rechner aufzubauen, von dem sie geladen wurden. Somit lassen sich auch eventuell benötigte weitere Klassen oder Dateien zur Laufzeit des Applets nachladen. Allerdings können natürlich keine direkten Zugriffe auf Dateien (zum Beispiel Grafiken, die dargestellt werden sollen) verwendet werden, da diese lediglich im Appletviewer funktionieren würden, während sie im Browser zu einer Sicherheitsverletzung führen.
Übertragen wir unseren Frame `FrameMitBild` aus Abschnitt 6.5.1 in ein Applet

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /** Erzeuge ein einfaches Applet mit einem Bild-Label */
5  public class AppletMitBild extends JApplet {
6    Container c;          // Container dieses Applets
7    JLabel lab;           // Label das im Applet erscheinen soll
8
9    public void init() {
10     c = getContentPane();            // Container bestimmen
11     c.setLayout(new FlowLayout());   // Layout setzen
12
13     // Bildobjekt erzeugen
14     Icon bild = new ImageIcon("babycat.gif");
15     // Label mit Text und Bild beschriften
16     lab = new JLabel("Spotty", bild, JLabel.CENTER);
17     // Text unter das Bild setzen
18     lab.setHorizontalTextPosition(JLabel.CENTER);
```

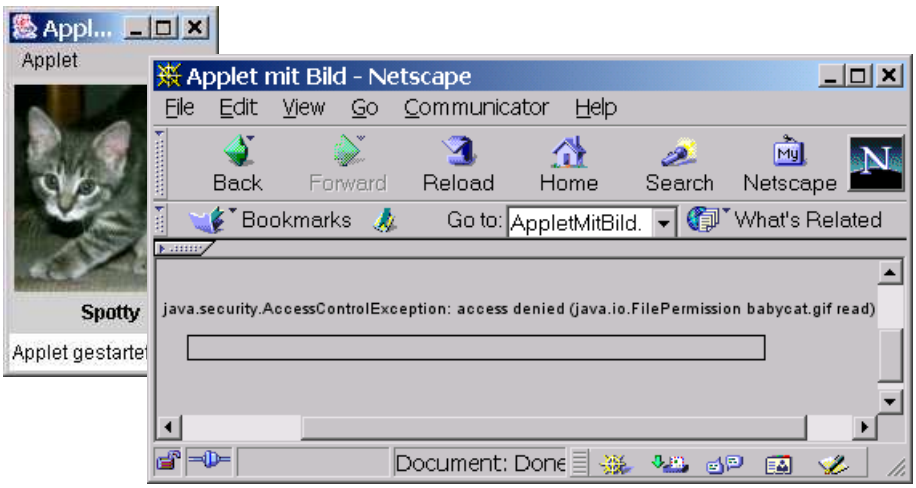

Abbildung 10.7: Applet mit Bild im Appletviewer und im Browser

```
19      lab.setVerticalTextPosition(JLabel.BOTTOM);
20      // Fuege das Label dem Frame hinzu
21      c.add(lab);
22    }
23  }
```

so können wir dieses zwar im Appletviewer problemlos ausführen, erhalten aber in einem Browser eine Fehlermeldung (siehe Abbildung 10.7).
Um dies zu beheben, müssen wir auf die Bild-Datei über eine URL zugreifen, die mit dem Rechner in Verbindung zu bringen ist, von dem das Applet bzw. die HTML-Seite geladen wurde. Um diese URL zur Laufzeit des Applets zu bestimmen, stehen in der Klasse `JApplet` die Methoden

- **`public`** `URL getCodeBase()`
  liefert die URL des Verzeichnisses, in dem das Applet liegt.

- **`public`** `URL getDocumentBase()`
  liefert die URL des Verzeichnisses, in dem die HTML-Datei, in die das Applet eingebettet ist, liegt.

zur Verfügung. Wenn wir also davon ausgehen, dass die Bild-Datei im gleichen Verzeichnis zu finden ist wie das Applet, so können wir uns die URL der Bild-Datei dadurch konstruieren, dass wir der URL des Applet-Verzeichnisses einfach den Dateinamen mittels einer String-Addition hinzufügen und die so entstehende Zeichenkette wieder in ein `URL`-Objekt verwandeln. In unserem modifizierten Applet

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.net.*;
4
5  /** Erzeuge ein einfaches Applet mit einem Bild-Label */
6  public class AppletMitBildBrowse extends JApplet {
7    Container c;          // Container dieses Applets
8    JLabel lab;           // Label das im Applet erscheinen soll
9
10   public void init() {
11     c = getContentPane();            // Container bestimmen
12     c.setLayout(new FlowLayout());   // Layout setzen
13
14     // Bildobjekt erzeugen
15     URL bildURL = createImageURL("babycat.gif");
16     Icon bild = new ImageIcon(bildURL);
17     // Label mit Text und Bild beschriften
18     lab = new JLabel("Spotty", bild, JLabel.CENTER);
19     // Text unter das Bild setzen
20     lab.setHorizontalTextPosition(JLabel.CENTER);
21     lab.setVerticalTextPosition(JLabel.BOTTOM);
22     // Fuege das Label dem Applet hinzu
23     c.add(lab);
24   }
25
26   public URL createImageURL(String file) {
27     String path = getCodeBase() + file;
28     try {
29       return new URL(path);
30     }
31     catch(MalformedURLException mfue) {
32       System.err.println(path + " hat nicht die Form einer URL");
33       return null;
34     }
35   }
36 }
```

haben wir dafür nun die Methode `createImageURL` ergänzt, in der wir, unter der Annahme, dass das Applet und die Bild-Datei im gleichen Verzeichnis liegen, mit `getCodeBase` die URL des Applet-Verzeichnisses bestimmen und diese (durch impliziten Aufruf der Methode `toString`) als Zeichenkette mit dem Namen der Bild-Datei verknüpfen. Unter Beachtung einer möglichen `MalformedURLException` liefert die Methode schließlich die kombinierte URL als Ergebnis zurück.
Bei der Erzeugung des `ImageIcon`-Objekts für die Label-Beschriftung in der `init`-Methode genügt es nun, die URL der Bild-Datei zu bestimmen und anschließend auf den Konstruktor der Klasse `ImageIcon` zurückzugreifen, der ein `URL`-Objekt übergeben bekommt. In dieser Variante lässt sich unser Applet nun auch problemlos im Browser starten (siehe Abbildung 10.8).

Abbildung 10.8: Applet mit Bild im Browser (URL-Variante)

10.6 Übungsaufgaben

Aufgabe 10.1

Schreiben Sie eine Applet-Variante des Bilderrahmen-Frames aus Abschnitt 7.1.2. Beachten Sie dabei, dass Sie auf die Bilddateien nur über eine URL zugreifen können.

Aufgabe 10.2

Erstellen Sie ein Java-Applet mit der in Abbildung 10.9 dargestellten grafischen Oberfläche. Das Applet soll es ermöglichen, unterhalb der Beschriftung `Argument` eine Zahl einzugeben. Diese soll dann als Argument für die Sinus- oder die Cosinus-Funktion (je nach Markierung der entsprechenden Checkbox) verwendet werden. Durch den Button rechts oben soll die Berechnung ausgelöst und das Ergebnis direkt unter dem Button angezeigt werden.
Verwenden Sie in Ihrer Applet-Klasse die privaten Variablen `argumentLabel`, `argument`, `sinCheckbox`, `cosCheckbox`, `executeButton`, `resultLabel` für die sechs benötigten Swing-Komponenten und vereinbaren Sie eventuell notwendige Hilfsobjekte.
Schreiben Sie die Methode `init` so, dass ein geeignetes Layout festgelegt wird, die benötigten Swing-Komponenten erzeugt werden (beachten Sie bei den `JCheckBox`-Objekten, dass jeweils nur eines davon aktiviert sein darf und zu Be-

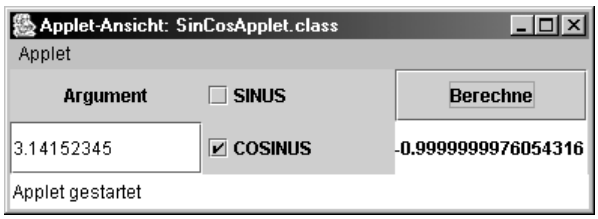

Abbildung 10.9: Das Applet aus Aufgabe 10.2

ginn die Sinus-Funktion ausgewählt sein soll), die Swing-Komponenten in der richtigen Reihenfolge ins Applet eingefügt werden und der Berechnungs-Button mit einem Event-Listener verküpft wird.
Realisieren Sie die Ereignis-Behandlung für den Berechnungs-Button in Form einer inneren Klasse `ExecuteListener`, die das Interface `ActionListener` implementiert. Bei der Betätigung des Berechnungs-Buttons soll jeweils der im Textfeld eingegebene Wert bestimmt werden, die Sinus- oder die Cosinus-Berechnung (abhängig vom Zustand der Checkbox-Markierungen) durchgeführt und das Ergebnis in dem dafür vorgesehenen Label angezeigt werden. Bei einer unzulässigen Eingabe soll eine entsprechende Exception abgefangen und dies über den Text des Resultats-Labels dem Benutzer bzw. der Benutzerin mitgeteilt werden.

Aufgabe 10.3

Im Rahmen Ihres Programmierjobs bei der *Pleiten-Pech-und-Pannen-Bank* haben Sie die Aufgabe bekommen, ein Applet zu schreiben, das als einfaches Euro-Umrechnungsprogramm dient und eine Abbildung 10.10 entsprechende Gestalt und Funktionalität aufweist. Dabei soll die zweite Zeile des Applets zur Eingabe und die vierte Zeile zur Ausgabe dienen. In der dritten Zeile soll mit Hilfe einer Klapptafel die Umrechnungs-Währung (Mark, Schilling oder Gulden) eingestellt werden können. Nach Eingabe eines Euro-Betrages in der zweiten Zeile soll beim Druck auf die Eingabetaste in die eingestellte Währung umgerechnet und der Betrag in der vierten Zeile angezeigt werden. Wählt man in der Klapptafel eine andere Währung aus, so soll ebenfalls sofort in die neu eingestellte Währung umgerechnet und der Betrag in der vierten Zeile angezeigt werden. Durch Aktivieren der Rundungs-Option in der letzten Zeile des Applets kann erreicht werden, dass nach jeder Konvertierung das Ergebnis auf zwei Stellen nach dem Komma gerundet ausgegeben wird.
Statten Sie Ihr Applet mit einem geeigneten Layout und den benötigten Swing-Komponenten aus, und verknüpfen Sie das Texteingabefeld und die Klapptafel mit passenden Listener-Objekten unter Verwendung von anonymen Klassen. Nach Eingabe eines Werts im Texteingabefeld soll sowohl beim Druck auf die Eingabetaste als auch bei einer Änderung des aktuellen Klapptafel-Eintrags die Instanzmethode `wandle` des Applets aufgerufen werden.

Abbildung 10.10: Das Applet aus Aufgabe 10.3

Die Methode `wandle` soll den Wert im Eingabefeld bestimmen (falls dabei eine `NumberFormatException` auftritt, soll der Wert 0.0 verwendet werden), mit Hilfe der Methode `convertTo` aus einer vorgegebenen Klasse `Utils` den Wert in die gerade in der Klapptafel eingestellte Währung umrechnen und den neuen Betrag im Ausgabe-Label anzeigen. Die Darstellung soll dabei unter Beachtung des eingestellten Anzeige-Modus, also bei aktiviertem Häkchen mit Rundung (dazu wird das `DecimalFormat`-Objekt aus der `Utils`-Klasse benutzt) und sonst ohne Rundung erfolgen.
Die vorgegebene Klasse `Utils` ist wie folgt definiert:

```
1  import java.text.*;
2  public class Utils {
3    // Methode zur Konvertierung des Euro-Werts 'euroWert'
4    // in die durch 'waehrung' spezifizierte Waehrung
5    public static double convertTo (String waehrung, double euroWert) {
6      if (waehrung.equals("Mark"))
7        return 1.95583 * euroWert;
8      else if (waehrung.equals("Schilling"))
9        return 13.7603 * euroWert;
10     else // waehrung.equals("Gulden")
11       return 2.20371 * euroWert;
12   }
13
14   // DecimalFormat-Objekt zur Darstellung
15   // von Zahlen mit zwei Nachkommastellen
16   public static DecimalFormat
17     zweiNachKomma = new DecimalFormat("#.00");
18 }
```