4. Applikationen

Applikationen sind vom Webbrowser und Appletviewer losgelöste, eigenständige Anwendungen. Sie sind vergleichbar mit Anwendungen bisheriger Programmiersprachen.

Da sie unabhängig von Browser und Appletviewer arbeiten, unterliegen sie auch nicht deren Sicherheitskriterien, sie können also auf beliebige Daten lesend und auch schreibend zugreifen. Außerdem können sie Applikationen erstellen, die ein ServerSocket enthalten, und sich so einen eigenen Intranet- oder Internetserver programmieren.

Während bei den Applets der Bytecode durch die jeweilige virtuelle Maschine interpretiert wird, muss bei den Applikationen ein auf das Betriebssystem abgestimmter Interpreter diese Aufgabe übernehmen. Es existieren bereits Java-Prozessoren, die über die Fähigkeit verfügen, den Bytecode direkt auszuführen.

Im Gegensatz zu Applets handelt es sich bei Applikationen zunächst um Programme ohne eine grafische Oberfläche.

Um eine Anwendung mit einer grafischen Oberfläche zu erstellen, müssen wir uns hier der Klasse *Frame* bedienen. Diese erlaubt das Generieren von Fenstern sowie die Einbindung visueller Steuermittel, grafischer Elemente und Menüs.

4.1 Grundlegender Aufbau

Applikationen sind im Prinzip nichts anderes als eine Ansammlung mehrerer Klassen. Jede Klasse, die zum Start einer Applikation dienen soll, muss entweder als *public* deklariert sein oder aber ohne jedes Sichtbarkeitsattribut verwendet werden sowie eine *main()*-Methode implementiert haben.

Das Grundgerüst für eine Applikation erstellen

Deklarieren Sie eine Klasse mit einer Bezeichnung Ihrer Wahl.

Jede Klasse kann als Basisklasse einer Applikation benutzt werden. In der Regel werden Applikationen jedoch von der Klasse *java.lang.Object* oder bei GUI-Anwendungen von der Klasse *java.awt.Frame* abgeleitet, dies ist jedoch kein Muss.

Eine Klassen-Deklaration besteht aus drei Teilen:

- dem Schlüsselwort für das Sichtbarkeitsattribut
- dem Schlüsselwort *class*
- dem Namen, den Sie der Klasse geben wollen.

Im Anschluss an den Namen stehen noch zwei geschweifte Klammern, innerhalb derer die Klasse definiert wird.

Fügen Sie zwischen den geschweiften Klammern der Klasse die *main()*-Methode ein. Diese ist immer vom Typ *public static void* und kann ein String-Array als Argumente erhalten. Bei diesen Strings handelt es sich um die Parameter, die optional beim Programmstart eingelesen werden können.

Fügen Sie Variablen, Methoden und weitere Klassen hinzu, die den Programmablauf realisieren. Die grundlegenden Initialisierungen und weiteren Vorbereitungen für den Programmablauf sind in der *main()*-Methode der Hauptklasse oder in ihrem Konstruktor zu definieren.

Beispiel

Grundgerüst einer Klassendefinition als Quelltext

```
public class gewaehlter_Name {

   // Deklaration von Variablen und Objekten

   public static void main(String args[ ]) {

      // Hier können Sie die Initialisierung einfügen

   }
   // Einfügen der Methoden zur Ausführung des Programms

}   // Ende der Klasse
// weitere Klassendefinitionen (optional, dürfen nicht
// als public deklariert sein)
```

Dieses mit Kommentaren versehene Programm wird vom Compiler ohne Fehlermeldung übersetzt und vom Interpreter ausgeführt, auch wenn es keinerlei Funktionalität besitzt. Java ruft beim Start einer Applikation die Methode *main()* auf und übergibt dieser sämtliche Parameter. Eine weitere automatische Steuerung,

ähnlich den Applets, existiert hierbei nicht. Der Programmierer selbst muss also für die korrekte Initialisierung sorgen.

4.2 Erstellen einer Kommandozeilen-Anwendung

Es ist nicht immer unbedingt erforderlich, dass ein Programm über eine grafische Oberfläche verfügt.

So können beispielsweise Programme, die für einen Batchlauf vorgesehen sind, völlig auf eine Oberfläche verzichten, da sie keinerlei Benutzer-Eingaben erwarten. Solche Programme nennt man Kommandozeilen-Anwendungen oder auch Konsolen-Anwendungen.

Eine einfache Konsolen-Anwendung können Sie innerhalb einer einzigen Klasse realisieren. Sie müssen hierzu lediglich das oben gezeigte minimale Programmgerüst erweitern.

Unmittelbar nach der Klassendeklaration werden üblicherweise die Deklarationen der Variablen, Objekte etc. eingefügt, die im ganzen Programm sichtbar sein sollen.

Im Anschluss daran folgt die *main()*-Methode, die der Startpunkt jeder Applikation ist und optional einen oder mehrere Konstruktoren besitzen kann. Zuletzt sind noch die Methoden, oder besser gesagt die Programmlogik, zu ergänzen, die für den Programmablauf zuständig sind.

Ein einfaches Programm

Das Programm hat die Aufgabe, eine *int*-Variable zu definieren und diese auf der Kommandozeilen-Ebene auszugeben. In der *main()*-Methode wird ein Objekt der Klasse *ZeigeInt* erstellt, anschließend wird die Methode *SchreibeInt()* aufgerufen.

```
class ZeigeInt {
   private int intWert = 654;
   public static void main(String args[ ]) {
      ZeigeInt obj = new ZeigeInt ( );
      obj.SchreibeInt( );
   }
   public void SchreibeInt( ) {
      System.out.println(intWert);
   }
}
```

4.3 Erstellen einer GUI-Anwendung

In der Regel ist ein Verzicht auf eine grafische Oberfläche heutzutage nicht mehr möglich. Die Benutzer von heute sind durch aufwendig gestaltete Betriebssysteme ganz andere Anwendungen gewöhnt.

Doch auch hier hat Java es nicht nötig, sich zu verstecken. Gerade durch die grafischen SWING-Komponenten, aber auch die klassischen AWT-Komponenten kann Java durchaus mit klassischen Systemen mithalten oder diese durch seine Plattform-Neutralität sogar noch übertreffen.

Ein Programm mit einer grafischen Oberfläche zu erstellen ist bei Applikationen nicht ganz so einfach wie bei Applets.

Um eine GUI-Anwendung zu entwickeln, bedienen wir uns der von Windows abgeleiteten Klasse *Frame*. Um eine Anwendung in einem Frame darzustellen, können wir entweder die gesamte eigene Klasse einfach von der Klasse *Frame* ableiten (wie in den folgenden Beispielen) oder aber zuerst einen Frame innerhalb einer Klasse erzeugen.

Das reguläre Schließen eines Frames ermöglichen

Damit sind jedoch noch nicht alle Unterschiede einer Applikation zu einem Applet dargestellt.

Im Gegensatz zu der von *Panel* abgeleiteten Klasse *Applet* enthält ein Frame keine Routine, die sein Schließen automatisch ermöglicht.

Da wir eine Windows-Applikationen nicht jedes Mal mit der Tastenkombination Strg+Alt+Entf oder einem Rechnerneustart beenden möchten, müssen wir die Behandlung des entsprechenden Ereignisses selbst einfügen.

Das folgende Programm verwendet die Methode *processWindowEvent(Window Event evt)* aus dem neuen Package *java.awt.event.**.

Beschriftung der Titelleiste eines Frames

Möchten Sie Ihrem Frame einen Namen hinzufügen, gibt es zwei Möglichkeiten.

Zum einen können Sie für Ihre Klasse, wie im folgenden Beispiel, einen Konstruktor schreiben, der diese Aufgabe übernimmt. Das Schlüsselwort *super* darin ist eine Referenz auf die Superklasse, also die "Elternklasse". Dies bedeutet nichts anderes, als dass mit dem Befehl

```
super("Hello Window-World");
```

der Konstruktor der Klasse *Frame* aufgerufen wird, der einen String als Argument hat.

Die zweite Möglichkeit, die Titelleiste eines Frames zu beschriften, ist die Verwendung der Methode *setTitle(String)*. Mit dieser Methode können Sie die Beschriftung der Titelleiste auch nach der Erstellung des Frames verändern, unabhängig davon, auf welche Art die ursprüngliche Beschriftung vorgenommen worden ist.

Beispiel

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

Import der benötigten Packages

```
public class TitelApplication extends Frame {
  static Label l;
```

Definition der Klasse und der nötigen Variable

```
  TitelApplication() {
    super("Hello Window-World");
    enableEvents(WindowEvent.WINDOW_CLOSING);
}
```

Erzeugen einer neuen Instanz und Festlegen der Titelzeile

```
  public static void main(String args[]) {
    TitelApplication frame = new TitelApplication();
    frame.setLayout(new FlowLayout ());
```

Instanz der eigenen Klasse bilden und Festlegung des Layouts. Layout-Manager werden noch an anderer Stelle besprochen.

```
    l = new Label("Hello World");
    frame.add(l);
    frame.setSize(300,200);
    frame.setVisible(true);
  }
```

- Durch die *add()*-Methode wird dem Frame ein neues Element – in diesem Fall ein statischer Text – hinzugefügt.
- Die Methode *setSize()* vergrößert den Frame auf eine Größe von 300 x 200 Punkte.
- Durch *setVisible(true)* wird der Frame letzendlich angezeigt.

```
    protected void processWindowEvent(WindowEvent evt) {
      if(evt.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
      }
    }
```

Dieser Event-Handler ist notwendig, damit die Applikation auch durch das Schließen des Frames beendet werden kann.

```
  }
```

<Hinweis> Positionierung von Elementen

Wenn Sie einem Frame Elemente, wie Buttons oder anderes, hinzufügen, müssen Sie berücksichtigen, dass für diesen Container der BorderLayout-Manager voreingestellt ist.

Das bedeutet in der Praxis, dass Sie beim Hinzufügen von Steuerelementen in der *add( )*-Anweisung auch angeben müssen, wo diese positioniert werden sollen. Anderenfalls werden sie nicht sichtbar sein.

Textausgabe und Zeichnen in einem Frame

In einem Frame können Sie, wie auch in Applets, die *paint()*-Methode anwenden. Die grafischen Kontrollelemente aus dem Package *java.awt.** stehen Ihnen natürlich auch zur Verfügung.

Beispiel

Windows-Applikation mit GUI-Elementen

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

Import der benötigten Packages

```
public class GUIApplication {
  static Fenster fenster;
```

Definition der Applikation und der nötigen Variablen

```
    public static void main(String args[]) {
      GUIApplication ac = new GUIApplication();
        fenster = new Fenster(ac);
    }
```

Erzeugen einer Instanz der Anwendung und eines Fensters

```
    public void schrOben() {
      fenster.c.schrText(0);
    }

    public void schrUnten() {
      fenster.c.schrText(1);
    }

  }  // Ende der Klasse GUIApplication
```

Der Applikation-Teil ist damit abgeschlossen, die Bearbeitung selbst findet innerhalb einer weiteren Klasse statt.

```
// Diese Klasse ist verantwortlich fuer die grafische Ausgabe
class Fenster {
  Can c;

  public Fenster(GUIApplication ac) {
    Frame f = new Frame("Windows-App");
    winEx exhop = new winEx(winEx.CLOSEIT,ac);
    f.addWindowListener(exhop);
    f.setLayout(new FlowLayout());
```

Der neu erzeugten Instanz wird ein EventListener hinzugefügt. Dadurch erst wird es dem Programm möglich, auf Ereignisse zu reagieren.

```
    Ali schr1 = new Ali(Ali.SCHR1, ac);
    Ali schr2 = new Ali(Ali.SCHR2, ac);
```

Definition zweier ActionListener. Diese ermöglichen dem Programm eine Reaktion auf bestimmte Komponenten – in unserem Fall zwei Buttons.

```
    Button b1, b2;
    f.add(b1 = new Button("Obere Zeile"));
    f.add(b2 = new Button("Untere Zeile"));

    b1.addActionListener(schr1);
    b2.addActionListener(schr2);
```

Erzeugen zweier Buttons und Festlegen der Aktion, der diese Buttons auslösen sollen

```
    c = new Can();
    c.setSize(150,100);
    Mausi mausi = new Mausi(ac);
    c.addMouseListener(mausi);
    f.add(c);
```

Erzeugen einer Zeichenfläche und Aktivieren der Maus-Überwachung

```
      f.setSize(200,165);
      f.setVisible(true);
```

Fenster vergrößern und anzeigen

```
    }
  }
  class Can extends Canvas {
    public void paint(Graphics g) {
      g.setColor(Color.blue);
      g.drawRect(0,0, this.setSize().width-1,
        this.setSize().height-1);
    }
```

Neuzeichnen der Zeichenfläche mit blauer Hintergrundfarbe und Rahmen

```
    public void schrText(int i) {
      Graphics g = getGraphics();
      Font f = new Font("TimesRoman", Font.PLAIN, 20);
      g.setFont(f);
      Loesche();
      if(i == 0)
        g.drawString("Obere Textzeile", 10,30);
      if(i == 1)
        g.drawString("Untere Textzeile", 10,70);
    }
```

Durch *getGraphics()* wird der Zeichenbereich für grafische Elemente zugänglich. Wir setzen nun einen neuen Font mittels *setFont()*, löschen den Zeichenbereich und schreiben einen Text in den gewählten Bereich.

```
    public void Loesche() {
      Graphics g = getGraphics();
      Rectangle r = getBounds();
      g.setColor(getBackground());
      g.fillRect(2,2,r.width-3, r.height-3);
      g.setColor(getForeground());
    }

  Löschen der Zeichenfläche}
  class Ali implements ActionListener {
    static final int SCHR1 = 0;
    tatic final int SCHR2 = 1;
    int id;
    GUIApplication ac;
```

Definition der ActionHandler zur Behandlung der Buttons

```
    public Ali(int id, GUIApplication ac) {
      this.id = id;
```

```
      this.ac = ac;
    }

    public void actionPerformed(ActionEvent e) {
      switch(id) {
        case SCHR1:
          ac.schrOben();
          break;
        case SCHR2:
          ac.schrUnten();
          break;
      }
    }
```

Hier wird überprüft, welcher der Buttons benutzt wurde, da diese Methode ja für beide Buttons gleichzeitig genutzt wird. Danach wird in die entsprechende Zeichenmethode verzweigt.

```
  }
  class winEx extends WindowAdapter {
    static final int CLOSEIT = 0;
    int id;
    GUIApplication ac;

    public winEx(int id,GUIApplication ac) {
      this.id = id;
      this.ac = ac;
    }
    public void windowClosing(WindowEvent e) {
      System.exit(0);
    }

    public void windowDeactivated(WindowEvent e) {
      System.exit(0);
    }
  }
```

Diese Methoden sind notwendig, um ein Programmende durch Schließen des Frames zu ermöglichen.

```
  class Mausi extends MouseAdapter {
    GUIApplication ac;

    public Mausi(GUIApplication ac) {
      this.ac = ac;
    }

    public void mouseClicked(MouseEvent e) {
      ac.fenster.c.Loesche();
    }
  }
```

Diese Methoden werden benötigt, um eine Mouse-Aktion zu verarbeiten. In diesem Fall lösen wir bei einem Klick mit der Maus die *Löschen*-Methode aus.

Dieses Beispielprogramm enthält in einem Frame drei Elemente:

- zwei Buttons
- einen Canvas (bzw. ein Objekt einer von *Canvas* abgeleiteten Klasse)

Wird einer der Buttons mit der Maus angeklickt, erscheint jeweils einer der beiden Schriftzüge in der Zeichenfläche. Klicken Sie mit der Maus auf den Canvas, wird der Schriftzug gelöscht.

Diese GUI-Applikation enthält also schon die wichtigsten Grundmerkmale einer kompletten Anwendung:

- die grafische Oberfläche,
- das Hinzufügen von GUI-Elementen,
- die Möglichkeit, auf Benutzeraktionen mit der Maus zu reagieren.

Unser Beispiel verwendet das neue Event-Handling-Modell. Dabei werden den GUI-Elementen Listener-Objekte zugewiesen, die für die Ausführung des Programms von Interesse sind. Die Listener-Objekte fangen die Ereignisse ab und sorgen für eine entsprechende Reaktion des Programms.

Im Beispiel sind daher drei Klassen zur Ereignisbehandlung definiert:

- eine zur Behandlung von ActionEvents,
- eine für die Behandlung von Mausereignissen im Bereich des Canvas,
- eine Klasse, um auf WindowEvents reagieren zu können.

4.4 Parameterübergabe an Applikationen

In Kapitel 4 zum Thema Applikationen sind wir nicht detaillierter auf die Definition der Start-Methode *main* eingegangen, doch diese bietet den Schlüssel zur Parameter-Übergabe an Applikationen.

Jeder Parameter, der hinter der Startklasse angegeben wird, kann vom Programm als Parameter ausgewertet werden und damit natürlich auch zur Ablaufsteuerung dienen. Jeder Parameter wird dabei als Teil eines String-Arrays behandelt. Das String-Array selbst wird der *main*-Methode der Startklasse übergeben.