Kapitel 2

GUI-Programmierung

Das erste Window-System wurde in den 70er Jahren von Xerox PARC entwickelt. Ende der 70er Jahre traten die grafischen Oberflächen mit den Apple Computern Lisa und Macintosh ihren Siegeszug an.

Grafische Benutzungsoberflächen (*Graphical User Interfaces*, abgekürzt *GUI*) haben sich heute zum Standard für anwenderfreundliche Applikationsprogramme entwickelt und rein textuelle Mensch-Maschine-Kommunikation weitgehend abgelöst. Was dem Benutzer im allgemeinen das Leben erleichtern soll, stellt für den Programmierer eine echte Herausforderung dar.

Auf dem zweidimensionalen Bildschirm müssen eine Vielzahl von Fenstern, Icons und Infoboxen mit unterschiedlichster Funktionalität dargestellt werden. Diese müssen durch Eingabegeräte wie Maus oder Tastatur vom Benutzer arrangiert, aktiviert oder mit Eingaben versehen werden können.

Neben der angestrebten intuitiven Bedienbarkeit von grafischen Benutzungsoberflächen erlaubt ein Window-System dem Benutzer auch die Ausgaben mehrerer Programme gleichzeitig zu beobachten und so quasi parallel zu arbeiten.

2.1 Der Window-Manager

Für die geometrische Verwaltung der einzelnen Anwendungen auf einem Bildschirm steht dem Anwender ein Window-Manager zur Verfügung. Er legt die Fenster-Layout-Politik fest.

Der Window-Manager manipuliert Umrandung, Größe und Position der Fenster sowie die Reihenfolge, in der die Anwendungen übereinanderliegen. So kann er verdeckte Fenster in den Vordergrund holen, Fenster über den Schirm bewegen und deren Größe verändern.

Für den Inhalt ihrer Fenster ist die Applikation selbst verantwortlich. Muß ein Fenster nachgezeichnet werden, weil es z.B. durch eine Verschiebe-Aktion nun nicht mehr verdeckt ist, sendet der Window-Manager lediglich eine Redraw-Nachricht an die Applikation.

Außerdem kann der Window-Manager neue Applikationen starten.

Für (fast) jedes Betriebssystem gibt es ein zugehöriges Oberflächensystem, das mit Hilfe eines API (Application Programmer's Interface) manipuliert werden kann. Die Programmierung einer grafischen Applikation erfolgt mit Hilfe einer Graphical User Interface (GUI)-Sprache, die in einer Hochsprache alle notwendigen API-Vokabeln zur Beschreibung des Aufbaus und Ablaufs einer interaktiven grafischen Anwendung bereitstellt. Die Programmierung erfolgt zumeist in einer gängigen Program-

miersprache, die GUI-Bibliotheken verwendet.

Die Beispiele in dieser Vorlesung werden in Java 1.4 unter Zuhilfenahme der Swing-API implementiert.

2.2 Swing

Betrachtet man die verschiedenen Systeme und ihre GUIs, so stellt man fest, daß die Funktionalität bei allen relativ ähnlich ist, die Programmierumgebungen aber völlig verschieden sind. Will man für eine Applikation eine grafische Benutzungsschnittstelle entwickeln, die auf allen verwendeten Systemen eine identische Funktionalität aufweist, so benötigt man eine portable Programmierumgebung. Eine solche Umgebung sind die *Swing-Komponenten* der Programmiersprache Java. Sie sind eine Teilmenge der *Java Foundation Classes* (JFC), basieren auf dem *Abstract Window Toolkit* (AWT) und erweitern dieses um eine Reihe mächtiger GUI-Elemente.

Das AWT umfaßt einige der Java-Standard-Klassen, die für die portable Programmierung von GUI-Applikationen entwickelt wurden. Es heißt *abstract*, weil es die GUI-Elemente nicht selber rendert, sondern sich auf existierende Window-Systeme abstützt.

Im Gegensatz zum AWT kann mit den Swing-Komponenten eine Applikation so implementiert werden, daß alle GUI-Elemente unabhängig von der Plattform, auf der das Programm ausgeführt wird, das gleiche Aussehen und die gleiche Funktionalität haben (*pluggable look and feel* (plaf)). Unter Java2 beinhaltet dies auch den Austausch von Daten zwischen den GUI-Elementen durch den User (*drag and drop*). Dadurch, daß die Swing-Komponenten **keinen** architekturspezifischen Code enthalten, sind sie wesentlich flexibler als die AWT-Klassen. Diese Unabhängigkeit vom Betriebs- bzw. Window-System ist allgemein Teil der Java-Philosophie.

2.2.1 Swing-Übersicht

In diesem Abschnitt wird eine kurze Übersicht der wesentlichen Konzepte der Swing-Komponenten und einiger AWT-Klassen gegeben. Die für die Programmierung notwendigen Details sind bei Java generell der HTML-Online-Dokumentation zu entnehmen.

Das Basis-Paket für Grafik- und Oberflächenprogrammierung ist `javax.swing`. Wie schon erwähnt, werden die dort zur Verfügung gestellten Klassen auf entsprechende Elemente eines schon vorhandenen Window-Systems abgebildet. Zu diesem Zweck gibt es das Paket `java.awt.peer`, das aber normalerweise vom Programmierer nicht direkt benutzt wird.

Weitere verwandte Klassen sind z.Zt. (d.h. in der Java-Version 1.1.x) `java.awt.image` mit Klassen für die direkte Manipulation von Bildern, `java.awt.event` für das neue Eventhandling-Konzept, sowie `java.awt.datatransfer` mit Möglichkeiten zum Datenaustausch zwischen Applikationen. Erwähnt werden sollte hier auch `javax.swing.applet`, dessen einzige Klasse `JApplet` verwendet wird, wenn man Java-Programme in einem HTML-Dokument benutzen will.

Die Klassen des swing-Pakets lassen sich grob in drei Gruppen einteilen: Grafikklassen, in denen grafische Objekte wie Farben, Fonts, Bilder beschrieben werden; Komponenten, d.h. Klassen, die GUI-Komponenten wie z.B. Buttons, Menüs und Textfelder zur Verfügung stellen; sowie Layout-Manager, die die Anordnung von GUI-Komponenten in einer Applikation kontrollieren. Aus den verwandten Paketen sollen hier noch die Event-Listener besprochen werden, mit deren Hilfe die Interaktion zwi-

schen User und Applikationen implementiert werden muß. Die folgende Übersicht erhebt keinen Anspruch auf Vollständigkeit, sondern beschreibt nur kurz die für Computergrafik-Programme wichtigen Klassen. Weitere Klassen sowie alle Details sind der entsprechenden On-Line-Dokumentation zu entnehmen.

Die Grafik-Klassen

Die Grafik-Klassen implementieren jeweils einen bestimmten Aspekt und sind weitgehend unabhängig voneinander. Wichtige Klassen des AWT sind `Color`, `Cursor`, `Font`, `FontMetrics`, `Image`, `MediaTracker`, deren Bedeutung aus dem jeweiligen Namen ersichtlich ist. Wichtige Hilfsklassen sind `Dimension`, `Point`, `Polygon`, `Rectangle`, die jeweils entsprechende Objekte verwalten. Dabei ist zu beachten, daß alle diese Objekte keine Methoden haben, mit denen sie sich z.B. auf dem Schirm darstellen können. Diese Funktionalität wird durch die Klasse `Graphcis` bereitgestellt, in der Methoden vorhanden sind, um Bilder und andere grafische Objekte zu zeichnen, zu füllen, Farben oder Fonts zu ändern, Ausschnitte zu kopieren oder zu clippen usw. Diese abstrakte Klasse wird von diversen Komponenten bereitgestellt und kann dann verwendet werden, um das Aussehen dieser Komponente zu manipulieren.

Die GUI-Komponenten

Die Komponenten, mit deren Hilfe der Benutzer mit einer Oberfläche interagieren kann, lassen sich in drei Gruppen einteilen:

Es gibt Komponenten, die andere Komponenten aufnehmen und anordnen können. Sie werden *Container* genannt und unterteilen sich wiederum in drei Gruppen:

- *Top-Level-Container*, wie `JFrame`, `JDialog` und `JApplet`, die jeweils ein eigenstaendiges "Fenster" erzeugen, das direkt in die grafische Benutztungsoberfläche eingeblendet und vom Window-Manager verwaltet wird.
- *General-Purpose Container*, wie `JPanel` und `JScrollPane`, die in erster Linie beliebige andere Componenten aufnehmen und deshalb nur eingeschränkte eigene Funktionalität bieten.
- *Special-Purpose Container*, wie `JInternalFrame` und `JLayeredPane`, die eine festgelegte Rolle in der Benutzerschnittstelle spielen und deshalb nur eingeschränkt konfgurierbar sind.

Weiterhin gibt es Komponenten, die eine spezielle Funktionalität bereitstellen und als *Basic Controls* bezeichnet werden.

- `JButton`: Es wird ein mit einem Titel oder Bild versehenes Feld erzeugt, das mit der Maus angeklickt werden kann, wodurch ein entsprechender Event erzeugt wird.
- `JCheckbox`: Ein Schalter mit zwei Zuständen, der bei einem Mausklick diesen wechselt und dabei einen Event auslöst.
- `JComboBox`: Eine Gruppe von Einträgen, von der immer nur einer sichtbar ist. Das Auswählen eines Eintrags erzeugt einen entsprechenden Event.
- `JList`: Es wird eine Liste von Strings angezeigt, aus der Einträge selektiert bzw. deselektiert werden können, was zur Erzeugung von entsprechenden Events führt.

- `JScrollBar`: Ein Rollbalken, der mit der Maus bewegt werden kann, wodurch entsprechende Events erzeugt werden.

Außerdem gibt es noch *Displays*, die auf unterschiedliche aber für die einzelne Klasse spezifische Weise Informationen anzeigen. Einige Klassen können editierbar (*editable*) erzeugt werden:

- `JTextArea`: Ein Feld, in dem ein längerer Text angezeigt und evtl. editiert werden kann.
- `JTextField`: Ein Text-Feld, in dem ein einzelner String angezeigt und evtl. editiert werden kann.

Andere sind nicht editierbar (*uneditable*). Hierzu gehören z.B.:

- `JProgressBar`: Ein Fortschrittsanzeiger, der einen prozentualen Wert grafisch darstellen kann.
- `JLabel`: Diese einfachste Komponente erzeugt ein Feld, in dem ein String oder ein Bild ausgegeben wird.

Da es die hier genannten Komponenten zum Teil auch in *java.awt* gibt, wurde den Swing-Komponentennamen zur besseren Unterscheidung jeweils ein "J" vorangestellt.

Da alle Komponenten von der Klasse `javax.swing.JComponent` abgeleitet wurden, die ihrerseits von `java.awt.Container` abstammt, kann jede Instanz einer dieser Klassen wiederum selber andere Komponenten aufnehmen und diese in ihrem Inneren frei anordnen. Dieses *Layout* regelt eine weitere wichtige Gruppe von Klassen:

Die Layout-Manager

Die wesentliche Aufgabe eines Containers ist die Darstellung der darin enthaltenen Komponenten. Zur Steuerung der Anordnung dieser Komponenten werden Klassen verwendet, die das Interface `Layout-Manager` adaptieren. Ein solcher Layout-Manager implementiert dazu eine spezielle Layout-Politik, die mehr oder weniger konfigurierbar ist. Es gibt die folgenden vordefinierten Layout-Manager, die alle aus `java.awt` stammen:

- `FlowLayout`: Dies ist das einfachste Layout, welches bei `JPanel` voreingestellt ist. Jede Komponente wird in seiner bevorzugten Größe rechts von der vorigen Komponente eingefügt. Ist der rechte Rand des Containers erreicht, wird eine weitere Zeile angehängt, in der weitere Komponenten eingefügt werden können.
- `BorderLayout`: Dies ist ebenfalls ein einfaches Layout, das als Default-Layout bei `JFrame` verwendet wird. Es besteht aus den fünf Gebieten, Nord, Süd, Ost, West und Zentrum, in denen jeweils beliebige Komponenten plaziert werden können. Im Gegensatz zum `FlowLayout` werden hier die Komponenten an die Größe des Containers angepaßt: Die Nord- und Süd-Komponenten werden in X-Richtung gestreckt bzw. gestaucht; die Ost- und West-Komponenten in Y-Richtung und die Zentrums-Komponente in beiden Richtungen.
- `CardLayout`: Dies ist ein spezielles Layout, bei dem beliebig viele Komponenten eingefügt werden können, von denen aber immer nur eine zu sehen ist, da sie wie bei einem Kartenspiel übereinander angeordnet werden.

- `GridLayout`: Hier werden die Komponenten in einem zweidimensionalen Gitter angeordnet. Dabei werden alle Komponenten auf die gleiche Größe gebracht, wobei die größte Komponente die Gittergröße vorgibt.

- `GridBagLayout`: Dies ist der flexibelste Manager, mit dem beliebige Anordnungen möglich sind. Es wird wieder von einem zweidimensionalen Gitter ausgegangen, wobei hier jetzt aber eingestellt werden kann, wie viele Gitterzellen eine Komponente belegt und wie sie sich anpaßt.

Alle diese Layout-Manager bestimmen nur die Anordnung der GUI-Elemente, nicht aber deren Look-And-Feel. Diese Aufgabe übernimmt der *User Interface Manager* (`UIManager`). Mit dieser Klasse kann aus den mitgelieferten (z.Zt. Java, Mac, Motif und MS Windows) und evtl. selbstimplementierten Look-And-Feel's gewählt werden.

Event-Handling

Eine offene Frage ist nun noch, wie das Aktivieren einer GUI-Komponente eine entsprechende Aktion im Benutzer-Programm auslöst. Dazu gibt es in AWT folgendes Konzept: Eine Aktion des Benutzers — wie z.B. ein Mausklick — erzeugt in der Java-Maschine einen *Event*. Je nach Art der Aktion und des Kontextes, in dem sie stattfindet, ergeben sich verschiedene Event-Typen, die wiederum verschiedene Argumente haben können. Die Events, die in Zusammenhang mit einer Komponente ausgelöst werden, können im Anwender-Programm empfangen werden, indem ein sogenannter *Event-Listener* mit der Komponente verbunden wird. Für die verschiedenen Event-Typen gibt es verschiedene Listener, von denen einer oder mehrere bei einer Komponente eingetragen werden können. Ein Listener ist dabei ein Interface, in dem alle Methoden vorgegeben werden, die für eine Menge von zusammengehörigen Events notwendig sind. Tabelle 2.1 gibt eine Übersicht der verschiedenen Event-Listener.

Für einige der Swing-Komponenten wurden zusätzliche EventListener implementiert. Sie befinden sich in dem Paket `javax.swing.event`. Sie kommen nur bei spezialisierten Komponenten zum Einsatz und werden daher hier nicht ausführlich beschrieben.

Listener	Methoden	Komponenten
Action	actionPerformed	JButton, JList, JMenuItem, JTextField
Adjustment	adjustmentValueChanged	JScrollBar
Component	componentHidden	JComponent
	componentMoved	
	componentResized	
	componentShown	
Container	componentAdded	Container
	componentRemoved	
Focus	focusGained	JComponent
	focusLost	
Item	itemStateChanged	JCheckbox, JComboBox, JList
Key	keyPressed	JComponent
	keyReleased	
	keyTyped	
Mouse	mouseClicked	JComponent
	mouseEntered	
	mouseExited	
	mousePressed	
	mouseReleased	
MouseMotion	mouseDragged	JComponent
	mouseMoved	
Text	textValueChanged	JTextComponent
Window	windowActivated	JWindow
	windowClosed	
	windowClosing	
	windowDeactivated	
	windowDeiconified	
	windowIconified	
	windowOpened	

Tabelle 2.1: AWT-Event-Listener

2.3 Swing-Beispiel

In diesem Abschnitt wird eine einfache grafische Anwendung vorgestellt, in der eine Zahl per Knopfdruck wahlweise hoch- oder runtergezählt und sowohl numerisch als auch per Slider angezeigt wird. Diese Anwendung wird zum Einen als eigenständige Applikation und zum Anderen als Applet, welches in einem Webbrowser zu betrachten ist, realisiert.

Die Anwendung besteht aus fünf Klassen. Der Kern mit dem grafischen User-Interface und dem Eventhandling ist gemäß der *Model-View-Controller*-Architektur in den Klassen `Zustand` (*Model*), `RaufRunter`(*View*) und `KnopfKontrollierer` (*Controller*) implementiert. Die Klassen

`RaufRunterApp` und `RaufRunterApplet` stellen die Top-Level Container dar, in denen die Anwendung als Application bzw. als Applet ausgeführt wird.

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

public class RaufRunter extends JPanel implements Observer {

  private JButton rauf;                              // Button zum raufzaehlen
  private JButton runter;                            // Button zum runterzaehlen
  private Zustand z;                                 // Zustand
  private JLabel  ergebnis;                          // Label zur Anzeige
  private JSlider schieber;                          // Slider zur Visualisierung
  private Font    font;                              // Font fuer Label

  public RaufRunter() {                              // Konstruktor

    setLayout(new GridLayout(0,1));                  // einspaltiges Gridlayout

    rauf   = new JButton("Addiere");                 // Knopf rauf anlegen
    runter = new JButton("Subtrahiere");             // Knopf runter anlegen
    schieber = new JSlider(0,100,42);                // Slide von 0 bis 100
    ergebnis = new JLabel("42",JLabel.CENTER);       // JLabel ergebnis anlegen
    font = new Font("SansSerif",Font.BOLD,30);       // Schriftgroesse einstellen
    ergebnis.setFont(font);                          // an Label uebergeben

    add(rauf);                                       // JButton rauf einfuegen
    add(ergebnis);                                   // JLabel ergebnis einfuegen
    add(schieber);                                   // Slider einfuegen
    add(runter);                                     // JButton runter einfuegen

    z = new Zustand(42);                             // Zustand initialisieren
    z.addObserver(this);                             // dort als Observer eintragen

    KnopfKontrollierer raufK;                        // fuer rauf-Eventhandling
    raufK = new KnopfKontrollierer(z,+1);            // soll hochzaehlen
    rauf.addActionListener(raufK);                   // Listener an rauf haengen

    KnopfKontrollierer runterK;                      // fuer runter-Eventhandling
    runterK = new KnopfKontrollierer(z,-1);          // soll runterzaehlen
    runter.addActionListener(runterK);               // Listener an runter haengen
  }

  public void update(Observable z, Object dummy){ // wird aufgerufen bei notify
    ergebnis.setText(((Zustand)z).get() + " ");    // Zaehlerstand anzeigen
    schieber.setValue(((Zustand)z).get());         // Zaehlerstand visualisieren
  }
}
```

Die Klasse `RaufRunter` stellt die *View* dar und stammt von `JPanel` ab. Bei der Instanziierung wird zunächst das Layout festgelegt und im weiteren werden die benötigten Komponenten erstellt. Für das

Programm wird ein `JButton` zum Hochzählen, ein `JButton` zum Runterzählen, ein `JLabel` zur Anzeige und ein `JSlider` zur Visualisierung des Zahlenwerts benötigt. Alle Komponenten werden innerhalb des einspaltigen Gridlayouts auf dieselbe Größe gesetzt. Diese ist nicht fest und kann sich ggf. durch die Größe des Top-Level Containers (das Applet oder die Applikation) ändern. Diese vier Komponenten werden dem `JPanel` hinzugefügt, welches in den jeweiligen Top-Level Container eingefügt werden kann.

Weiterhin wird eine Instanz der Klasse `Zustand` initialisiert, welche die Rolle des *Model* übernimmt, sowie zwei Instanzen des `KnopfKontrollierer`, welche als `ActionListener` beim jeweiligen Knopf eingehängt werden.

Schließlich wird mit der Methode `update` das Interface `Observer` implementiert, wodurch auf eine Zustandsänderung reagiert werden kann, die vom `Observable` per `notify` gemeldet wird.

```
import java.util.Observer;
import java.util.Observable;

public class Zustand extends Observable{       // Zustand

  private int zaehler;                         // Zaehler

  public Zustand(int zaehler){                 // Konstruktor
    this.zaehler=zaehler;                      // initialisiere Zaehler
  }

  int get(){return zaehler;}                   // Zaehler abfragen

  void aendern(int delta){                     // Zaehlerstand aendern
    zaehler = zaehler + delta;                 // erhoehen oder erniedrigen
    setChanged();                              // notiere Aenderung
    notifyObservers();                         // Observer benachrichtigen
  }
}
```

Die Klasse `Zustand` stellt das *Model* dar und beinhaltet den aktuellen Zählerstand. Sie ist von der Klasse `Observable` abgeleitet und verfügt daher über Methoden `setChanged` und `notifyObservers`, durch die beim Observer die Methode `update` gestartet wird. Durch den Konstruktor erhält der Zähler seinen initialen Wert und mit der Methode get() kann der aktuelle Zählerstand in Erfahrung gebracht werden.

Die Klasse `KnopfKontrollierer` stellt den *Controller* dar und ist für das Eventhandlung zuständig. Dazu wird das Interface `ActionListener` implementiert, welches aus der einzigen Methode `actionPerformed` besteht. Dieser Listener wird in unterschiedlichen Instanziierungen den beiden `JButton`s übergeben. Bei einem klick auf einen Button wird dieser Event an den Listener übergeben und dort die Methode `actionPerformed` aufgerufen. Da der `JSlider` nur zu Visualisierung des aktuellen Zahlenwerts benötigt wird, muss auf benutzerdefiniertes Zerren am Slider nicht reagiert werden.

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

public class KnopfKontrollierer implements ActionListener {

  private Zustand z;                                  // Verweis auf Zustand
  private int delta;                                  // spezifischer Inkrement

  public KnopfKontrollierer(Zustand z, int delta) {
    this.z       = z;                                 // merke Zustand
    this.delta   = delta;                             // merke Inkrement
  }

  public void actionPerformed(ActionEvent e) {        // bei Knopfdruck
    z.aendern(delta);                                 // Zustand aendern
  }
}
```

Es fehlem noch die beiden Top-Level Container, welche den Rahmen für die Anwendung bilden, in dem die Instanz der Klasse `RaufRunter` ausgeführt wird.

```
import java.awt.BorderLayout;
import javax.swing.JFrame;

public class RaufRunterApp {

  public static void main(String args[]) {
    JFrame rahmen = new JFrame("RaufRunter-Applikation");
    rahmen.getContentPane().add(new RaufRunter(), BorderLayout.CENTER);
    rahmen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    rahmen.pack();
    rahmen.setVisible(true);
  }
}
```

In `RaufRunterApp` wird ein `JFrame` erzeugt und in dessen `ContentPane` wird die `RaufRunter`-Komponente eingefügt. Mit der Methode `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)` wird festgelegt, dass die Applikation beim Schließen des Fensters endet, sonst würde sie weiterlaufen und wäre lediglich unsichtbar.

Mit der Methode `pack()` wird die Größe der einzelnen Componenten bestimmt und dem Top-Level Container mitgeteilt. Zuletzt muss das `JFrame` mit der Methode `setVisible` sichtbar gemacht werden.

Um die Anwendung als Applet im Browser zu betrachten wird die Klasse `RaufRunterApplet` benötigt:

```
import java.awt.BorderLayout;
import javax.swing.JApplet;

public class RaufRunterApplet extends JApplet {

  public void init() {
    getContentPane().add(new RaufRunter(), BorderLayout.CENTER);
  }
}
```

Beim Start als Applet instanziiert der Appletviever (oder Browser) eine Instanz der Klasse `RaufRunterApplet` und ruft dort die `init`-Methode auf. In dieser Methode wird eine neue Instanz der Klasse `RaufRunter` erzeugt und der `ContentPane` des Applets hinzugefügt.

Das Applet wird dann in eine HTML-Seite eingebunden:

```
<HTML>
  <HEAD>
    <TITLE>RaufRunter-Applet</TITLE>
  </HEAD>
  <BODY>
    <CENTER>
    <P>
    <APPLET
       width=183
       height=183
       code=RaufRunterApplet.class
       archive=raufRunter.jar>
    </APPLET>
    </CENTER>
  </BODY>
</HTML>
```

Achtung: Da das Applet Swing-Klassen benutzt, kann es notwendig sein, ein aktuelles Java Plugin zu installieren.

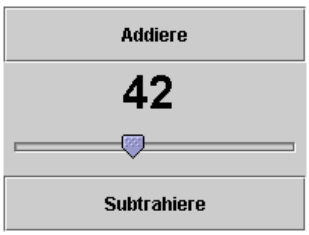

Abbildung 2.1: Screenshot vom RaufRunterApplet