Informatik IIb

Objektorientierte Programmierung mit Java

Oliver Jack

Fachhochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Wintersemester 2010/11

Vorlesung 8. Datenbank, Multimedia

Lernziele

- Kenntnis des Lesens aus SQL-Datenbanken
- Kenntnis von elementaren Grafikdarstellungsmöglichkeiten
- Kenntnis von Zugriff auf Audiodaten

JDBC: Java, DBs und SQL

- Java bietet mit der Java Database Connectivity die Möglichkeit (JDBC), auf SQL-kompatible DB-Server zuzugreifen.
- SQL steht Structured Query Language, ANSI-Standard für eine Datenbank-Anfragesprache, SQL-kompatible Datenbanken speichern ihre Daten relational, d. h. in Tabellenform, z. B.

Vorname	Nachname
Oliver	Jack
Burkart	Voß

- SQL-Anfragen bestehen meist aus:

```
SELECT * FROM Mitarbeiter [WHERE Bedingung]
```

JDBC: Java, DBs und SQL (Forts.)

- JDBC (Package java.sql) ist eine API, die die notwendigen Java-Konstrukte bereit stellt, um:
  - mit einer SQL-DB zu verbinden
  - SQL-Kommandos auszuführen
  - die Ergebnisse zu verarbeiten
- Die JDBC-API schirmt dabei das Java-Programm (und dessen Autor) von den technischen Details der DB-Verbindung ab, daher wird für jeden DB-Typ ein spezifischer Treiber benötigt
- Mit MySQL ist ein weit verbreiteter DB-Server kostenlos im WWW verfügbar
- Java-Treiber gibt es im Netz (http://www.mysql.com/downloads/api-jdbc-stable.html)

JDBC Beispiel

```
// ...
String url = "jdbc:mysql://10.90.1.188/testdb";
Class.forName("com.mysql.jdbc.Driver").newInstance();
conn = DriverManager.getConnection(url,
  "testdb", "testdb");
Statement s = conn.createStatement();
s.executeQuery ("SELECT * FROM Mitarbeiter");
ResultSet rs = s.getResultSet();
while (rs.next()) {
  System.out.println(rs.getString("Vorname") + " " +
    rs.getString("Nachname"));
}
conn.close();
// ...
```

JDBC Beispiel (Forts.)

- Der URL-String zur Verbindung mit dem DB-Server:

```
jdbc:mysql://hostname:port/databasename
```

- Den Treiber erzeugen:

```
Class.forName("com.mysql.jdbc.Driver").
  newInstance();
```

- Der Treiber wird in die JVM geladen und registriert sich dabei beim DriverManager des SQL-Packages.
- Somit sind systemweit verschiedene SQL-Treiber möglich und das Programm braucht keine Referenz auf den Treiber.
- Der DriverManager sucht an Hand der URL den passenden Treiber heraus. Beim DriverManager eine Verbindung zum Server holen:

```
conn = DriverManager.getConnection(url,
  "testdb", "testdb");
```

JDBC Beispiel (Forts.)

- Ein SQL-Statement anlegen und eine Anfrage dafür erzeugen:

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT␣*␣FROM␣Mitarbeiter");
```

- Die Ergebnisse (Tabellenzeilen) der Anfrage auslesen:

```
ResultSet rs = s.getResultSet ();
while (rs.next ()) {...}
```

- Veränderungen an der Datenbank sind möglich z.B. mit:

```
s.executeUpdate("INSERT␣INTO␣" +
"mitarbeiter␣(Nachname,␣Vorname)␣" +
"VALUES␣('Wagner',␣'Herbert')");
```

Multimedia in Java

Java unterstützt plattformunabhängig eine breite Palette an Multimedia-Fähigkeiten

- GUI-bezogen mit Swing (Icons u.v.m)
  - Grafik
  - Java2D
  - Java3D
- JavaSound
- Java Media Framework
  - Audio
  - Video
  - Capturing

Einfache Grafik

Programme sollten normalerweise nicht direkt auf dem Bildschirm zeichnen, sondern immer in einem Fenster

- Unter Swing ist es daher notwendig, die paint-Methode einer JComponent zu überschreiben
- Diese wird dann jedes Mal vom System aufgerufen, wenn die Komponente neu gezeichnet werden muss
- Die paint-Methode erhält vom System automatisch einen Grafik-Kontext übergeben (wie bei einem Applet)

```
public class DrawExample extends JFrame {
  public void paint(Graphics g) {
  g.setColor(Color.GREEN);
  g.drawString("Hello␣World!");
  }
}
```

Graphics

- Jedes Graphics-Objekt enthält folgende Eigenschaften:
  - Die zugehörige Komponente
  - Eine Nullpunkt-Verschiebung
  - Das aktuelle Clipping
  - Die aktuelle Zeichenfarbe
  - Den aktuellen Zeichensatz
  - Pixeloperation (direkt zeichnen oder XOR, ggf. XOR-Farbe)
- Und jede Menge Methoden zum Manipulieren der Eigenschaften sowie z.B. zum Zeichnen
  - von Polygonen, Kreisen usw...
  - von Text
  - von Bildern

Bilder laden

Einfachste Version

- Lädt das Bild vollständig, bevor der Programmfluss weiter geht
- Meldet aber keine Fehler (daher ggf. image != null prüfen)
- Auch Skalierung des Bildes ist direkt möglich:

```
g.drawImage(image, 10, 10, image.getWidth(null) * 2,
image.getHeight(null) * 2, null);
```

- Ein ImageIcon kann auch verwendet werden, um z.B. einem JLabel ein Icon hinzuzufügen: `new JLabel(new ImageIcon("image.gif"));`

```
public class LoadImage extends JFrame {
  ...
  Image image = new ImageIcon("image.gif").getImage();
  ...
  public void paint(Graphics g) {
  g.drawImage(image, 10, 10, null);
  }
}
```

Bilder laden (Forts.)

Java unterstützt standardmäßig die folgenden Bild-Formate

- Graphics Interchange Format (GIF)
- Portable Network Graphics (PNG)
- Joint Photographic Experts Group (JPEG)

Größere Bilder (z.B. aus dem Internet) sollten nicht blockierend, mit einem ImageObserver geladen werden

```
public class LoadImage extends JFrame {
  ...
  Image image =
      Toolkit.getDefaultToolkit().getImage("image.gif");
  ...
  g.drawImage(image, 10, 10, this);
}
```

Ein ImageOberserver ist ein Interface, das von allen Komponenten (also auch JFrame) implementiert ist: Es erfolgen automatische Updates, sobald neue Bildteile geladen sind

Bilder ohne Komponenten

Image-Operationen sind auch vollständig ohne Swing- Komponenten möglich

```
BufferedImage image =
    new BufferedImage(width, height,
                      BufferedImage.TYPE_INT_RGB);
Graphics2D g = image.createGraphics();
```

- `BufferedImage.TYPE_INT_ARGB` speichert zusätzlich einen Alpha-Wert pro Pixel
- `BufferedImage TYPE_BYTE_GRAY` dient zur Erstellung von Graustufen-Bildern
- `Graphics2D` erbt von Graphics und bietet damit alle deren Operationen an
- zusätzlich noch z.B. Rotationen oder Clipping von nicht-rechteckigen Bereichen

Bild-Manipulationen

- Bilder können unter Java2D mit Hilfe eines AffineTransform-Objektes verändert werden
- Eine affine Transformation ist eine lineare Abbildung einer Ursprungskoordinate auf eine Zielkoordinate
  - parallele Linien bleiben dabei parallel
  - erfolgt über eine Transformationsmatrix
- Folgende affine Transformationen sind möglich
  - Translation
  - Rotation
  - Skalierung
  - Scherung

```
AffineTransform tx = new AffineTransform();
tx.rotate(Math.PI);
g.drawImage(image, tx, this);
```

Farbraum konvertieren

- Auch die Konvertierung beispielsweise eines farbigen Bildes in Graustufen braucht nicht per Hand zu erfolgen
- In Java existieren Farbräume und Klassen die zwischen den Farbräumen konvertieren
- Bilder werden also gefiltert

```
ColorSpace cs =
    ColorSpace.getInstance(ColorSpace.CS_GRAY);
ColorConvertOp op =
    new ColorConvertOp(cs, null);
bufferedImage = op.filter(bufferedImage, null);
```

Filtern

In der digitalen Signalverarbeitung wird sehr häufig mit Filtern gearbeitet

- Jedes bessere Bildbearbeitungsprogramm bietet ebenfalls solche Filteroprationen
- Grob gesagt wird hierfür ein Bild als Matrix betrachtet
- Auch ein Filter ist eine (kleinere, meist quadratische) Matrix, die über ein Bild geschoben wird und neue Werte berechnet (Faltung)
- Beispiel Weichzeichnungsfilter

```
Kernel kernel = new Kernel(3, 3, new float[] {
    1f/9f, 1f/9f, 1f/9f,
    1f/9f, 1f/9f, 1f/9f,
    1f/9f, 1f/9f, 1f/9f});
BufferedImageOp op = new ConvolveOp(kernel);
bufferedImage = op.filter(bufferedImage, null);
```

Bilder speichern

- Das Speichern von Bildern erfolgt ähnlich problemlos, wie das Laden
- Java unterstützt standardmäßig das Speichern von JPEG und PNG
- JPEG-Kompression ist über ImageWriter eintstellbar
- GIF aus lizenzrechtlichen Gründen nicht implementiert, aber über Blibliotheken von Drittanbietern realisierbar

```
import javax.imageio.*;
...
BufferedImage image = new BufferedImage
                (w, h, BufferedImage.TYPE_INT_RGB);
ImageIO.write(image, "jpeg", new File("file.jpeg"));
```

Audio

Seit Version 1.3 ist die Java Sound API integriert

- Unterstützung für:
- AIFF, AU, WAV, MIDI, RMF
- 8 oder 16 Bit Mono/Stereo
- 8 bis 48 kHz

Einfaches Laden von Audioclips über eine statische Methode der Klasse Applet

```
import java.applet.Applet;
AudioClip audioClip = Applet.newAudioClip(new URL("file:" +
    System.getProperty("user.dir") + "/" + "bottle.wav"));
audioClip.play();
```

Audio (Forts.)

Es sind aber noch wesentlich komplexere Audio-Bearbeitungen möglich

- Über AudioStreams können die Daten in den Speicher gelesen, dort bearbeitet und wieder geschrieben werden
- Beispiel Lesen:

```
import javax.sound.sampled.*;
AudioInputStream stream = AudioSystem.getAudioInputStream(
            new File("bottle.wav"));
```

- Auch die Aufnahme vom Mikrofon eines Rechners ist möglich
- Da die dazu notwendigen Code-Beispiele sind relativ komplex, z.B. SoundPlayer.java

Java Media Framework (JMF)

- Möglichkeiten
  - Abspielen von zahlreichen Multimedia-Formaten (z.B. auch MPEG)
  - Streaming über das Netz ist möglich
  - Audio/Video-Capturing und natürlich Speichern ist möglich
  - Multimedia-Boradcast über das Internet wird unterstützt
- Bei der Entwicklung des JMF wurde auf möglichst wenig Interaktion mit dem Programmierer Wert gelegt
- Das JMF kann Daten von DataSource lesen und in DataSink schreiben (über Streams): das können jeweils Dateien, Hardware oder das Netzwerk sein
- Daten werden über Player abgespielt, die soweit möglich selbst das passende Ausgabegerät wählen

JMF-Player

Player werden durch eine Manager-Factory erzeugt

- so können Player leicht erweitert werden
- Player „kommunizieren“ über Events
- Sie bieten Methoden wie start(), stop() usw.
- Beispiel: mit dem JMF ein MP3-File (nur unter Windows) zu laden und abzuspielen:

```
import javax.media.*;
import java.io.*;
public class MyMP3 {
  public static void main(String args[]) throws Exception {
    Player player = Manager.createPlayer(
       new File("file.mp3").toURL() );
    player.start();
} }
```

JMF-Player (Forts.)

- Um auf Player-Events zu reagieren, muss das Interface ControllerListener implementiert werden
- Dazu genügt der folgende Code

```
public synchronized
void controllerUpdate(ControllerEvent evt) {
  if (evt instanceof EndOfMediaEvent) {
    System.out.println("Play finished");
    System.exit(0);
  }
}
```

JMF-Player (Forts.)

Wird die controllerUpdate-Methode wie folgt erweitert, ist es möglich, vorgefertigte JMF-Controls zur Steuerung des Players zu nutzen

```
if(controllerevent instanceof
    RealizeCompleteEvent) {
  java.awt.Component component =
      player.getControlPanelComponent();
  getContentPane().add(component, "Center");
  pack();
  setVisible(true);
}
```

JMF-Player (Forts.)

- Und auch die Video-Wiedergabe ist mit dem bekannten Code-Gerüst möglich
- Dazu muss wiederum die controllerUpdate-Methode geringfügig erweitert werden

```
public synchronized
void controllerUpdate(ControllerEvent evt) {
  if (evt instanceof RealizeCompleteEvent) {
    Component myVisual = player.getVisualComponent();
    if (myVisual != null)
      getContentPane().add(myVisual, BorderLayout.CENTER);

    Component controlPanel =
        player.getControlPanelComponent();
    getContentPane().add(controlPanel, BorderLayout.SOUTH);
    pack();
    setVisible(true);
  }
}
```

JRadioButton und ButtonGroup

Wichtige Methoden

- `JRadioButton(String, boolean)` Erzeugt einen `RadioButton` mit Text
- `JRadioButton(String, Icon, boolean)` Erzeugt einen `RadioButton` mit Text und Icon
- `ButtonGroup()` Erzeugt `ButtonGroup`-Instanz
- `add(AbstractButton)` Zufügen eines (Radio)Button zur `ButtonGroup`
- `remove(AbstractButton)` Entfernen eines Button aus der `ButtonGroup`

- Die `ButtonGroup` stellt sicher, das immer genau ein `RadioButton` aktiv ist
- `ActionListener` werden verwendet, um zu ermitteln welcher Button aktiviert wurde.

Beispiel Buttongroup

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

public class RadioButtonExample
 implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Button␣pressed,␣Action:␣"
    + e.getActionCommand());
  }
  public static void main(String[] args) {
    RadioButtonExample buttonExample =
     new RadioButtonExample();
    JFrame frame = // Erzeugt ein Fenster
     new JFrame("Transportmittel");
    Container contentPane = frame.getContentPane();
```

Beispiel Buttongroup (Forts.)

```
    ButtonGroup buttonGroup = new ButtonGroup();
    JRadioButton button =
     new JRadioButton("Auto", true);
    button.addActionListener(buttonExample);
    button.setActionCommand("auto");
    buttonGroup.add(button);
    contentPane.add(button, BorderLayout.NORTH);
    button = new JRadioButton("Zug2");
    button.addActionListener(buttonExample);
    button.setActionCommand("zug");
    buttonGroup.add(button);
    contentPane.add(button, BorderLayout.SOUTH);
    frame.setDefaultCloseOperation(
     JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }
}
```

JList

Wichtige Methoden

- `JList()` Erzeugt eine `List`
- `JList(Object[])` Erzeugt eine List
- `void setListData(Object[])` Liste mit Objekten füllen
- `void setListData(Vector)` Liste mit Objekten füllen
- `addListSelectionListener(ListSelectionListener)` fügt Listener hinzu
- `int getSelectedIndex()` Liefert den Index des selektierten Objektes

JList (Forts.)

Wichtige Methoden

- `int[] getSelectedIndices()` Liefert die Indices der selektierten Objekte
- `Object getSelectedValue()` Liefert das selektierte Objekt
- `Object[] getSelectedValues()` Liefert die selektierten Objekte
- `int getSelectionMode()` Liefert den Selektionsmodus: `SINGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION`, `MULTIPLE_INTERVAL_SELECTION`

Beispiel JList

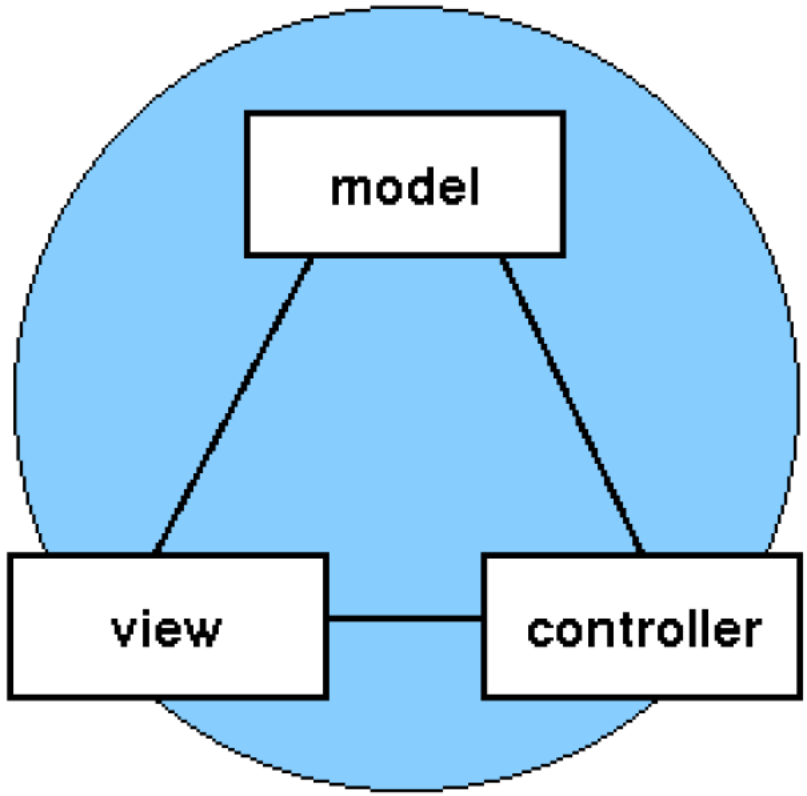
```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
public class ListExample
 implements ListSelectionListener {
  public void valueChanged(
    ListSelectionEvent e) {
    if (e.getValueIsAdjusting()) return;
    JList list = (JList) e.getSource();
    if (!list.isSelectionEmpty()) {
      String selected =
       (String) list.getSelectedValue();
      System.out.println("List␣Auswahl:"
       + selected);
    }
  }
```

Beispiel JList

```
  public static void main(String[] args) {
    JFrame frame = new JFrame("SwingExample");
    Container contentPane = frame.getContentPane();
    String[] liste =
     { "Rot", "Gelb", "Blau", "Schwarz" };
    JList list = new JList(liste);
    list.setSelectionMode(
     ListSelectionModel.SINGLE_SELECTION);
    list.addListSelectionListener(
     new ListExample());
    contentPane.add(list, BorderLayout.CENTER);
    frame.setDefaultCloseOperation(
     JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }
}
```

MVC-Architektur

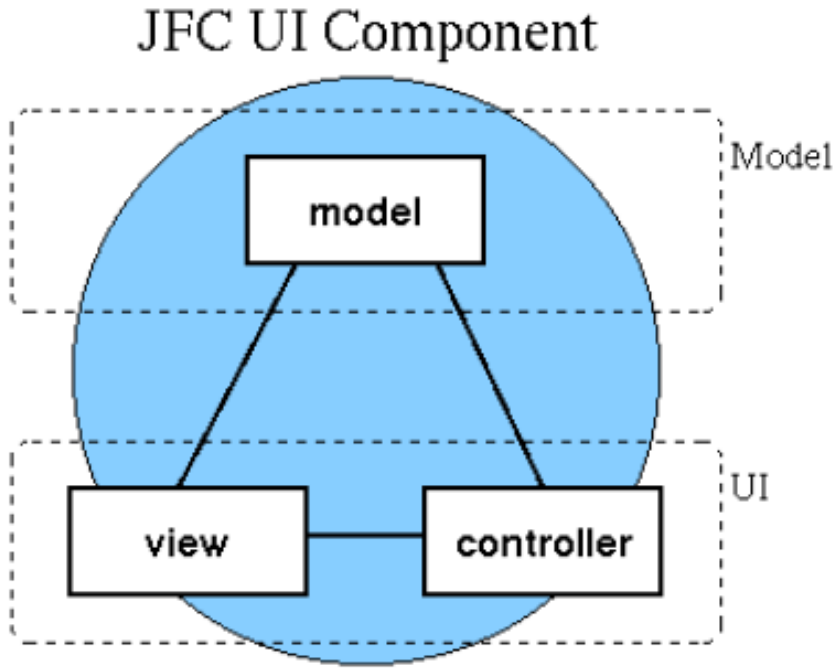
- Model: Datenmodell, das Daten speichert und manipuliert
- View: Darstellung des Inhalts des Datenmodells
- Controller: Steuert das Programm (Model, View) in Abhängigkeit von Benutzereingaben



Wichtig: Ein Model kann mehrere Views / Controller haben

MVC-Architektur in Swing

- Model:
  - Liefert internen Zustand
  - Manipuliert internen Zustand
  - Hinzufügen/Entfernen von Event-Listenern
  - Löst Events aus
- View + Controller:
  - Stellt die Daten dar
  - Liefert geometrische Informationen
  - Bearbeitet Events



MVC-Beispiele in Swing

Model	View-Controller	Listener
DefaultListModel	JList	ListDataListener
DefaultTableModel	JTable	TabelModelListener
DefaultListSelectionModel	JList, JTable	ListSelectionListener
DefaultTreeModel	JTree	TreeModelListener
TreeSelectionModel	JTree	TreeSelectionListener
DefaultBoundedRangeModel	Slider	ChangeListener
DefaultComboBoxModel	JComboBox	ListDataListener
DefaultButtonBodel	JButton,	Action-, Item-,
	JCheckBox	ChangeListener
	JRadioButton	

Beispiel ListModel

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

public class ListModelExample
 implements ActionListener {
  protected JList list;
  protected DefaultListModel listModel;

  public ListModelExample() {
    JFrame frame =
     new JFrame("ListModelExample");
    Container contentPane =
     frame.getContentPane();
    String[] liste =
     { "Rot", "Gelb", "Blau", "Schwarz" };
```

Beispiel ListModel (Forts.)

```
listModel = new DefaultListModel();
for (int i = 0; i < liste.length; i++)
  listModel.addElement(liste[i]);
list = new JList(listModel);
list.setSelectionMode(
 ListSelectionModel.SINGLE_SELECTION);
contentPane.add(new JScrollPane(list),
 BorderLayout.CENTER);
```

Beispiel ListModel (Forts.)

```
    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(2, 1));
    JButton button = new JButton("remove");
    button.addActionListener(this);
    panel.add(button);
    button = new JButton("insert");
    button.addActionListener(this);
    panel.add(button);
    contentPane.add(panel, BorderLayout.SOUTH);
    frame.setDefaultCloseOperation(
    JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
  }
```

Beispiel ListModel (Forts.)

```
  public void actionPerformed(ActionEvent e) {
    JButton button = (JButton) e.getSource();
    if (button.getText().equals("remove")) {
      if (!list.isSelectionEmpty()) {
        int index = list.getSelectedIndex();
        list.clearSelection();
        listModel.remove(index);
      }
    } else {
      listModel.addElement("Neue␣Farbe");
    }
  }

  public static void main(String[] args) {
    ListModelExample app = new ListModelExample();
  }
}
```

Zusammenfassung

- Datenbankzugriff erfolgt über JDBC mit executeQuery (Lesen) und executeUpdate (Schreiben).
- Grafik wie in AWT kann in Swing über die paint-Methode einer JComponent realisiert werden.
- Bilder (Pixelgrafik) kann geladen und vielfältig manipuliert werden.
- Für Zugriff und Manipulation von Audiodaten kann die Sound-API oder das JMF verwendet werden, die controllerUpdate-Methode dient zur Steuerung eines Players.
- Benutzungsschnittstellen lassen sich über eine Model-View-Controller-Architektur realisieren