Informatik B

Vorlesung 21

MVC, Swing

Rückblick

- AWT
  - Layout
  - Container
  - Steuerelemente
  - Events
- Applets

Swing

- Unter die Swing-Komponenten fallen neue grafische Elemente
- Diese sind, anders als die plattformabhängigen Peer-Komponenten des herkömmlichen AWTs, vollständig in Java implementiert
- Einige Swing-Komponenten wie z.B. ein **`JFrame`** basieren auf einer schwergewichtigen AWT-Komponente
- Alle Komponenten des Swing-Sets haben die Fähigkeit das Aussehen der Komponenten zur Laufzeit zu ändern, ohne das Programm neu zu starten (Pluggable Look & Feel)

Swing vs. AWT

- Swing bietet viel mehr Komponenten als AWT, so bietet AWT zum Beispiel keine Tabellen oder Bäume
- Schaltflächen und Labels können Symbole aufnehmen, die sie beliebig um Text angeordnet darstellen
- Swing-Komponenten können transparent und beliebig geformt sein
- Eine Schaltfläche kann wie unter Mac OS X abgerundet sein.
- Jede Swing-Komponente kann einen Rahmen bekommen
- AWT-Komponenten arbeiten nicht nach dem Model/View-Prinzip, nach dem die Daten getrennt von den Komponenten gehalten werden

Fenster

- Mit der Klasse **`JFrame`** kann ein Fenster geöffnet werden
- Soll der Closebutton mit Funktionalität versehen werden, muss kein Listener angehängt werden
- Dazu gibt es die Methode **`setDefaultCloseOperation(int status)`**
- Als Status kann gesetzt werden:
  - **`JFrame.DO_NOTHING_ON_CLOSE`**
  - **`JFrame.HIDE_ON_CLOSE`** (Defaultwert)
  - **`JFrame.DISPOSE_ON_CLOSE`**
  - **`JFrame.EXIT_ON_CLOSE`**
- Beispiel: **`swing1`**

JComponent

- Die aus AWT bekannten **`Component`**-Objekte sind in der Regel in Swing ebenfalls vorhanden
- Bei Swing ist den **`Component`**-Objekten ein „**`J`**“ vorangestellt
- Zudem bieten viele Swing-Components mehr Möglichkeiten
- Es können Bilder und Text auf Components verwendet werden
- Alle **`Component`**-Objekte sind in der Lage einen ToolTipText anzuzeigen
- Zur Beschriftungsanzeige können HTML-Tags verwendet werden
- Beispiel: **`swing2`**

Border

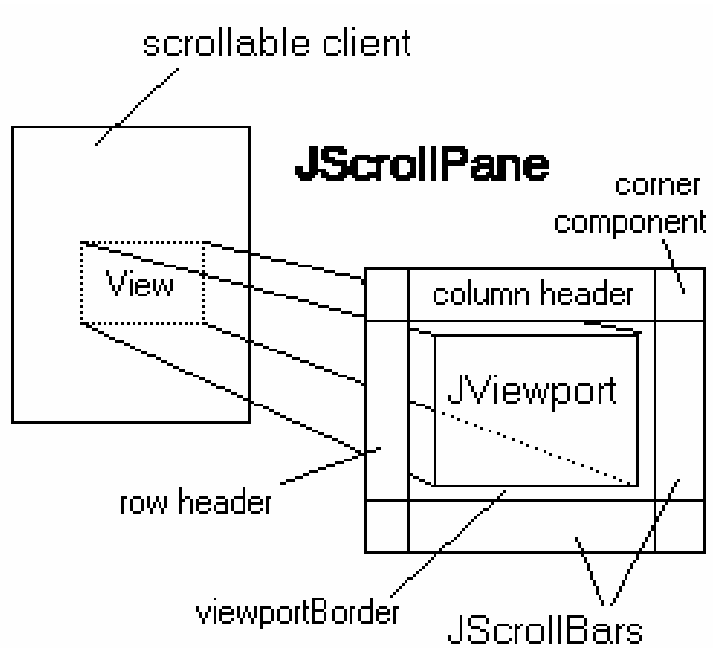
- Jeder Swing-Komponente kann mit der Methode **`setBorder()`** ein Rahmen zugewiesen werden
- Ein Rahmen ist eine Klasse, die das Interface **`Border`** implementiert
- Swing stellt einige Standardrahmen zur Verfügung:
  - **`AbstractBorder`** Abstrakte Klasse, die die Schnittstelle minimal implementiert
  - **`BevelBorder`** (Eingelassener) 3D-Rahmen
  - **`CompoundBorder`** Rahmen, der andere Rahmen aufnehmen kann
  - **`EmptyBorder`** Rahmen, dem freier Platz zugewiesen werden kann
  - **`EtchedBorder`** Noch deutlicher markierter Rahmen
  - **`LineBorder`** Rahmen in einer einfachen Farbe in gewünschter Dicke
  - **`MatteBorder`** Rahmen, der aus Kacheln von Icons besteht
  - **`SoftBevelBorder`** 3D-Rahmen mit besonderen Ecken
  - **`TitledBorder`** Rahmen mit einem String in einer gewünschten Ecke
- Beispiel: **`swing3`**

Container

- Zu den wichtigsten Containern in Swing zählen:
  - **`JPanel`** Ist im Wesentlichen eine **`JComponent`** mit der Möglichkeit, Objekte nach einem bestimmten Layoutverfahren anzuordnen
  - **`JScrollPane`** Kann Bereiche einer sehr großen Komponente mit Rollbalken anzeigen
  - **`JTabbedPane`** Zeigt Reiter in einem Karteikasten an
  - **`JSplitPane`** Ermöglicht die Darstellung zweier Komponenten über- oder nebeneinander, wobei ein so genannter Divider eine Größenveränderung erlaubt

JScrollPane

- Eine **`JScrollPane`** besitzt einen Client (das anzuzeigende Objekt)
- Darüber liegt ein Viewport, der durch die Scrollbars bewegt werden kann
- Beispiel: **`swing4`**



`JTabbedPane`

- Die **`JTabbedPane`** bietet die Funktionalität eines **`CardLayout`**
- Die einzelnen Registerreiter sind vorhanden und wechseln bei Klick die Ansicht
- Jede einzelne Pane kann mit einem ToolTip versehenen werden
- Die einzelnen Registerreiter können einen Titel und/oder ein Bild enthalten
- Die Registerreiter können mit Tastaturkürzel versehen werden
- Beispiel: **`swing5`**

`JCheckBox`

- Die **`JCheckBox`** ist eine Schaltfläche
- Es lassen sich verschiedene Grafiken für den eingeschalteten und ausgeschalteten Zustand zuweisen
- Im Konstruktor kann ein Wahrheitswert angeben werden, der bestimmt, ob das Feld am Anfang gesetzt ist oder nicht
- Ändert sich der Zustand eines Felds (Selektion oder Deselektion), wird ein **`ItemEvent`** an alle registrierten **`ItemListener`** weitergeleitet
- Beispiel: **`swing6`**

Look And Feel

- Das Aussehen der Komponenten lässt sich frei bestimmen und erzeugt so bei jedem Benutzer auf seiner Architektur die Illusion, es wäre eine plattformabhängige Applikation
- Das Programm kann sich also hinsichtlich des Aussehens in die anderen Programme eingliedern und fällt nicht als fremd auf
- Um das Aussehen von Java-Applikationen zu ändern, gibt es eine Reihe von Möglichkeiten:
  - Beim Programmstart einen Schalter setzen
  - Eine Konfigurationsdatei in das lib-Verzeichnis legen
  - Im Java-Programm von Hand das L&F verändern

Look And Feel : `UIManager`

- Das L&F von Applikationen lässt sich zur Laufzeit ändern
- Dazu muss die statische Methode **`setLookAndFeel()`** der Klasse **`UIManager`** aufgerufen werden
- Als Argument erwartet die Methode einen Klassennamen der das Aussehen bestimmt
- Einige spezielle L&F sind nicht auf jeder Architektur erlaubt
- Apple verbietet z.B. sein eigenes L&F auf Plattformen anderer Hersteller

Look And Feel : **`UIManager`**

- Das Setzen eines L & Fs mit **`setLookAndFeel()`** führt zu keinem Neuzeichnen der Komponenten
- Werden bereits Komponenten dargestellt muss das Neuzeichnen mit der Methode **`SwingUtilities.updateComponentTreeUI( component );`** angestoßen werden
- Welche L&F Klassen vorhanden sind kann mit der Klassenmethode **`getInstalledLookAndFeels()`** erfragt werden
- Beispiel: **`swing7`**

Weitere **`Component`**-Objekte

- Es gibt zahlreiche weitere Steuerelemente:
  - **`JTree`**: Darstellung von Baumstrukturen
  - **`JProgressBar`**: Fortschrittsbalken
  - **`JTable`**: Tabellenstrukturen
  - **`JSlider`**: Schieberegler
  - **`JScrollBar`**: Verschiebeleiste
- Beispiel: **`swing8`**

Menüs

- Eine Anwendung kann in Java auch mit einem Menü versehen werden
- Dazu kann einem **`JFrame`** mit der Methode **`setJMenuBar`** ein Menü zugeordnet werden
- Ein Menü ist wie die Container auch hierarchisch angeordnet
- Der Toplevel-Container ist **`JMenuBar`**
- Ein **`JMenuBar`**-Objekt enthält **`JMenu`** Einträge
- Ein **`JMenu-Objekt`** enthält **`JMenuItem`** Einträge

Menüs

- Die **`JMenuBar`** stellt das gesamte Menü dar
- Das **`JMenu`** stellt einen Menüabschnitt dar
- Ein **`JMenuItem`** ist ein konkreter Menüeintrag im Menü
- Um schnellen Zugriff auf das Menü zu haben können Tastaturkürzel für das Menü festgelegt werden
- Einem **`JMenu`** kann mit der Methode **`setMnemonic(int mnemonic);`** ein Kürzel übergeben werden
- Wählbare Zeichen sind in der Klasse **`KeyEvent`** definiert
- *Mnemoniks*, werden durch Unterstreichen hervorgehoben und sind immer eine Tastenkombination (in der Regel „ALT“+“ausgewähltes Zeichen“)

Menüs

- Einem **`JMenuItem`** kann direkt beim Erzeugen ein Tastaturkürzel übergeben werden
- Damit der Menüpunkt mit dem Tastaturkürzel aufgerufen werden kann, muss zunächst das Menü mit dem Item geöffnet sein
- Schneller geht es, wenn ein *Accelerator* gesetzt wird: **`setAccelerator(KeyStroke ks)`**
- Ein **`KeyStroke`** ist eine Tastaturkombination
- Z.B. „ALT+1“: **`KeyStroke.getKeyStroke(KeyEvent.VK_1, ActionEvent.ALT_MASK)`**
- Beispiel: **`menu1`**

Menüs

- Menüs können optisch strukturiert werden
- Dazu können **`JMenuItem`**-Objekte mit einem Icon versehen werden
- Desweiteren können Separatoren eingefügt werden, um das Menü zu unterteilen
- Anstelle eines **`JMenuItem`** Eintrages sind auch **`JRadioButtonMenuItem`** und **`JCheckBoxMenuItem`** Einträge möglich
- Beispiel: **`menu2`**

Menüs

- Zur weiteren Strukturierung können Untermenüs angelegt werden
- Dazu wird einem **`JMenu`**-Objekt anstelle eines **`JMenuItem`**-Objektes ein weiteres **`JMenu`**-Objekt hinzugefügt
- Auf diese Art und Weise sind tief verschachtelte Menüs möglich
- Beispiel: **`menü3`**

MVC

- Das *Model-View-Controller-Pattern* dient zur Trennung der Daten und der Sicht auf die Daten
- Das Model enthält den aktuellen Datenbestand
- Die *View* stellt lediglich Daten dar
- Der *Controller* verknüpft *View* und *Model* miteinander
- Dadurch ist gewährleistet, dass die *View* einfach ausgetauscht werden kann

Model

- Das *Model* enthält die darzustellenden Daten
- Woher die Daten kommen und wie diese zusammenhängen, spielt keine Rolle
- Das *Model* kennt weder die *View* noch die Steuerung, es weiß also gar nicht, wie, ob und wie oft es dargestellt und verändert wird
- Je nach Anwendung müssen Änderungen im Modell beobachtbar sein
- In der OOP wird dies unter anderem durch das *Observable-Pattern* implementiert

View (Präsentation)

- Die *View* ist für die Darstellung der relevanten Daten aus dem Modell zuständig
- Sie ist prinzipiell nicht für die Interaktion mit dem Benutzer verantwortlich, sondern
  - Darstellung der Daten
  - Aktualisierung der Darstellung
- Je nach Design leitet sie auch Benutzeraktionen (Events) an die Steuerung weiter

Controller (Steuerung)

- Der *Controller* verwaltet die *Views*
- Er enthält die Intelligenz und steuert den Ablauf (engl. Workflow) der Präsentation
- Der *Controller* sollte die *Views* und das Modell kennen
- Der *Controller* kann das Interface **`ChangeListener`** implementieren
- Der *Controller* selbst wird dann als Listener an das Modell gehängt
- Ändert sich das Modell kann der *Controller* darüber alle *Views* benachrichtigen
- Er nimmt Benutzeraktionen der Sichten entgegen, wertet diese aus und agiert entsprechend
- Beispiel: **`mvc1`**

MVC bei Swing

- In Java sind die *View* und der Controller durch ein **`ComponentUI`**-Objekt dargestellt
- Da das Aussehen und Verhalten von Java-Komponenten frei bestimmt werden kann, gibt es für alle konkreten Swing-Komponenten ein **`ComponentUI`**-Objekt, das die Darstellung und Benutzeraktionen übernimmt
- Ein **`JButton`**-Objekt selbst besitzt z.B. keine **`paint()`**-Methode
- Die Daten des Buttons befinden sich im **`ButtonModel`**

MVC bei Swing

- Für manche Komponenten sind sehr unterschiedliche Modelle notwendig
- Eine Schaltfläche visualisiert meistens eine Zeichenkette
- Eine Tabelle repräsentiert aber nicht immer nur einfache Texte, sondern auch oft komplexe Objektstrukturen
- Dafür werden spezielle *Model* implementiert, die Daten für die Ansicht zur Verfügung stellen

MVC bei Swing

- **`ListModel`**
- **`ComboBoxModel`**
- **`ButtonModel`**
- **`BoundedRangeModel`**
- **`SingleSelectionModel`**
- **`TableModel`**
- **`TableColumnModel`**
- **`TreeModel`**
- **`TreeSelectionModel`**

MVC bei Swing

- Der Vorteil des MVC liegt zum einen bei der Mehrfachverwendung eines *Model*
- So können sich verschiedene grafische Componenten ein *Model* teilen
- Ein anderer Vorteil ist die Implementierung eines eigenen *Model*, welches noch auf andere Ereignisse reagieren kann
- Es werden also nicht mehr Werte an einer grafischen **`Component`** gesetzt, sondern am entsprechenden *Model*
- Die grafischen Componenten besitzen **`ChangeListener`**, die sich ans *Model* hängen
- Ein *Model* sollte daher immer die Möglichkeit haben **`ChangeListener`**-Objekte aufzunehmen
- Beispiel: **`swing9`**

Zusammenfassung

- Swing
- JComponent
- Look and Feel
- MVC
- Menüs

Ausblick

- Besprechung der Evaluation
- Midlet-Programmierung