Programmieren in Java

Vorlesung 04: Graphische Benutzerschnittstelle am Beispiel von Swing

Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2013

Inhalt

Graphische Benutzerschnittstellen

- Fallstudie
- Packages
- JFrame
- BorderLayout
- JLabel
- ActionListener
- Anonyme Klassen
- Geschachtelte Klassen
- JPanel
- Color
- BoxLayout

Fallstudie

- Einstieg in die Programmierung mit Swing
- Einstiegsbeispiel: GUI zum Durchblättern von Strassenkarten

Packages

- Package `monopoly`: Code für die Programmlogik.
- Neue Package `monopoly.viewer`: Code für Benutzerschnittstelle — getrennt von Programmlogik, sollte nicht vermischt werden.
- Obwohl die Quelldateien für `monopoly.viewer` im Unterverzeichnis `viewer` vom Verzeichnis `monopoly` abgelegt werden müssen, *besteht logisch keine Beziehung: keine Vererbung, keine automatische Übernahme von Bindungen*.
- Mit anderen Worten: Obwohl die Benennung der Packages eine Hierarchie suggerieren, ist dies *nicht* der Fall.

JFrame

- Einstiegspunkt: Klasse `javax.swing.JFrame`
- Instanzen entsprechen einem Fenster auf dem Bildschirm
- Typische Initialisierung:

```
JFrame frame = new JFrame("Monopoly Viewer");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
addContents(frame.getContentPane());
frame.pack();
frame.setVisible(true);
```

JFrame/2

- Argument des Konstruktors = Titel des Fensters
- Die *default close operation* bestimmt, was passiert, wenn das Fenster geschlossen wird. In diesem Falle wird die Anwendung geschlossen.
- Zum Fenster gehört eine "Arbeitsfläche", auf die mit `getContentPane()` zugegriffen werden kann.
- Auf der Arbeitsfläche werden weitere GUI-Komponenten (durch die Methode `addContents()`) abgelegt.
- Die Methode `pack()` bestimmt berechnet ein Layout für die Inhalte der Arbeitsfläche, das den geringsten Platzverbrauch hat.
- Ein Fenster wird zuerst unsichtbar erzeugt und muss auf "visible" gesetzt werden um auf dem Bildschirm dargestellt zu werden.
- Weitere Infos: `http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html`

JButton

- Die Instanzen der Klasse `JButton` sind Knöpfe, die angeklickt werden können.
- Das Argument des Konstruktors ist der String, der auf dem Knopf erscheint:

```
JButton close = new JButton("close");
```

- Weitere Konfiguration durch Setter-Methoden wie `setForeground()` oder `setBackground()`
- Weitere Infos: `http://docs.oracle.com/javase/tutorial/uiswing/components/button.html`

BorderLayout

- Jede Arbeitsfläche (Typ `Container`) besitzt einen Layoutmanager, der die Platzierung der untergeordneten Komponenten regelt.
- Die Arbeitsfläche des Top-level Fensters verwendet dafür `BorderLayout`.
- Diese Layout verwaltet fünf Stellen der Arbeitsfläche, genannt `NORTH`, `SOUTH`, `WEST`, `EAST` und `CENTER` entsprechend ihrer physikalischen Anordnung.
- Zum Ablegen auf der Arbeitsfläche dient die `add()` Methode mit einem Argument, dass die Platzierung regelt.

BorderLayout / 1

Beispiel

```
JButton close = new JButton("close");
JButton previous = new JButton("<-");
JButton next = new JButton("->");

vmain.add(previous, BorderLayout.WEST);
vmain.add(next, BorderLayout.EAST);
vmain.add(close, BorderLayout.SOUTH);
```

Weitere Infos zum BorderLayout http://docs.oracle.com/javase/tutorial/uiswing/layout/border.html

JLabel

- Eine Komponente, die einen Text oder eine Graphik darstellt und sonst keine weitere Funktion hat.
- Das Konstruktorargument ist der initiale Text.

```
JLabel content = new JLabel("content");
```

- Weitere Eigenschaften können konfiguriert werden:

```
content.setPreferredSize(new Dimension(100, 30));
content.setHorizontalAlignment(SwingConstants.CENTER);
```

- Einfügen ins Layout wie gehabt:

```
vmain.add(content, BorderLayout.NORTH);
```

- Weiteres: `http://docs.oracle.com/javase/tutorial/uiswing/components/label.html`

ActionListener

- Knöpfe (buttons) erhalten ihre Funktion durch Hinzufügen eines `ActionListener`:

```
close.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.exit(0);
    }
});
```

- Das Argument ist ein Objekt, das das Interface `ActionListener` implementiert.
- So ein Objekt wird im Beispiel mit Hilfe einer *anonymen Klasse* definiert.

Anonyme Klassen

- Wenn `new` auf ein Interface oder eine abstrakte Klasse angewendet wird, dann kann ein Rumpf wie für eine Klassendefinition folgen.
- Insbesondere müssen dort die Methoden implementiert werden, die vom Interface gefordert werden bzw. in der Klasse als abstrakt definiert sind.
- Java erzeugt aus dieser Syntax eine neue Klasse, die ansonsten nicht zugreifbar ist, und erzeugt eine neue Instanz davon.
- Der Rumpf der anonymen Klasse kann lokale Variablen verwenden, die in der aktuellen Methode definiert sind. Diese Variablen müssen als `final` definiert sein.

Anonyme Klassen / Beispiel

```
final ContentMaster cm = new ContentMaster();
previous.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        content.setText(cm.getPrevious());
    }
});
next.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        content.setText(cm.getNext());
    }
});
```

- Die `setText()` Methode kann den Text eines Labels zur Laufzeit abändern. Hier geschieht das beim Drücken auf die Knöpfe `previous` und `next`.

Geschachtelte Klassen (Nested Classes)

- Eine anonyme Klasse ist ein Spezialfall einer *geschachtelten Klasse*, d.h. eine Klasse, die innerhalb einer anderen Klasse definiert werden kann.
- Die geschachtelte Klasse ist gemäß ihrer Sichtbarkeitsdeklaration zugreifbar:
  - private geschachtelte Klasse

    ```
    public class VMain {

            static private class ContentMaster {
    ```

    `ContentMaster` ist nur innerhalb von `VMain` bekannt.
  - öffentliche geschachtelte Klasse

    ```
    public class VMain {

            static public class ContentMaster {
    ```

    ist von außerhalb als `VMain.ContentMaster` zugreifbar.

Geschachtelte Klasse für Anonyme Klasse

- Anstelle der obigen anonymen Klassen als `ActionListener` kann ein beliebiges anderes Objekt verwendet werden, solange es das Interface implementiert.
- Beispiel:

```
private static class Previous implements ActionListener {
  public Previous (Container content, ContentMaster cm) {
    this.content = content; this.cm = CM;
  }
  public void actionPerformed (ActionEvent e) {
    content.setText (cm.getNext());
  }
}
...
previous.addActionListener (new Previous (content, cm));
```

JPanel

- Eine Instanz von `JPanel` ist eine GUI-Komponente, die selbst wieder als Arbeitsfläche dienen kann. Sie hat keine speziellen Konfigurationsmöglichkeiten.
- Beispiel:

```
private static Component makeStreetHeader(Street s) {
  JPanel header = new JPanel(new BorderLayout());
  header.setPreferredSize(new Dimension(STREET_WIDTH, STREET_H
  header.setMaximumSize(new Dimension(STREET_WIDTH, STREET_HEA
  header.setBackground(VGroup.getColor(s.getColorGroup()));
  // ...
  return header;
}
```

JPanel / 2

- Das Konstruktorargument ist ein Layoutmanager, in diesem Fall das bereits bekannte `BorderLayout`, da der Standard-Layoutmanager für ein `JPanel` nicht passend ist.
- Die Konfiguratoren `setPreferredSize` und `setMaximumSize` geben Hinweise an den Layoutmanager. Dieser versucht, den PreferredSize zu verwenden, und verspricht, den MaximumSize nicht zu überschreiten.
- `setBackground` definiert die Hintergrundfarbe (siehe folgende Folie)
- Weitere Infos: `http://docs.oracle.com/javase/tutorial/uiswing/components/panel.html`

JPanel / BorderLayout

```
JLabel line = new JLabel(s.getName());
line.setHorizontalAlignment(SwingConstants.CENTER);
line.setForeground(Color.WHITE);
header.add(line, BorderLayout.NORTH);
```

- Einfügen von Elementen ins `JPanel` mittels `add` unter Angabe eines Hinweises an den Layoutmanager.
- `HorizontalAlignment` gibt an, wie der Layoutmanager das Element ausrichten soll.
- `Foreground` ist die Farbe, in der der Text des JLabel erscheinen soll.

Color

- Farben werden durch Elemente der Klasse `java.awt.Color` definiert und zwar durch die R, G und B Komponenten, die einem `Color` Konstruktor übergeben werden.
- Für einige Farben gibt es vordefinierte Konstanten, z.B. `Color.WHITE`, `Color.BLACK` usw.
- Die statische Methode `Color.decode` übersetzt einen String mit einer hexadezimalen Spezifikation eines 24Bit RBG Werts in eine Farbe.
- Beispiel

```
static private String[] colors = {
    "0x8B4513", "0x87CEEB", "0x9932CC", "0xFFA500",
    "0xFF0000", "0xFFFF00", "0x00FF00", "0x0000FF"};

public static Color getColor(Group g) {
    return Color.decode(colors[g.ordinal()]);
}
```

BoxLayout

```
private static Component makeStreetPanel(Street s) {
  JPanel spane = new JPanel();
  spane.setLayout(new BoxLayout(spane, BoxLayout.Y_AXIS));
  spane.setPreferredSize(new Dimension(STREET_WIDTH, STREET_HE
  spane.setBackground(Color.WHITE);
  spane.setBorder(BorderFactory.createLineBorder(Color.BLACK,
```

- Das Top-Level Panel für die Darstellung einer Straße verwendet nicht das `BorderLayout`, sondern das besser geeignete `BoxLayout`. Das ordnet die Komponenten entweder horizontal oder vertikal in einer Reihe an.
- Der Konstruktor fürs `BoxLayout` benötigt seinen Container als Argument und einen Hinweis, ob die Ausrichtung horizontal oder vertikal erfolgen soll.
- `Border` kann auf eine Spezifikation für die Ornamentierung des Randes gesetzt werden. In diesem Fall ist es eine schwarze Linie der Breite 2 Punkte.

JPanel / BoxLayout

```
  spane.add(makeStreetHeader(s));
  spane.add(Box.createVerticalGlue());
  spane.add(makeStreetFooter());
  return spane;
}
```

- Einfügen in den Container: wie gehabt.
- Das `BoxLayout` stellt verschiedene Platzhalter zum Auffüllen eines Layouts zur Verfügung. `VerticalGlue` ist ein flexibler Platzhalter, der sich so weit vertikal ausdehnt, wie notwendig um den Platz zu füllen. Es gibt Varianten für horizontalen Platz und mit fester Größe.