Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

JButton, JCheckBox, JRadioButton

In Swing existieren drei wesentliche Klassen für Schaltflächen:

- JButton für normale Schaltflächen,
- JCheckBox für Ankreuzfelder,
- JRadioButton für Optionsfelder (*radio button*)

Diese Klassen erben von der Klasse `AbstractButton`.

Die Klasse AbstractButton

Die Klasse `AbstractButton` bietet folgende wesentliche Methoden:

`void setText(String text)` setzt den Text der Schaltfläche,

`void setIcon(Icon icon)` setzt das Icon der Schaltfläche,

`void addActionListener(...)` registriert einen neuen `ActionListener`,

`void setActionCommand(String command)` definiert einen Text, der einem ausgelösten `ActionEvent` mitgegeben wird.

getActionCommand/setActionCommand

Wozu dient die Methode `setActionCommand`?

In der Praxis ist ein `ActionListener` häufig bei mehreren Schaltflächen angemeldet, und in der Methode `actionPerformed` soll je danach, welche Schaltfläche betätigt wurde, eine unterschiedliche Reaktion erfolgen. Diese Unterscheidung kann erfolgen:

- Nach dem Quellobjekt des Events (`e.getSource()`) oder
- nach dem Aktions-Text des Events (`e.getActionCommand()`)
  Dieser Text besteht entweder aus:
  - dem Text der Schaltfläche oder
  - dem durch `setActionCommand(...)` definierten Text der Schaltfläche.

Beispiel

```
public class MyApp extends JFrame implements ActionListener {
    public MyApp() throws HeadlessException {
        super("Meine Anwendung");
        this.setLayout(new GridLayout(0,1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton b = new JButton("OK");
        b.addActionListener(this);
        b.setActionCommand("OK1");
        this.add(b);
        b = new JButton("OK");
        b.setActionCommand("OK2");
        b.addActionListener(this);
        this.add(b);
        b = new JButton("OK");
        b.setActionCommand("OK3");
        b.addActionListener(this);
        this.add(b);
```

Beispiel (Forts.)

```
        this.pack();
        this.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println(e.getActionCommand());
    }

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


Die Klasse JCheckBox

JCheckBox dient der Darstellung von Kontrollfeldern zum Ankreuzen. Diese können selektiert oder deselektiert sein. Für die Änderung bzw. Abfrage des Status der Selektion existieren die Methoden

boolean getSelected() `true`, falls das Feld selektiert ist, sonst `false`

void setSelected(boolean b) selektiert das Feld (`true`) bzw. deselektiert (`false`) es.

Beim Selektieren oder Deselektieren des Kontrollfeldes wird ein ItemEvent ausgelöst, das bei einem mit addItemListener hinzugefügten ItemListener der Methode itemStateChanged übergeben wird.

Beispiel

```
public class MyApp extends JFrame implements ItemListener {
    public MyApp() throws HeadlessException {
        super("Meine Anwendung");
        this.setLayout(new GridLayout(0,1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JCheckBox b = new JCheckBox("eins");
        b.addItemListener(this);
        this.add(b);
        b = new JCheckBox("zwei");
        b.addItemListener(this);
        this.add(b);
        b = new JCheckBox("drei");
        b.addItemListener(this);
        this.add(b);

        this.pack();
        this.setVisible(true);
    }
```

Beispiel (Forts.)

```
    @Override
    public void itemStateChanged(ItemEvent e) {
        JCheckBox b = (JCheckBox) e.getSource();
        String val = e.getStateChange()==ItemEvent.SELECTED ?
                    "selektiert" : "deselektiert";
        System.out.println(b.getActionCommand()+": "+val);
    }

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


Die Klasse JRadioButton

Bei Optionsfeldern, so genannten *Radio-Buttons*, kann aus einer Gruppe von Feldern immer nur eines selektiert sein. Hierzu dient die Klasse `JRadioButton`.

Sich gegenseitig ausschließende Radio-Buttons werden hierbei einer `ButtonGroup` zugeordnet, die dafür sorgt, dass bei Auswahl eines Feldes alle anderen deselektiert werden.

```
ButtonGroup g = new ButtonGroup();
JRadioButton b = new RadioButton("eins");
g.add(b);
```

Die Zuordnung zu einer `ButtonGroup` hat keinen Einfluß auf die grafische Anordnung der Felder.

Beispiel

```
public class MyApp extends JFrame implements ActionListener {
    public MyApp() {
        super("Meine Anwendung");
        this.setLayout(new GridLayout(0,1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        ButtonGroup buttons = new ButtonGroup();
        JRadioButton b = new JRadioButton("eins");
        buttons.add(b);
        b.addActionListener(this);
        this.add(b);
        b = new JRadioButton("zwei");
        buttons.add(b);
        b.addActionListener(this);
        this.add(b);
        b = new JRadioButton("drei");
        buttons.add(b);
        b.addActionListener(this);
        this.add(b);
```

Beispiel (Forts.)

```
        this.pack();
        this.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Auswahl: "+e.getActionCommand());
    }

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


Events und Listener in Java

Wann immer in grafischen Anwendung in Java „etwas passiert“, sind daran zwei Dinge beteiligt:

Events Diese beschreiben:

- Art des Events
- Quelle
- spezifische Informationen (z. B. Maus-Koordinaten, gedrückte Shift- oder Alt-Taste)

Listener die auf Aktionen lauschen und diese Events geeignet auswerten.

Events

Die in AWT und Swing auftretenden Events erben direkt oder indirekt von der Klasse `AWTEvent`.

Konstruktor und Methoden:

AWTEvent(Object source, int id) wobei `source` die Quelle des Events (dies kann z. B. eine Instanz von `JButton` oder `JTextField` sein) und `id` den Typ des Events kennzeichnet.

Object getSource() gibt die Quelle des Events zurück.

int getID() gibt den Event-Typ zurück.

Die Klasse ActionEvent

Direkte Aktionen, die mit einer Komponente verknüpft sind, können i. A. am Einfachsten über `ActionEvents` abgehandelt werden, die als Parameter der Methode `actionPerformed` eines ActionListeners auftauchen.

Solche Aktionen können auf verschiedenen Wegen auftreten (Selektion eines Listenelements, Klicken auf einen Button, Eingabe von Enter in einem Textfeld), allerdings stecken im `ActionEvent` nur relativ wenig Informationen:

- die Quelle des Events (`getSource()` geerbt von `AWTEvent`) und
- evtl. bei der Auswahl gehaltene Shift-, CTRL- oder ALT-Tasten

Beispiel: Testen auf Shift- und ALT-Taste

```
public class MyApp extends JFrame implements ActionListener {
    public MyApp() throws HeadlessException {
        super("Meine Anwendung");
        this.setLayout(new GridLayout(0,1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton b = new JButton("OK");
        b.addActionListener(this);
        this.add(b);

        this.pack();
        this.setVisible(true);
    }
```


Beispiel (Forts.)

```
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button gedrückt");
        if ((e.getModifiers() & ActionEvent.SHIFT_MASK) != 0) {
            System.out.println("mit Shift");
        }
        if ((e.getModifiers() & ActionEvent.ALT_MASK) != 0) {
            System.out.println("mit ALT");
        }
    }

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

Weitere Events

Für komplexere Auswertungen von Events reicht `ActionEvent` nicht aus. Hier benötigt man speziellere Events, die auch mit speziellen Listener-Versionen behandelt werden. Dies sind z. B.

MouseEvent Klick-Event der Maus, hierbei können Maustaste, Art der Aktion (Doppelklick) und Pixelposition der Aktion ausgewertet werden, wird den Methoden von `MouseListener` und `MouseMotionListener` als Parameter übergeben,

KeyEvent erlaubt die Auswertung der gedrückten Taste und evtl. Modifizierer, wird den Methoden von `KeyListener` übergeben,

WindowEvent für Aktionen mit dem Fenster (verkleinern, vergrößern, schließen), wird den Methoden von `WindowListener` übergeben.

Beispiel

In einer Anwendung soll ein Element aus einer Liste bei folgenden Ereignissen in ein Label übernommen werden:

- Klick auf eine Schaltfläche „Übernehmen“,
- Doppelklick auf das Listenelement,
- Eingabe von „Enter“ in der Liste

Nur das erste Ereignis kann über `ActionListener` implementiert werden, da bei den beiden anderen die Maus- bzw. Tastenaktion speziell ausgewertet werden muss.

Implementierung

```
public class MyApp extends JFrame implements ActionListener,
                                   MouseListener, KeyListener {
    private JLabel output;
    private JList list;
    private JButton button;

    public MyApp() {
        super("Meine Anwendung");
        this.setLayout(new GridLayout(0,1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        String[] items = {"Eins", "Zwei", "Drei", "Vier"};
        list = new JList(items);
        list.addMouseListener(this);
        list.addKeyListener(this);
        add(list);

        output = new JLabel("Keine Auswahl");
        add(output);
```

Implementierung (Forts.)

```
        button = new JButton("Übernehmen");
        button.addActionListener(this);
        add(button);

        this.pack();
        this.setVisible(true);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        output.setText((String)list.getSelectedValue());
    }
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount()==2) {
            actionPerformed(new ActionEvent(e.getSource(),
                        ActionEvent.ACTION_PERFORMED,null));
        }
    }
```


Implementierung (Forts.)

```
    @Override
    public void keyTyped(KeyEvent e) {
        if (e.getKeyChar()==KeyEvent.VK_ENTER) {
            actionPerformed(new ActionEvent(e.getSource(),
                        ActionEvent.ACTION_PERFORMED,null));
        }
    }
    @Override
    public void mouseEntered(MouseEvent arg0) {}
    @Override
    public void keyPressed(KeyEvent arg0) {}
    @Override
    public void keyReleased(KeyEvent arg0) {}
    @Override
    public void mouseExited(MouseEvent arg0) {}
    @Override
    public void mousePressed(MouseEvent arg0) {}
```


Implementierung (Forts.)

```
    @Override
    public void mouseReleased(MouseEvent arg0) {}

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

Adapterklassen für Events

An dem vorigen Beispiel mit der Behandlung eines Tastatur- und eines Maus-Ereignisses wird ein Problem bei der Implementierung von speziellen Listener-Varianten deutlich: Es müssen viele Methoden implementiert werden, auch wenn nur für wenige Aktionen tatsächlich eine Implementation erfolgen soll. Dies wird schnell unübersichtlich!

Aus diesem Grund existieren für komplexe Events Adapterklassen, die das entsprechende Listener-Interface (mit leeren Methoden) implementieren:
`MouseListener` ↔ `MouseAdapter`,
`MouseMotionListener` ↔ `MouseMotionAdapter`,
`KeyListener` ↔ `KeyAdapter`,
`WindowListener` ↔ `WindowAdapter`

Neue Version des Beispiels mit Inneren Klassen

Nun soll das Beispiel neu implementiert werden, indem für das Maus- bzw. Tastatur-Ereignis neue Klassen entwickelt werden, die von den entsprechenden Adaptern erben.

Da diese Klassen nur innerhalb der Anwendungsklasse selber benötigt werden, ist es hier möglich, sich den Aufwand „normaler“ Klassen in eigenen Quelltext-Dateien zu ersparen. Stattdessen können diese Klassen innerhalb der Anwendungklasse definiert werden, die dann nur innerhalb dieser Anwendungsklasse bekannt sind. Hierdurch bleibt die Struktur des Java-Projektes übersichtlicher.

Solche Klassendefinitionen innerhalb anderer Klassen nennt man *Innere Klassen*

Implementierung

```
public class MyApp extends JFrame implements ActionListener {
    private JLabel output;
    private JList list;
    private JButton button;

    public MyApp() {
        super("Meine Anwendung");
        this.setLayout(new GridLayout(0,1));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        String[] items = {"Eins", "Zwei", "Drei", "Vier"};
        list = new JList(items);
        list.addMouseListener(new MyMouseAdapter());
        list.addKeyListener(new MyKeyAdapter());
        add(list);
```

Implementierung (Forts.)

```
        output = new JLabel("Keine Auswahl");
        add(output);

        button = new JButton("Übernehmen");
        button.addActionListener(this);
        add(button);

        this.pack();
        this.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        output.setText((String)list.getSelectedValue());
    }
```


Implementierung (Forts.)

```
    class MyMouseAdapter extends MouseAdapter {
        @Override
        public void mouseClicked(MouseEvent e) {
            if (e.getClickCount()==2) {
                actionPerformed(new ActionEvent(e.getSource(),
                        ActionEvent.ACTION_PERFORMED,""));
            }
        }
    }
```


Implementierung (Forts.)

```
    class MyKeyAdapter extends KeyAdapter {
        public MyKeyAdapter() {
        }
        @Override
        public void keyTyped(KeyEvent e) {
            if (e.getKeyChar()==KeyEvent.VK_ENTER) {
                actionPerformed(new ActionEvent(e.getSource(),
                        ActionEvent.ACTION_PERFORMED,""));
            }
        }
    }
}
```