Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

Menüs

Action

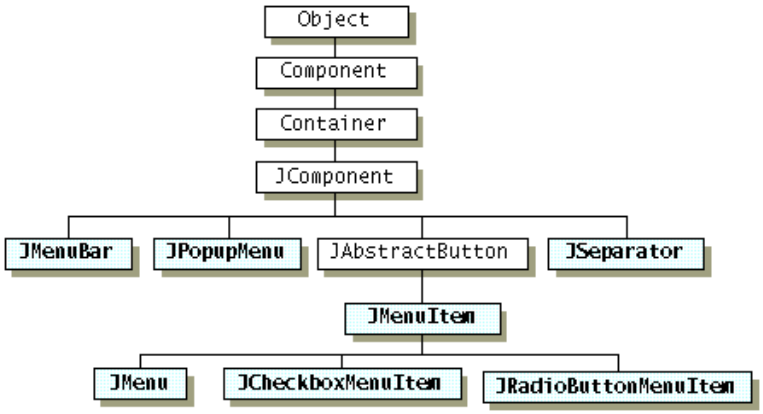
GridBagLayout

Menüs

In Fenstern von grafischen Anwendungen kennt man folgende Arten von Menüs:

- Menüleisten (am oberen Rand des Fensters)
- Symbolleisten
- Popup-Menüs, die mit der rechten Maustaste geöffnet werden.

Menü-Klassen in Java



Einträge in Menüs

Jedes Menü kann folgende Einträge enthalten:

- Normale Einträge (Klasse `JMenuItem`)
- Ankreuzfelder (Klasse `JCheckBoxMenuItem`)
- Optionsfelder (Klasse `JRadioButtonMenuItem`)
- Untermenüs (Klasse `JMenu`), die wiederum o. a. Einträge und Trennlinien enthalten können.

Für das Hinzufügen von Einträgen aller Art existiert die Methode `add(...)`. Separator-Linien in Untermenüs werden mit `addSeparator()` hinzu gefügt.

Einfaches Beispiel

```
public class MyApp extends JFrame {
    public MyApp()  {
        super("Meine Anwendung");
        setSize(300,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JMenuBar menuBar = new JMenuBar();
        JMenu menu;
        JMenuItem item;

        menu = new JMenu("Datei");
        item = new JMenuItem("Öffnen");
        menu.add(item);
        item = new JMenuItem("Speichern");
        menu.add(item);
        menuBar.add(menu);
```

Einfaches Beispiel (Forts.)

```
            menu = new JMenu("Hilfe");
            item = new JMenuItem("Über diese Anwendung");
            menu.add(item);
            menuBar.add(Box.createHorizontalGlue());
            menuBar.add(menu);
            setJMenuBar(menuBar);

            setVisible(true);
        }

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


Die Klasse JMenu

Die Klasse `JMenu` dient zum Erstellen eines Untermenüs, in dem neue Einträge vorhanden sein können.
Bei den meisten Anwendungen bestehn die Menü-Einträge im MenuBar selber aus solchen Untermenüs.

Dem Untermenü werden neue Elemente mit der Methode `add(JMenuItem)` hinzu gefügt. Es ist im Gegenzug auch möglich, Elemente aus einen Untermenü zu entfernen:

`remove(JMenuItem)` für ein Objekt vom Typ `JMenuItem`,

`remove(int)` mit Angabe der Position (beginnend bei 0),

`removeAll()` zum Entfernen aller Elemente.

Beispiel

```
menu = new JMenu("Datei");
item = new JMenuItem("Öffnen");
menu.add(item);

item = new JMenuItem("Speichern");
menu.add(item);

JMenu submenu = new JMenu("Letzte Dateien");
submenu.add("Datei 1");
submenu.add("Datei 2");
menu.add(submenu);

menuBar.add(menu);
```

Checkboxen und Radiobuttons

Da die Klassen `JCheckBoxMenuItem` und `JRadioButtonMenuItem` von der Klasse `MenuItem` erben, können diese einfach zu Menüs hinzugefügt werden. Hierbei gilt, dass Radio-Buttons, einer `ButtonGroup` zugeordnet werden, die überwacht, dass nur einer der Buttons selektiert ist.

Beispiel:

```
ButtonGroup sprachen = new ButtonGroup();
JMenu submenu = new JMenu("Sprache");
rbItem = new JRadioButtonMenuItem("Deutsch");
sprachen.add(rbItem);
submenu.add(rbItem);
rbItem = new JRadioButtonMenuItem("Englisch");
sprachen.add(rbItem);
submenu.add(rbItem);
menu.add(submenu);
menu.add(new JCheckBoxMenuItem("Regelmäßig speichern"));
```


Accelerator Keys und Mnemonics

Zusätzlich zur Maus sollen Menü-Einträge oft auch per Tastatur zugänglich sein:

- Ein Menüeintrag aus der Menüleiste soll mit `ALT+Taste`, Einträge aus einem aufgeklappten Menü per Taste gewählt werden können, wobei der entsprechende Buchstabe unterstrichen ist. Zu diesem Zweck dient die Methode `setMnemonic(...)`.
- Wichtige Menüeinträge sollen direkt ohne Aufklappen des Untermenüs ausgewählt werden können. Dies sind häufig Aktionen wie Öffnen (`Ctrl-O`) oder Speichern (`Ctrl-S`). Zu diesem Zweck dient die Methode `setAccelerator(KeyStroke)`. Der KeyStroke für `Ctrl-S` wird dabei repräsentiert durch
  `KeyStroke.getKeyStroke(KeyEvent.VK_S, KeyEvent.CTRL_MASK)`
  oder kürzer
  `KeyStroke.getKeyStroke("ctrl S")`

Beispiel

```
menu = new JMenu("Datei");
menu.setMnemonic(KeyEvent.VK_D);
item = new JMenuItem("Öffnen");
item.setAccelerator(KeyStroke.getKeyStroke("ctrl O"));
item.addActionListener(this);
menu.add(item);
item = new JMenuItem("Speichern");
item.setAccelerator(KeyStroke.getKeyStroke("ctrl S"));
item.addActionListener(this);
menu.add(item);
item = new JMenuItem("Speichern unter ...");
item.setMnemonic(KeyEvent.VK_U);
item.addActionListener(this);
menu.add(item);
```

Toolbars

Für die Erstellung eines Toolbars wird die Klasse `JToolBar` verwendet. Dieser können Buttons hinzugefügt werden, die Icons oder Text enthalten. Aber wie definiert man einen Button, der ein Bild enthält?

Hierfür wird eine (selbst definierte) statische Methode verwendet, die eine Grafikdatei in ein Icon verwandelt:

```
private static ImageIcon createIcon(String fname) {
    return new ImageIcon("/images/"+fname);
}
```

Hier wird davon ausgegangen, dass sich die Dateien im Unterverzeichnis `images` befinden.

Beispiel

```
JToolBar toolbar = new JToolBar();
JButton button;
toolbar.setFloatable(false);

button = new JButton(createIcon("Open24.gif");
button.setActionCommand("Öffnen");
button.addActionListener(this);
toolbar.add(button);

button = new JButton(createIcon("Save24.gif"));
button.setActionCommand("Speichern");
button.addActionListener(this);
toolbar.add(button);

button = new JButton(createIcon("SaveAs24.gif"));
button.setActionCommand("Speichern unter ...");
button.addActionListener(this);
toolbar.add(button);
add(toolbar,BorderLayout.PAGE_START);
```


Einschub: Fertige Icon-Bibliothek

Sun stellt das *Java look and feel Graphics Repository* zur Verfügung, in dem sich Standard-Icons befinden: http://java.sun.com/developer/techDocs/hi/repository/.

Diese lassen sich als `.jar`-Datei herunter laden, die dann in den CLASSPATH einer Anwendung integriert werden kann. Die am häufigsten verwendeten Icons befinden sich dort im Abschnitt *General*.

Hierfür muss die Methode `createIcon` folgendermaßen geändert werden:

```
private static ImageIcon createIcon(String fname) {
    return new ImageIcon(
        App.class.getResource("/toolbarButtonGraphics/general/"+fr
    );
}
```

Popup-Menüs

Ein Popup-Menu öffnet sich an der Stelle, an der die entsprechende Maustaste (i. A. rechts) betätigt wird.

Hierfür existiert die Klasse `JPopupMenu`, die sich im Prinzip wie ein normales `JMenu` verhält. Allerdings ist der Anwender selbst dafür verantwortlich,

- *dass* das Popup-Menu überhaupt erscheint,
- *wo* das Popup-Menu erscheint.

Man muss also einen entsprechenden `MouseListener` selbst definieren!
Hierzu kann man die Klasse `MouseAdapter` verwenden.

Beispiel

```
public class MyApp extends JFrame {
    public MyApp()  {
        ...
        inputField = new JTextField(30);
        add(inputField,BorderLayout.CENTER);

        JPopupMenu popup = new JPopupMenu();
        JMenuItem item = new JMenuItem("Löschen");
        item.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                inputField.setText("");
            }
        });
        popup.add(item);
        inputField.addMouseListener(new PopupListener(popup));
        ...
    }
```


Beispiel (Forts.)

```
    class PopupListener extends MouseAdapter {
        JPopupMenu popup;

        public PopupListener(JPopupMenu popup) {
            this.popup = popup;
        }

        @Override
        public void mouseReleased(MouseEvent e) {
            if (e.isPopupTrigger()) {
                popup.show(e.getComponent(), e.getX(), e.getY());
            }
        }
    }
}
```

Die Klasse Action

Häufig soll ein und dieselbe Aktion auf verschiedene Arten und Weisen ausgelöst werden, z. B das Öffnen einer Datei durch

- Auswahl aus dem Datei-Menü
- Klick auf das Ordner-Symbol im Toolbar
- Eingabe von `Ctrl-O`

Zur Vereinfachung gibt es die Klasse `AbstractAction`, die das Interface `ActionListener` implementiert. Dieses kann an verschiedenen Stellen (Buttons, Menüeinträgen) verwendet werden und stellt dort überall dieselbe Funktionalität zur Verfügung.

Eigenschaften von AbstractAction

In der Klasse`AbstractAction` können verschiedene Eigenschaften mit der Methode `putValue(String,Object)` gesetzt werden:

`NAME`	Name (benutzt für Menüeinträge, Buttons, Checkboxen, . . .)
`SMALL_ICON`	Icon (für Menüeinträge, Buttons, Checkboxen, . . .)
`LARGE_ICON_KEY`	Icon (für Buttons, Checkboxen, . . .)
`ACCELERATOR_KEY`	Shortcut für Menüeinträge
`MNEMONIC_KEY`	Mnemonic (für Buttons, Checkboxen, . . .)
`SHORT_DESCRIPTION`	Tooltip-Text
`ACTION_COMMAND_KEY`	String für ActionCommand

Beispiel

```
public class MyApp extends JFrame {
    public MyApp()  {
        ...
        OpenAction openAction = new OpenAction();

        menu = new JMenu("Datei");
        menu.setMnemonic(KeyEvent.VK_D);
        item = new JMenuItem(openAction);
        menu.add(item);
        menuBar.add(menu);
        setJMenuBar(menuBar);
```

Beispiel (Forts.)

```
        JToolBar toolbar = new JToolBar();
        JButton button;
        toolbar.setFloatable(false);
        button = new JButton(openAction);
        button.setText(null);
        toolbar.add(button);
        add(toolbar,BorderLayout.PAGE_START);
        ...
    }
```


Beispiel (Forts.)

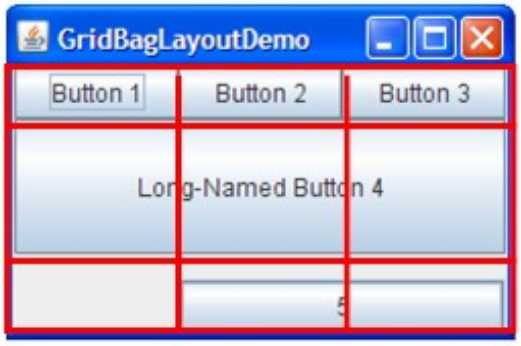
```
class OpenAction extends AbstractAction {
    public OpenAction() {
        putValue(Action.NAME,"Öffnen");
        putValue(Action.MNEMONIC_KEY,KeyEvent.VK_F);
        putValue(Action.ACCELERATOR_KEY,KeyStroke.getKeyStroke(
              KeyEvent.VK_O,KeyEvent.CTRL_MASK));
        putValue(Action.ACTION_COMMAND_KEY,"Öffnen");
        putValue(Action.LARGE_ICON_KEY,new ImageIcon(
              MyApp.class.getResource("/images/Open16.gif")));
        putValue(Action.SHORT_DESCRIPTION,"Datei öffnen");
    }

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


GridBagLayout – Einführung

In vielen Anwendungen soll die Anordnung der Komponenten gitterartig erfolgen. Beim bisher benutzen `GridLayout` sind allerdings alle Zellen des Grids gleich groß, was zu einem unausgewogenen Layout führt. Hier ist das `GridBagLayout` überlegen.
Das `GridBagLayout` hat folgende Eigenschaften:

- Komponenten können mehrere Zeilen oder Spalten einnehmen.
- Es kann definiert werden, in welche Richtungen eine Komponente wachsen kann.
- Die Ausrichtung von Komponenten innerhalb der Zelle kann definiert werden.
- Zellen kann ein Gewicht gegeben werden, das definiert, ob (und wie stark) die Zelle wachsen kann.

Beispiel eines GridBagLayouts



Die Klasse `GridBagConstraints`

Die zuvor beschriebenen Eigenschaften werden in einem `GridBagConstraints`-Objekt beschrieben. Dieses enthält die Attribute

gridx, gridy Position der Komponente (Default: 0). Neben positiven Zahlen ist auch `GridBagConstraints.RELATIVE` zulässig.

gridwidth, gridheight gibt die Zahl der Spalten bzw. Zeilen an, die die Komponente einnimmt (Default: 1). `GridBagConstraints.REMAINDER` zeigt an, dass diese Komponente die letzte in der Zeile bzw. Spalte ist und alle übrigen Zellen einnimmt.

fill gibt an, ob die Komponente gedehnt wird. Zulässige Werte: `GridBagConstraints.NONE` (default), `.BOTH`, `.HORIZONTAL`, `.VERTICAL`

Die Klasse `GridBagConstraints` (Forts.)

ipadx, ipady Padding in Pixeln *innerhalb* der Komponente

anchor Anordnung der Komponente innerhalb der Zelle (wie in `BorderLayout`)

weightx, weighty gibt an, wie stark die Zelle wachsen darf. Erlaubte Werte liegen zwischen 0 (gar nicht, default) und 1 (maximales Wachstum). Haben mehrere Komponenten innerhalb einer Zeile bzw. Spalte Gewichte > 0, so wird freier Platz auf die Komponenten anhand des Gewichtsverhältnisses verteilt.

Beispiel

```
public class GridBagLayoutDemo extends JFrame {
    public GridBagLayoutDemo() {
        super("GidBagLyaoutDemo");
        setLayout(new GridBagLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button;
        GridBagConstraints c;

        button = new JButton("Button 1");
        c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = 0;
        c.gridy = 0;
        c.weightx = 1;
        add(button, c);
```


Beispiel

```
        button = new JButton("Button 2");
        c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = GridBagConstraints.RELATIVE;
        c.gridy = 0;
        c.weightx = 0.5;
        add(button, c);

        button = new JButton("Button 3");
        c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        c.gridx = GridBagConstraints.RELATIVE;
        c.gridy = 0;
        add(button, c);
```


Beispiel

```
        button = new JButton("Long-Named Button 4");
        c = new GridBagConstraints();
        c.fill = GridBagConstraints.BOTH;
        c.ipady = 40;      //make this component tall
        c.weighty = 0.5;
        c.gridwidth = 3;
        c.gridx = 0;
        c.gridy = GridBagConstraints.RELATIVE;
        add(button, c);
```

Beispiel

```
            button = new JButton("5");
            c = new GridBagConstraints();
            c.fill = GridBagConstraints.HORIZONTAL;
            c.ipady = 0;
            c.weighty = 1.0;
            c.anchor = GridBagConstraints.PAGE_END;
            c.gridx = 1;       //aligned with button 2
            c.gridwidth = 2;   //2 columns wide
            c.gridy = GridBagConstraints.RELATIVE; //third row
            add(button, c);
            pack();
            setVisible(true);
        }
        public static void main(String[] args) {
            GridBagLayoutDemo app = new GridBagLayoutDemo();
        }
    }
```