

## In diesem Kursteil

- Kursübersicht
- Modul 1 : Die Entwicklung von Java GUIs
  - Modul Einleitung
  - Lektion 1 - Java GUI Grundlagen
  - Lektion 2 - Layout Manager
  - Lektion 3 - Kreieren komplexer Layouts
  - Kreieren eines Taschenrechners
  - Praktische Übung
  - Quiz
  - Zusammenfassung
- Modul 2 : Das AWT Event Modell
  - Modul Einleitung
  - Lektion 1 - Event Grundlagen
  - Lektion 2 - JDK 1.0 versus neues Event Modell
  - Lektion 3 - Java GUI Verhalten
  - Taschenrechner mit Event Handler
  - Quiz
  - Zusammenfassung
- Modul 3 : Die AWT Komponenten Bibliothek
  - Modul Einleitung
  - Lektion 1 - AWT Schlüsselkomponenten
  - Lektion 2 - Kontrolle des Aussehens der Komponenten und Drucken
  - Kreieren eines GUIs für ein Zeichenprogramm
  - Quiz
  - Zusammenfassung
- Kurs Zusammenfassung

## *Java GUI und Applet Grundlagen*

### **1.1. Kursübersicht**

Wir möchten Ihnen einige weitere Aspekte von Java und des JDK's näher bringen. Sie haben bereits Grundkenntnisse der Java Programmiersprache. Nun werden Sie Komponenten und Layout Managers kennen lernen, mit deren Hilfe Sie Java Graphical User Interfaces (GUIs) entwickeln können. Zusätzlich lernen Sie das Abstract Windowing Toolkit (AWT) und sein Event Modell und die Komponenten Bibliothek kennen. Damit sind Sie in der Lage, einfache GUIs zu entwickeln.

#### **1.1.1.1. Lernziele**

Nach dem Durcharbeiten dieses Kurses sollten Sie in der Lage sein,

- das AWT Package und seine Komponenten zu beschreiben.
- Frame und Panel Container einzusetzen.
- Panels in anderen Containern zu plazieren, um komplexe Layouts aufzubauen.
- Programme zu schreiben, welche Events aus der Benutzeroberfläche abfangen.
- die passenden Interfaces und Handler Methoden für unterschiedliche Eventtypen einzusetzen.
- mit AWT Komponenten zum Bau von Benutzeroberflächen für praktische Anwendungen zu entwickeln.
- Fonts und Farben der AWT Komponenten passend einzusetzen.

## 1.2. *Modul 1 : Die Entwicklung von Java GUIs*

### **In diesem Modul**

- Modul 1 : Die Entwicklung von Java GUIs
  - Modul Einleitung
  - Lektion 1 - Java GUI Grundlagen
  - Lektion 2 - Layout Manager
  - Lektion 3 - Kreieren komplexer Layouts
  - Kreieren eines Taschenrechners
  - Praktische Übung
  - Quiz
  - Zusammenfassung

### 1.2.1. Modul Einleitung

In diesem Modul lernen Sie, wie man in Java ein GUI programmiert. Wir werden uns mit AWT beschäftigen; Swing wird in einem separaten Kurs behandelt.

Sie werden Beispiele von Containern, Komponenten und Layout Managern kennen lernen, sowie grafische Benutzeroberflächen (GUIs). Damit Sie die unterschiedlichen Layout Manager besser verstehen lernen, werden wir Beispiele mit diesen unterschiedlichen Layout Managern entwickeln.

#### 1.2.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

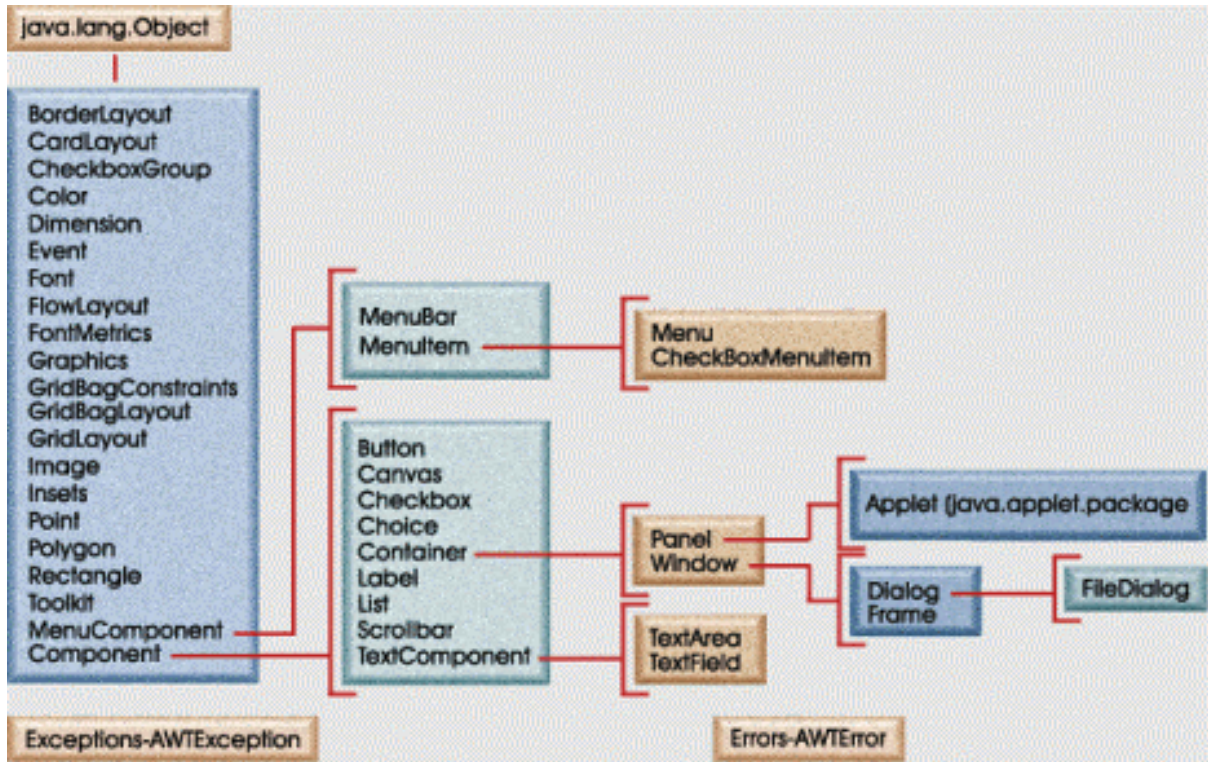
- das AWT Package und seine Komponenten zu beschreiben.
- Container, Komponenten und Layout Manager für die Entwicklung von GUIs einzusetzen.
- die Flow, Border, Grid und Card Layout Managers einzusetzen, um passende dynamische Layouts zu entwickeln.
- Frame und Panel Container einzusetzen.
- Panels innerhalb Containern einzusetzen, um komplexe Layouts aufzubauen.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.2.2. Lektion 1 - Java GUI Grundlagen

### 1.2.2.1. Das java.awt Package

Das java.awt Package enthält Klassen, mit deren Hilfe GUI Komponenten generiert werden können. Die Hierarchie dieses Package sehen Sie schematisch in der Grafik weiter unten.



java.lang.Object

MenuComponent

MenuItem

Menu

CheckboxMenuItem

Component

MenuBar  
Button  
Canvas  
Checkbox  
Choice  
Container

Panel  
Window

Applet  
Dialog  
Frame

FileDialog

Label  
List  
Scrollbar  
TextComponent

TextArea  
TextField

Zuerst eine kurze Erklärung der Klassen. Anschliessend folgen die Details. In diesem Modul werden wir die folgenden Klassen besprechen:

## 1.2.2.2. Applet

Diese Klasse implementiert genau das, was der Name besagt, ein Applet. Um eigene Applets zu kreieren, sollten Sie Unterklassen dieser Klasse bilden. Ein Applet wird in AWT als ein spezialisiertes Panel verstanden, welches in einen speziellen Container, den *Applet Context* eingebettet ist. Ein Applet Context kann ein Web Browser oder eine andere Applikation sein, mit der Applets dargestellt werden können. Ein Applet implementiert eine ganze Menge Methoden, welche der Applet Context verwendet, um das Applet zu initialisieren, zu stoppen, zu zerstören.

## 1.2.2.3. Panel Window

Diese Klasse ist ein `Container`, der selber in einem weiteren Container enthalten ist. Im Gegensatz zu `Frame` und `Dialog` ist `Panel` ein Container, welcher kein eigenes separates Fenster kreiert. Ein `Panel` kann eingesetzt werden, um Teile eines grösseren Interfaces aufzunehmen, als Teil eines `Frame` oder `Dialog` oder innerhalb eines anderen `Panel`.

`Applet` ist auch eine Unterklasse von `Panel`, womit klar ist, dass Applets in einem `Panel` angezeigt werden, welches in einem Web Browser oder Applet Viewer enthalten ist. Der Standard `LayoutManager` für ein `Panel` ist `FlowLayout`.

## 1.2.2.4. Dialog Frame

Diese Klasse repräsentiert ein optional grössenmässig anpassbares top-level Applikations-Fenster, mit Titelzeile und anderen plattformabhängigen Windows Dekorationen.

- `setTitle()` spezifiziert einen Titel,
- `setMenuBar()` spezifiziert eine Menüschiene,
- `setCursor()` spezifiziert einen Zeiger und
- `setIconImage` spezifiziert ein Icon für das Fenster.

Mit Hilfe der Methode `pack()` der Klasse `Window` werden alle Komponenten grössenmässig angepasst und ein initialer Layout festgelegt.

Mit Hilfe der Methode `show()` der Klasse `Window` wird ein Frame auf dem Bildschirm angezeigt, ganz zu oberst im Window Stack.

Die Methode `hide()` entfernt das Fenster vom sichtbaren Bildschirm.

Die Methode `dispose()` räumt auf, falls ein `Frame` nicht länger benötigt wird. Damit werden Window Systemressourcen freigegeben.

## 1.2.2.5. TextArea, TextField

Diese Unterklasse von `Component` zeigt eine Zeile Text an, welche optional auch editierbar sein kann. Die meisten der interessanten Methoden dieser Klasse sind in der Oberklasse `TextComponent` zusammengefasst.

Mit `setEchoCharacter()` können Sie beispielsweise ein Zeichen angeben, welches bei sensitiven Eingaben als Echo ausgegeben werden soll, beispielsweise bei der Eingabe eines Passworts.

## 1.2.2.6. Window

Diese Klasse repräsentiert ein top-level Window ohne Rahmen und Menü. `Window` ist ein `Container` mit `BorderLayout` als Standard Layout Manager. `Window` wird selten direkt gebraucht. In der Regel werden die Unterklassen `Frame` und `Dialog` eingesetzt, da sie nützlicher sind.

## 1.2.2.7. Panel

Diese Klasse ist ein `Container`, der selber wieder in einem `Container` enthalten ist. Im Gegensatz zu `Frame` und `Dialog` ist `Panel` ein `Container`, welcher kein separates Fenster kreiert.

`Panel` ist immer dann sinnvoll einsetzbar, wenn Elemente einer grösseren grafischen Oberfläche zusammengehalten werden müssen, innerhalb eines `Frame` oder `Dialog` oder eines anderen `Panel`.

`Applet` ist eine Unterklasse von `Panel` und kann somit in einem `Panel` innerhalb eines Web Browsers oder `Applet Viewers` angezeigt werden.

Der Standard `LayoutManager` für ein `Panel` ist `FlowLayout`.

## 1.2.2.8. Button

Diese Klasse repräsentiert einen GUI `PushButton`, welcher ein spezifiziertes `TextLabel` anzeigt. Mit `setActionCommand()` kann eine Zeichenkette an das `ActionEvent` (welches vom `Button` generiert wird) weitergegeben werden.

## 1.2.2.9. Checkbox

Diese Klasse repräsentiert eine GUI `Checkbox` mit `TextLabel`. Die `Checkbox` verwaltet einen `boolean` Zustand - ob sie gewählt ist oder nicht. Die `Checkbox` kann optional Teil einer `CheckboxGroup` sein, wodurch ein "Radio Button" Verhalten erreicht werden kann.

## 1.2.2.10. CheckBoxGroup

Ein `CheckboxGroup` Objekt impliziert gegenseitigen Ausschluss ("Radio Button" Verhalten) unter mehreren `Checkbox` Buttons. Eine `Checkbox` Komponente kann ein `CheckboxGroup` Objekt angeben (im Konstruktor oder durch `Checkbox.setCheckboxGroup()`, falls `Checkbox` zu einem `CheckboxGroup` Objekt gehört, wird die frühere Zugehörigkeit zu einer `Checkbox` überschrieben).

## 1.2.2.11. Choice

Diese Klasse repräsentiert ein "Option Menu" oder eine "DropDown Liste."

Die `addItem()` Method gestattet es, ein Item mit spezifiziertem Label einem `Choice` Menu hinzuzufügen.

`getSelectedIndex()` liefert die numerische Position des ausgewählten Items im Menü und `getSelectedItem()` liefert das Label des selektierten Items.

## 1.2.2.12. Container

Diese Klasse implementiert eine Komponente, welche andere Komponenten enthält. Sie können `Container` nicht direkt instanzieren. Sie müssen eine der Subklassen, beispielsweise `Panel`, `Frame` oder `Dialog` verwenden.

Nachdem ein `Container` kreiert wurde, kann sein `LayoutManager` mit `setLayout()` gesetzt werden, mit `add()` weitere Komponenten hinzugefügt werden oder mit `remove()` auch wieder entfernt werden.

`getComponents()` liefert ein Array von Komponenten im `Container`.

`locate()` bestimmt, innerhalb welche der enthaltenen Komponenten ein bestimmter Punkt fällt.

`list()` liefert Debugging Ausgaben.

## 1.2.2.13. TextComponent

Die `TextComponent` ist eine Komponente, welche editierbaren Text anzeigt. Der Text kannmodifiziert werden, entweder durch den Benutzer oder durch ein Programm.

Es gibt zwei `TextComponent` im `AWT Package`: Textfelder und Textflächen. Beide sind Unterklassen von `TextComponent`.

Die `TextComponent` Klasse selbst wird nicht instanziiert. Die Klasse dient im Wesentlichen der Kapselung gemeinsamer Funktionalitäten für die Textfeld und die Textflächen Textkomponenten.

## 1.2.2.14. Component

`Component` ist die Oberklasse aller GUI Komponenten (ausser den Menükomponenten) im `java.awt Package`. `Component` kann nicht direkt instanziiert werden; Sie müssen eine Unterklasse verwenden.

## 1.2.2.15. GridLayout

Diese Klasse implementiert das `LayoutManager Interface`, um `Component` Objekte in einem `Container` anzuordnen. Er teilt den `Container` in eine bestimmte Anzahl Zeilen und Spalten und arrangiert die Komponenten in diesen, von links nach rechts, von oben nach unten.

Falls eine der Anzahlen (Kolonnen, Spalten) null ist, wird dieser Wert auf Grund der Dimensionen und der Komponenten berechnet.

Sie sollten diesen Layout nicht mit dem flexibleren aber komplexeren `GridBagLayout` verwechseln.

## 1.2.2.16. FlowLayout

Die Klasse implementiert das `LayoutManager Interface`, um Komponenten in einem `Container` anzuordnen. Der `FlowLayout` funktioniert analog zur Anordnung der Worte in einem Dokument: von links nach rechts, oben nach unten.

Die Anzahl Elemente pro Zeile ergibt sich einfach aus den Dimensionen der Komponenten. Dem Konstruktor können Sie beispielsweise den Abstand der einzelnen Komponenten (horizontal, vertikal) übergeben.

## 1.2.2.17. BorderLayout

Diese Klasse implementiert das `LayoutManager` Interface, um `Component` Objekte in einem `Container` anzuordnen. Der `BorderLayout` arrangiert die Komponenten, die im `Container` sind (mittels `Container.add()` in den `Container` gestellt wurden) mittels Namen:

"North", "South", "East", "West" und "Center." Die Anordnung geschieht am Rand und im Zentrum des Containers.

Die `hgap` und `vgap` Argumente des `BorderLayout` Konstruktors spezifizieren die gewünschten horizontalen und vertikalen Abstände benachbarter Abstände.

## 1.2.2.18. Event

Events werden zwischen den AWT Komponenten ausgetauscht, als Objekte der `Event` Klasse. Falls ein Event, wie beispielsweise ein Mausklick geschieht, wird ein plattformunabhängiges `Event` Objekt kreiert. Dieses enthält Informationen über das Objekt, das Event. Alle Events benutzen die `Event` Klasse und verfügen somit über die gleichen Attribute. Aber je nach Event werden nicht alle Attribute eingesetzt. Welche Attribute konkret verwendet werden, wird vom Eventtypus bestimmt.

## 1.2.2.19. Exception

Diese Klasse ist die Wurzel für alle Exceptions in Java. Eine `Exception` signalisiert eine abnormale Bedingung, welche speziell behandelt werden muss, um einen Programmabbruch zu vermeiden. Alle Exceptions, welche nicht `RuntimeException` sind, müssen mit der `throws` Anweisung bei der Methodendefinition deklariert werden.

## 1.2.2.20. Error

Diese Klasse ist die Wurzel der Error Hierarchie in Java. Unterklassen von `Error` werden nicht in `throws` Anweisungen der Methodendefinition deklariert. Sie führen in der Regel zu einem Programmabbruch..

## 1.2.3. Container und Komponenten

Die grundlegenden Komponenten von Java AWT sind `components` und `containers`. `Components` sind generell sichtbare Aspekte eines GUI, beispielsweise ein `Button` oder ein `Label`. `Components` werden einem `Display` hinzugefügt, indem man Sie einem `container` hinzufügt.

Ein `container` kann eine oder mehrere `components` enthalten und kann falls erwünscht selber andere `containers` enthalten. Die Tatsache, dass ein `container` nicht nur `components` sondern auch noch `containers` enthalten kann, ist fundamental für den Aufbau von Layouts mit realistischer Komplexität.

### 1.2.3.1. Positionieren von Komponenten

Die Positionierung einer `component` in einem `container` wird durch den `Layout Manager` bestimmt. Ein `container` enthält eine Referenz auf ein bestimmtes `LayoutManager` Objekt. Immer wenn der `container` eine `component` positionieren muss, wird dafür der `LayoutManager` beauftragt. Das Gleiche passiert bei der Delegation des Entscheides bezüglich der Grösse einer `component`.

Weil der `LayoutManager` für die Grösse und Position der `components` im `container` verantwortlich ist, sollten Sie nicht versuchen, Grösse und Position der `components` selbst festzulegen. Falls Sie dies versuchen, beispielsweise mit den Methoden `setLocation()`, `setSize()` oder `setBounds()`, überschreibt der `LayoutManager` Ihre Entscheide.

Falls Sie Grösse und Position der `components` mit einem vorgegebenen `Layout Manager` nicht passend festlegen können, haben Sie die Möglichkeit den `LayoutManager` auszuschalten. Dies geschieht mittels der folgenden `container` Methode:

```
setLayout (null);
```

Danach *müssen* Sie `setLocation()`, `setSize()` oder `setBounds()` für die `components` setzen, um sie im `Container` zu lokalisieren.

Damit geraten Sie allerdings in eine plattformabhängige Situation, mit unterschiedlichen Fonts und Windowingsystemen. Besser ist es immer, wenn Sie einen neuen `LayoutManager` definieren.



## 1.2.4. Frames

Ein `Frame` ist eine Unterklasse von `Window`. Es ist ein `Window` mit Titel und veränderlicher Grösse. Der Konstruktor `Frame(String)` der `Frame` Klasse kreiert ein neues, unsichtbares `Frame` Objekt mit `String` als Titel. Ein `Frame` können Sie von der Grösse her mit der `setSize()` Methode verändern. Diese Methode wird aus der `Component` Klasse geerbt.

Die `setVisible()` und `setSize()` Methoden machen das `Frame` sichtbar. Als Alternative zu `setSize()` können Sie `pack()` verwenden. Damit wird einfach der verfügbare Raum aufgefüllt. Das folgende einfache Beispiel kreiert ein einfaches `Frame` mit einem bestimmten Titel, einer bestimmten Grösse und einer bestimmten Hintergrundfarbe.

```
package einframe;

import java.awt.*;
import java.applet.*;

public class MeinFrame extends Applet {
    private Frame fr;
    public void start ( ) {
        fr = new Frame("Hallo da draussen!");
        fr.setSize(500,500);
        fr.setBackground(Color.blue);
        fr.setVisible(true);
    }
    // ...
}
```

**Bemerkung** - Sie können `Frame` auch in einem `main` und separaten Klassen einsetzen. Das obige Fragment zeigt lediglich schematisch, wie ein Programm aussehen könnte.



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.2.5. Panels

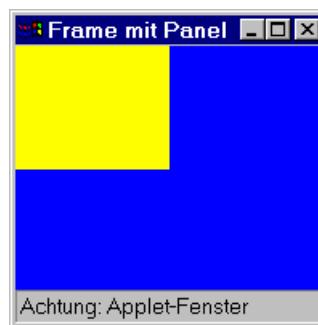
Panels, wie Frames, stellen Raum zur Verfügung, in den Sie GUI Komponenten oder andere Panels plazieren können. Panels werden mit dem Konstruktor `Panel()` kreiert.

Ein einmal kreiertes `Panel` Objekt muss einem `Window` oder `Frame` Objekt hinzugefügt werden, damit es sichtbar werden kann.

Dies erreichen Sie mit der `add()` Methode der `Container` Klasse. Unten sehen Sie ein einfaches Programmfragment, welches ein kleines **gelbes Panel** kreiert und ein `Frame` Objekt hinzufügt.

```
package einframemitpanel;

import java.awt.*;
import java.applet.*;
public class FrameMitPanel extends Applet {
    private Frame fr;
    public void start() {
        fr = new Frame("Frame mit Panel");
        Panel pan = new Panel();
        fr.setSize(200,200);
        fr.setBackground(Color.blue);
        fr.setLayout(null);
        pan.setSize(100,100);
        pan.setBackground(Color.yellow);
        fr.add(pan);
        fr.setVisible(true);
    }
    //...
}
```



Das Panel ist gelb;  
das Frame hat den blauen  
Hintergrund.

## 1.2.6. Container Layouts

Das Layout der Komponenten in einem `container` kann man mit Hilfe eines Layout Manager festlegen. Jeder Container, wie beispielsweise ein `panel` oder ein `frame`, besitzt einen *Standard Layout Manager* (der ihm zugeordnet ist). Sie können das Layout beim Kreieren des Containers ändern und festlegen.

Einige Layout Manager sind standardmässig in Java vorhanden:

- `FlowLayout` - diese Klasse positioniert die Komponenten in einem Container wie die Worte auf einer Seite - von links nach rechts und von oben nach unten. Immer wenn eine Zeile voll ist, wird eine neue Zeile angefangen.
- `BorderLayout` - der Standard Layout Manager für `Windows`, `Dialogs` und `Frames`. `BorderLayout` arrangiert Komponenten, welche dem `Container` hinzugefügt werden, an Positionen, welche als "North", "South", "East", "West" und "Center" bezeichnet werden. Eine "Center" bezeichnete Komponente wird ins Zentrum des Containers gestellt.
- `GridLayout` - Die `GridLayout` unterteilt einen `Container` in eine zu spezifizierende Anzahl Zeilen und Spalten und arrangiert die Komponenten in diesen, von links nach rechts und oben nach unten.
- `CardLayout` - Die `CardLayout` Klasse stellt alle seine Komponente so gross dar, wie den Container, sorgt aber dafür, dass jeweils nur eine Komponente sichtbar ist.
- `GridBagLayout` - Die `GridBagLayout` Klasse repräsentiert die komplexeste und mächtigste der `LayoutManager` Klassen im `java.awt` Package. Der Container wird eingeteilt in ein Raster mit Zeilen und Spalten. Jede Komponente wird in eine dieser Zellen eingefügt. Dabei wird die Grösse der Zelle der Grösse der Komponente angepasst. Es wird sichergestellt, dass die einzelnen Komponenten jeweils nicht überlappen.

Den `GridBagLayout` Manager werden wir nicht genauer besprechen. Allerdings können Sie im Design Modus des JBuilders weitere Borland spezifische Layout Manager kennen lernen und mit dem Designer einfache und ineffiziente Oberflächen entwickeln, zum Prototyping!

## 1.2.6.1. Ein einfaches Layout Manager Beispiel

Das folgende GUI Beispiel verwendet viele Methoden, die wir im Folgenden detaillierter besprechen werden. Das Beispiel dient einfach der Illustration, damit man sich an etwas festhalten kann.

```
1  import java.awt.*;
2  import java.applet.*;
2  public class BeispielGUI extends Applet {
3      private Frame f;
4      private Button b1;
5      private Button b2;
6      public void start() {
```

Die `start()` Methode startet das Applet, das Applet wird lebendig, "live." Dies geschieht dann, wenn das Applet das erste mal gestartet wird, nachdem die `init()` Method beendet wurde.

Die Methode wird auch ausgeführt,

- wenn beispielsweise der Brower ikonisiert wurde und wieder vergrößert wird;
- oder wenn der Browser wieder auf die selbe Seite (mit dem Applet) zurückkehrt, nachdem eine andere URL besucht wurde.

Die `start()` Methode (in diesem Beispiel) muss zweierlei tun:

- erstens - eine Instanz der Klasse `BeispielGUI` kreieren.  
Solange keine Instanz existiert, gibt es auch kein Frame und keine Buttons.
- zweitens - sobald das Applet gestartet wird, ruft `start()` die Instanzenmethode `go()`.  
In `go()` geschieht alles!

```
7      GUIBeispiel guiWindow = new GUIBeispiel();
8      guiWindow.go();
9  }
10     public void go() {
11         f = new Frame("GUI Beispiel");
```

Diese Methode kreiert eine Instanz der Klasse `java.awt.Frame`. Ein `Frame` in Java ist ein top-level Window, mit Titelzeile - im Konstruktor als Argument "GUI Beispiel" - und der Möglichkeit die Fenstergrösse zu verändern und je nach Plattform weiteren Möglichkeiten. Das kreierte Frame besitzt die Grösse null (!) ist nicht sichtbar!

```
12         f.setLayout(new FlowLayout() );
```

Damit wird eine Instanz des Flow Layout Managers kreiert und ins Frame eingefügt, dem Frame gegenüber als Layout Manager bekanntgegeben. Für jedes Farme gibt es einen Standard Layout Manager. Aber in diesem Beispiel möchten wir diesen nicht verwenden. Der Flow Layout Manager ist der einfachste Layout Manager in AWT. Er positioniert die Komponenten wie Worte auf einer Seite: von links nach rechts, von oben nach unten, Zeile für Zeile. Die Komponenten werden automatisch zentriert.

```
13         b1 = new Button("Klick");
```

Nun kreieren wir noch einen Knopf, einen `java.awt.Button`. Das Label auf dem Knopf wird einfach als Text an den Konstruktor übergeben.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
14         b2 = new Button("Nicht Klicken!");
15         f.add(b1)
```

Damit enthält das frame `f` (ein Container) die Komponente `b1`. Buttons sind Beispiele für Java Komponenten. Die Positionierung des Knopfes geschieht mittels Layout Manager..

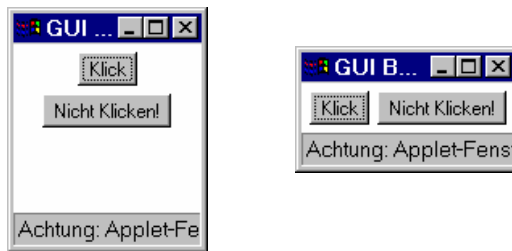
```
16         f.add(b2);
17         f.pack();
```

Packen bewirkt, dass die im Frame enthaltenen Komponenten "nett" zusammengefasst werden. Der Layout Manager muss angefragt werden, damit diese Methode ihre Aufgaben erledigen kann.

```
18         f.setVisible(true);
```

Nun zeigen wir das Frame dem Benutzer.

```
19     }
20 }
```



Falls genügende Platz da ist, wird das Layout einfach vertikal oder horizontal angepasst. Die Beispiele finden Sie wie üblich auf dem Server / dem Web / der CD.

## 1.2.7. Lektion 2 - Layout Manager

### 1.2.7.1. Flow Layout Manager

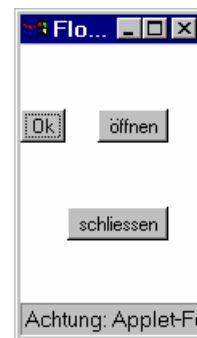
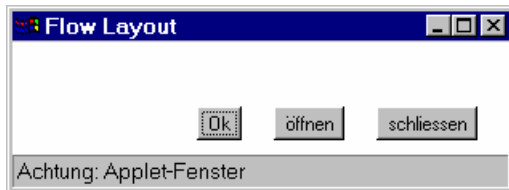
Der `FlowLayout` Manager positioniert die Komponenten zeilenweise. Jedesmal wenn eine Zeile voll ist, wird eine neue Zeile angefangen. Im Gegensatz zu anderen Layout Managern schreibt der `FlowLayout` Manager die Grösse der Komponenten nicht vor; Sie können also die Grösse frei vorgeben, sind aber auch für die passende Grösse verantwortlich.

Alle Komponenten besitzen eine Methode `getPreferredSize()`, welche von den Layout Managern eingesetzt wird, um abzufragen, welche Dimensionen einer Komponente gegeben werden soll.

Sie können auch an Stelle einer zentrierten Ausrichtung die Anordnung linksbündig oder rechtsbündig ausrichten.

Auch die Zwischenräume zwischen den Komponenten können spezifiziert werden, beispielsweise falls Sie einen breiteren Rand möchten.

Falls Sie die Grösse der verfügbaren Fläche verändern, kann der Layout Manager die Komponenten neu anordnen. Hier ein Beispiel (ein weiteres sehen Sie oben):



## 1.2.7.1.1. Flow Layout Manager Beispiel

Die folgenden Programmfragmente zeigen, wie der `FlowLayout` Manager eingesetzt werden kann, mit der `setLayout()` Methode aus der `Component` Klasse:

```
setLayout(new FlowLayout(FlowLayout.RIGHT, 20, 40)); // Ausrichtung+gaps
setLayout(new FlowLayout(FlowLayout.LEFT));
setLayout(new FlowLayout());
```

Der Parameter `FlowLayout.RIGHT` legt fest, dass die Komponenten rechtsbündig angeordnet werden. `FlowLayout.LEFT` entspricht der linksbündigen Anordnung. Die Parameter 20 und 40 geben den horizontalen (20) und vertikalen (40) Rand zwischen den Komponenten an.

Das Beispiel unten entspricht im Wesentlichen dem Beispiel in der JDK Dokumentation unter `FlowLayout` Manager. Das Programm befindet sich auch auf dem Web / Server / der CD. Testen Sie das Programm und verändern Sie die Grösse des Fensters. Sie können dann erkennen, wie der Layout Manager die Komponenten neu anordnet.

```
package flowlayoutmanager;

import java.awt.*;
import java.applet.*;

/**
 * Title:
 * Description:
 * @version 1.0
 */

public class FlowLayoutBeispiel extends Applet{
    private Frame f;
    private Button button1, button2, button3;

    public void init() {
        f = new Frame("Flow Layout");
        f.setLayout(new FlowLayout(FlowLayout.RIGHT, 20, 40));
        button1 = new Button("Ok");
        button2 = new Button("Öffnen");
        button3 = new Button("schliessen");
        f.add(button1);
        f.add(button2);
        f.add(button3);
        f.setSize (100,100);
    }

    public void start () {
        f.setVisible(true);
    }

    public void stop () {
        f.setVisible(false);
    }
}
```

## 1.2.7.2. BorderLayout Manager

Der BorderLayout Manager verwendet ein komplexeres Schema für die Platzierung der Komponenten auf den Panels oder in einem Window. Der BorderLayout definiert fünf feste Bereiche: North, South, East, West und Center.

North entspricht der Fläche oben im Panel;  
East beschreibt die Fläche rechtes;  
und so weiter...

Center beschreibt die verbleibende Fläche, nachdem North, South, East und West ausgefüllt wurden.

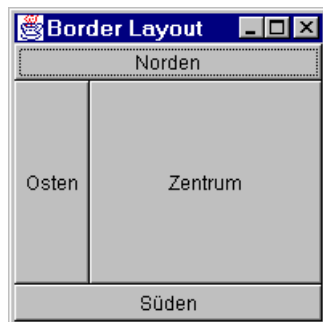
Der BorderLayout Manager ist der Standard Layout Manager für Dialogs und Frames. Wie Sie unten sehen können, verändert sich die relative Position beim Verändern des Fensters nicht. Aber die Grösse der Knöpfe passt sich jeweils der Fenstergrösse an.



Falls Sie eine Komponente nicht einer der Regionen zuordnen, sind sie nicht sichtbar. Die Regionen werden einfach mit Strings "North", "South", "East", "West" und "Center".

Die Schreibweise (erstes Zeichen gross, der Rest klein) muss genau so aussehen, wie oben! Die Bereiche sind auch in Konstanten festgehalten: `BorderLayout.NORTH`, `BorderLayout.SOUTH`, `BorderLayout.EAST`, `BorderLayout.WEST` und `BorderLayout.CENTER`.

Wie Sie oben sehen, können Sie die einzelnen Bereiche durch Strecken des Fensters verändern. Aber die relative Position bleibt erhalten.



Falls Sie einen Bereich weglassen, wird er auch nicht dargestellt, wie oben erkennbar.

Falls Sie einem Bereich mehrere Komponenten zuordnen, ist nur eine sichtbar.

Im nebenstehenden Beispiel wurden zwei Komponenten auf West gesetzt. Die erste der gesetzten Komponenten (mit dem Label "Westen") ist nicht sichtbar. Die später platzierte Komponente ("Osten") überschreibt die erste Komponente.

Wie man mehrere Komponenten in einen Bereich platzieren können, sehen wir später.



## 1.2.7.2.1. BorderLayout Manager Beispiel

Der folgende Programmcode wurde verwendet, um die oben wiedergegebenen Screen Snapshots zu generieren. Starten Sie das Beispiel und verändern Sie das angezeigte Fenster, um erkennen zu können, wie der `BorderLayout` Manager reagiert. Das Beispiel stammt im Wesentlichen aus der JDK Dokumentation zum Thema `BorderLayout` Manager.

Da der `BorderLayout` Manager der Standard Layout Manager für `Frames` ist, brauchen wir den Layout Manager nicht explizit mit der `setLayout()` Methode zu setzen.

Konstanten wie `BorderLayout.CENTER`, kann man auch in Verbindung mit dem `BorderLayout` Manager einsetzen.

`Runtime Exceptions` werden typischerweise geworfen, falls Sie "oNrth" eintippen.

```
package borderlayoutmanager;

import java.awt.*;
import java.applet.*;

/**
 * Title:
 * Description:
 * @author J.M.Joller
 * @version 1.0
 */

public class BorderLayoutBeispiel extends Applet {
    private Frame f;
    private Button bn, bs, bw, be, bc;

    public void init() {
        f = new Frame("Border Layout");
        bn = new Button("Norden");
        bs = new Button("Süden");
        bw = new Button("Westen");
        //be = new Button("Osten"); // Osten weglassen (Demo)
        be = new Button("Osten"); // sind zwei im Westen sichtbar? (Demo)
        bc = new Button("Zentrum");

        f.add(bn, "North");
        f.add(bs, "South");
        f.add(bw, "West");
        f.add(be, "West");
        f.add(bc, "Center");

        f.setSize (200, 200);
    }

    public void start () {
        f.setVisible(true);
    }

    public void stop () {
        f.setVisible(false);
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.2.7.3. Der GridLayout Manager

Der `GridLayout` Manager stellt recht flexible Layoutmöglichkeiten zur Verfügung. Sie definieren beim Kreieren des Layout Managers wieviele Zeilen und Spalten sie haben möchten. Die Komponenten werden dann in die resultierenden Zellen gestellt.

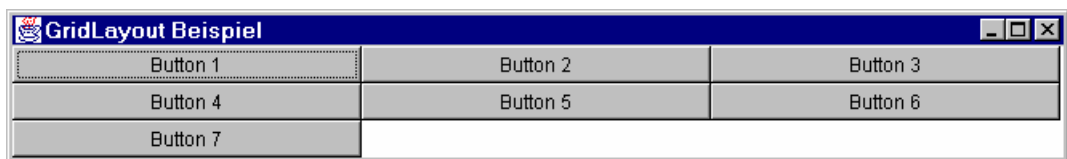
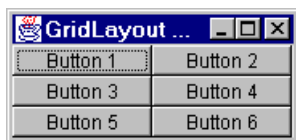
Beispiel:

Sie wählen `GridLayout` mit drei Zeilen und zwei Spalten:

```
new GridLayout (3, 2)
```

Wie beim `BorderLayout` Manager ändert sich die relative Position nicht falls Sie die Grösse der Fenster verändern. Die Grösse der Zellen ist gleich und wird einfach berechnet (Höhe bzw. Breite / Anzahl Komponenten).

Die Zellen werden der Reihe nach aufgefüllt und von links nach rechts dargestellt.



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.2.7.3.1. GridLayout Manager Beispiel

Auch dieses Programm ist recht einfach. Starten Sie das Programm und verändern Sie die Grösse oder die Anzahl Zellen und schauen Sie, wie der `GridLayout` Manager reagiert.

Das Beispiel verwendet die `pack()` Methode, um die einzelnen Zellen optimal anzuordnen.

```
package gridlayoutmanager;

import java.awt.*;
import java.applet.*;

/**
 * Title:
 * @author J.M.Joller
 * @version 1.0
 */

public class GridLayoutBeispiel extends Applet {
    private Frame f;
    private Button b1, b2, b3, b4, b5, b6;

    public void init() {
        f = new Frame("GridLayout Beispiel");

        f.setLayout (new GridLayout (3, 2));
        b1 = new Button("Button 1");
        b2 = new Button("Button 2");
        b3 = new Button("Button 3");
        b4 = new Button("Button 4");
        b5 = new Button("Button 5");
        b6 = new Button("Button 6");

        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.add(b6);

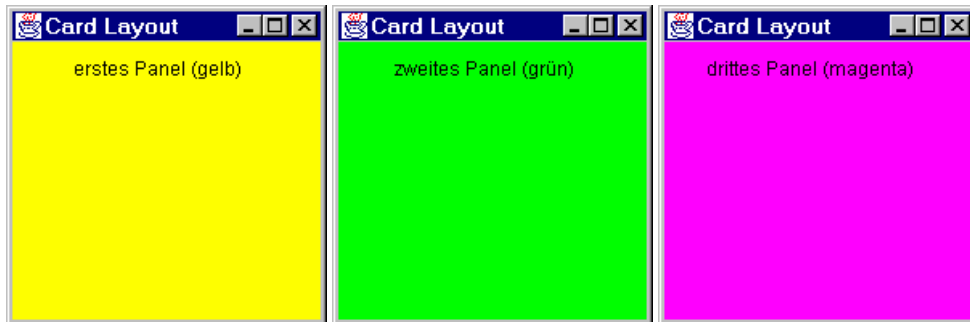
        f.pack();
        f.setVisible(true);
    }

    public void start() {
        f.setVisible(true);
    }

    public void stop() {
        f.setVisible(false);
    }
}
```

## 1.2.7.4. Der CardLayout Manager

Der `CardLayout` Manager simuliert einen Kartenstapel, wobei jeweils nur eine Karte sichtbar ist. Schauen wir uns ein Beispiel an, mit fünf Karten in einem `Frame`. Falls Sie mit der Maus ins `Panel` klicken, zeigt Ihnen `Frame` die nächste Karte:



Der Übergang von einer Karte zur nächsten lässt sich denkbar einfach zu programmieren:

```
public void mousePressed(MouseEvent me) {  
    // jetzt müssen wir jeweils aufs nächste Panel wechseln  
    // falls dieses Ereignis eintrifft  
    meinCardLayout.next(f);  
}
```

Sie sehen aber, dass es grafisch nicht erkennbar ist, wieviele Karten vorhanden sind. Alle Karten werden automatisch nach hinten geschoben, sobald Sie eine bestimmte Karte anwählen.

### 1.2.7.4.1. CardLayout Manager Beispiel

Der Programmcode zum obigen Beispiel ist recht einfach. Spielen Sie mit dem Beispiel und schauen Sie, wie der `CardLayout` Manager reagiert, falls Sie mit der Maus auf die Kartenfläche fahren und eine der Maustasten klicken.

```
package cardlayoutmanager;  
  
import java.awt.*;  
import java.awt.event.*;  
import java.applet.*;  
  
/**  
 * @author J.M.Joller  
 * @version 1.0  
 */  
  
public class CardLayoutBeispiel extends Applet implements MouseListener{  
  
    Panel p1, p2, p3, p4, p5;  
    Label l1, l2, l3, l4, l5;  
  
    CardLayout meinCardLayout;  
    Frame f;  

```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public void init () {
    f = new Frame ("Card Layout");
    meinCardLayout = new CardLayout();
    f.setLayout(meinCardLayout);

    // diese Panels
    // werden die Karten
    p1 = new Panel();
    p2 = new Panel();
    p3 = new Panel();
    p4 = new Panel();
    p5 = new Panel();

    // kreieren eines Labels pro Karte
    // ändern der Farbe pro Panel
    // damit werden sie besser unterscheidbar

    l1 = new Label("erstes Panel (gelb)");
    p1.setBackground(Color.yellow);

    l2 = new Label("zweites Panel (grün)");
    p2.setBackground(Color.green);

    l3 = new Label("drittes Panel (magenta)");
    p3.setBackground(Color.magenta);

    l4 = new Label("viertes Panel (weiss)");
    p4.setBackground(Color.white);

    l5 = new Label("fünftes Panel (cyan)");
    p5.setBackground(Color.cyan);

    // Panel und Label verbinden
    // mouseListener hinzufügen (ohne spezielle Aktion)
    p1.add(l1);
    p1.addMouseListener(this);

    p2.add(l2);
    p2.addMouseListener(this);

    p3.add(l3);
    p3.addMouseListener(this);

    p4.add(l4);
    p4.addMouseListener(this);

    p5.add(l5);
    p5.addMouseListener(this);

    // Panels zum CardLayout hinzufügen
    f.add(p1, "Erstes");
    f.add(p2, "Zweites");
    f.add(p3, "Drittes");
    f.add(p4, "Viertes");
    f.add(p5, "Fünftes");

    // Erstes Panel anzeigen
    meinCardLayout.show(f, "Erstes Panel");
    //meinCardLayout.first (f);

    f.setSize (200, 200);
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public void start () {
    f.setVisible(true);
}

public void stop() {
    f.setVisible(false);
}

public void mousePressed(MouseEvent me) {
    // jetzt müssen wir jeweils aufs nächste Panel wechseln
    // falls dieses Ereignis eintrifft
    meinCardLayout.next(f);
}

public void mouseClicked(MouseEvent me) {}

public void mouseReleased(MouseEvent me) {}

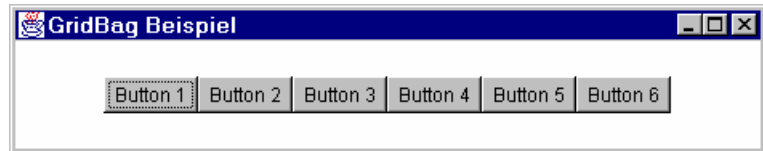
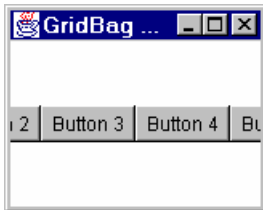
public void mouseEntered(MouseEvent me) {}

public void mouseExited(MouseEvent me) {}
}
```

## 1.2.7.5. GridBagLayout Manager

Der GridBagLayout Manager ist ein komplexerer Layout Manager. Er basiert auf dem GridBagConstraints Manager, gestattet aber den einzelnen Komponenten eine bestimmte Grösse anzunehmen, statt einfach die Zelle auszufüllen.

Sie können eine einzelne Komponente auch auf mehrere Zellen ausdehnen. Der Layout Manager ordnet die einzelnen Zellen so an, dass die jeweils grösste Platz hat.



### 1.2.7.5.1. GridBagLayout Manager Beispiel

Das obige einfache Beispiel zeigt, wie die Ausgabe des GridBagLayout aussieht.

```
package gridbaglayout;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/**
 * @author J.M.Joller
 * @version 1.0
 */
public class GridBagLayoutBeispiel extends Applet {
    private Frame f;
    private Button b1, b2, b3, b4, b5, b6;
    public void init() {
        f = new Frame("GridBag Beispiel");
        f.setLayout (new GridBagLayout());
        f.setSize(200,200);
        b1 = new Button("Button 1");
        b2 = new Button("Button 2");
        b3 = new Button("Button 3");
        b4 = new Button("Button 4");
        b5 = new Button("Button 5");
        b6 = new Button("Button 6");

        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.add(b6);
        f.setVisible(true);
    }
    public void start() {
        f.setVisible(true);
    }
    public void stop() {
        f.setVisible(false);
    }
}
```

## 1.2.8. Lektion 3 - Komplexere Layouts

### 1.2.8.1. Containers - Frames und Panels anders betrachtet

Das AWT stellt mehrere Container zur Verfügung. In dieser Lektion diskutieren wir zwei davon: Frames und Panels.

Frames haben wir bereits gesehen. Sie stellen die "top level" Windows dar, mit Titel, Rahmen und der Möglichkeit der Grössenanpassung. Das Aussehen und Verhalten hängt von der Plattform ab.

Falls Sie keine `setLayout()` Methode einsetzen, wird der jeweilige Standardlayout verwendet, wie wir bereits oben gesehen haben. Die meisten Applikationen verwenden mindestens ein Frame als Plattform für ihr GUI. Aber Sie können auch mehrere Frames gleichzeitig einsetzen.

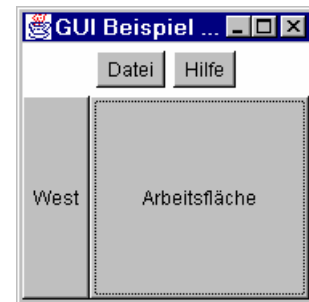
Panels sind Container. Sie besitzen *kein* eigenes Aussehen und können nicht alleine eingesetzt werden. Panels verwenden als Standard den `FlowLayout` Manager. Aber Sie können mittels `setLayout()` einen eigenen Standard definieren.

Panels werden kreiert und mit andern Containern kombiniert, genau wie Komponenten (Buttons, Checkboxes, Labels ...). Aber wenn ein Panel mit einem Container kombiniert wird, haben Sie folgende zusätzliche Möglichkeiten:

- Sie können dem Panel einen eigenen Layout Manager zuteilen. Damit können Sie einer Region Ihres Displays ein eigenes Layout zuordnen.
- Sie können dem Panel weitere Komponenten zuordnen und damit kreativere Layouts erstellen.

### 1.2.8.2. Kreieren von Panels und komplexen Layouts

Das Programmbeispiel unten verwendet ein Panel, um zwei Knöpfe in der Nord Region des Border Layouts zu platzieren.



Um ein passendes Layout zu erreichen, werden verschiedene Komponenten und Layouts kombiniert. Der Panel im Norden wird genau so behandelt wie eine Komponente.

Wenn Sie die Grösse des Fensters verändern, passt sich das Layout den neuen Gegebenheiten an. Grösse und Position der Knöpfe werden durch den Flow Layout Manager jeweils angepasst. Das nebenstehende Bild zeigt die minimale Breite des Fensters: schmäler lässt sich das Fenster nicht machen, ausser als Icon.



Der Flow Layout Manager ist der Standard Layout Manager für das Panel.

Panels werden mit dem Konstruktor `Panel()` kreiert. Nachdem ein `Panel` Objekt kreiert ist, muss es einem Container zugeordnet werden. Dies geschieht mit der `add()` Methode des Containers.



## 1.2.8.3. Komplexeres Layout Beispiel

Und hier folgt das Listing für das obige Programm:

```
package nochinguibeispiel;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * Title:
 * @author J.M.Joller
 * @version 1.0
 */

public class GUIBeispiel extends Applet {
    private Frame f;
    private Panel p;
    private Button bw, bc;
    private Button bfile, bhelp;

    public void start() {
        GUIBeispiel gui = new GUIBeispiel();
        gui.go();
    }

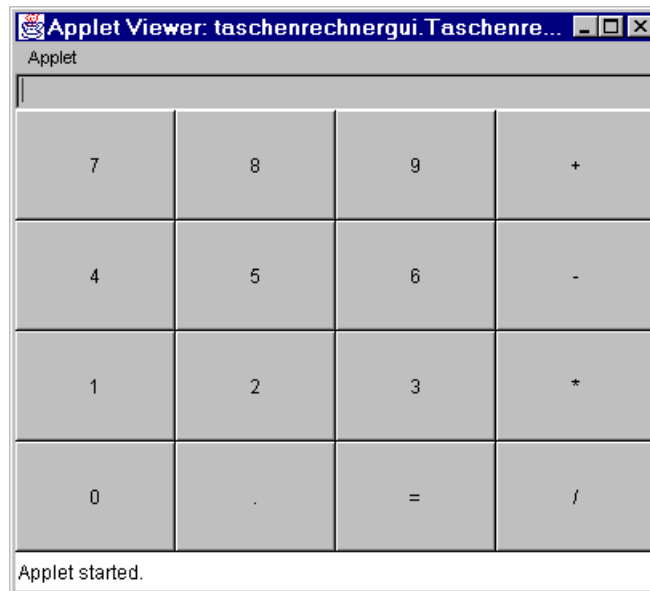
    public void go() {
        f = new Frame("GUI Beispiel Programm");
        bw = new Button("West");
        bc = new Button("Arbeitsfläche");
        f.add(bw, "West");
        f.add(bc, "Center");
        p = new Panel();
        f.add(p, "North");
        bfile = new Button("Datei");
        bhelp = new Button("Hilfe");
        p.add(bfile);
        p.add(bhelp);
        f.pack();
        f.setVisible(true);
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.2.9. Praktische Übung

### 1.2.9.1. Kreieren Sie ein GUI für einen Taschenrechner

Als erstes ist es Ihre Aufgabe das GUI eines Taschenrechners zu bauen. Die Event Handler und die eigentliche Funktionalität werden wir später hinzufügen.



### 1.2.9.2. Einführung in die Aufgabe

Um Ihnen viel Arbeit abzunehmen, können Sie im Folgenden einfach passenden Programmcode auswählen. Ziel ist es, wie oben erwähnt, einen Taschenrechner zu bauen, mit einem GUI etwa wie oben.

Das resultierende Applet sollte etwa wie oben aussehen.

Nachdem Sie das Applet GUI entwickelt / selektiert haben, können Sie auch mit dem JBuilder Designer ein ähnliches GUI zeichnen und den generierten Programmcode mit dem sicher viel einfacheren, ausgewählten vergleichen.

Sie finden eine Musterlösung auf dem Server / dem Web / der CD.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
private key0 Button;
```

```
private Button key0;
```

```
public Button key0;
```

```
import java.awt.*;  
import java.applet.*;
```

```
public class Calc extends Applet {
```



```
}
```

```
private Panel buttonArea;
```

```
buttonArea Panel;
```

```
private buttonArea Panel;
```

```
import java.awt.*;  
import java.applet.*;
```

```
public class Calc extends Applet {
```

```
private Button key0, key1, key2, key3, key4,  
key5, key6, key7, key8, key9;  
private Button keyequal, keyplus, keyminus,  
keyperiod, keymult, keydiv;
```



```
}
```

```
newButton(key,"0");
```

```
key0 = new Button(0);
```

```
key0 = new Button("0");
```

```
key0, key5, key/, keyd, keyd;  
private Button keyequal, keyplus, keyminus,  
keyperiod, keymult, keydiv;  
private Panel buttonArea;  
private TextField answer;  
private boolean periodSet = false;  
private boolean clearNow = false;  
private boolean clearAfter = false;  
private double firstNumber = 0;  
private String lastMethod;
```

```
public void init() {  
    answer = new TextField(15);  
    answer.setEditable(false);
```



```
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
buttonArea = new Panel();
buttonArea.setLayout(new GridLayout(4,4));
```

```
buttonArea = new Panel();
```

```
buttonArea = new Panel();
buttonArea.setLayout();
```

```
buttonArea.add(keydiv);
```

```
buttonArea.add(key7);
```

```
buttonArea.add(key0);
```

```
setLayout(new BorderLayout());
add("North",answer);
add("Center",buttonArea);
```

```
setLayout(new BorderLayout);
add("Center",answer);
add("North",buttonArea);
```

```
setLayout(new FlowLayout());
add("North",answer);
add("Center",buttonArea);
```

```
key1 = new Button("1");
key2 = new Button("2");
key3 = new Button("3");
key4 = new Button("4");
key5 = new Button("5");
key6 = new Button("6");
key7 = new Button("7");
key8 = new Button("8");
key9 = new Button("9");
keyequal = new Button("=");
keyplus = new Button("+");
keyminus = new Button("-");
keymult = new Button("*");
keydiv = new Button("/");
keyperiod = new Button(".");
```



```
}
}
```

```
key5 = new Button("5");
key6 = new Button("6");
key7 = new Button("7");
key8 = new Button("8");
key9 = new Button("9");
keyequal = new Button("=");
keyplus = new Button("+");
keyminus = new Button("-");
keymult = new Button("*");
keydiv = new Button("/");
keyperiod = new Button(".");
```

```
buttonArea = new Panel();
buttonArea.setLayout(new GridLayout(4,4));
```

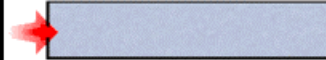


```
}
}
```

```
buttonArea.add(key4);
buttonArea.add(key5);
buttonArea.add(key6);
buttonArea.add(keyminus);
buttonArea.add(key1);
buttonArea.add(key2);
buttonArea.add(key3);
buttonArea.add(keymult);
buttonArea.add(key0);
buttonArea.add(keyperiod);
buttonArea.add(keyequal);
buttonArea.add(keydiv);
```

```
}
```

```
public void start() {
```



```
}
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        buttonArea.add(key4);
        buttonArea.add(key5);
        buttonArea.add(key6);
        buttonArea.add(keyminus);
        buttonArea.add(key1);
        buttonArea.add(key2);
        buttonArea.add(key3);
        buttonArea.add(keymult);
        buttonArea.add(key0);
        buttonArea.add(keyperiod);
        buttonArea.add(keyequal);
        buttonArea.add(keydiv);
    }

    public void start() {
        setLayout(new BorderLayout());
        add("North",answer);
        add("Center",buttonArea);
    }
}
```

## 1.2.9.3. Musterlösung für das Taschenrechner GUI

Damit Sie nicht auf dem Server nachschauen müssen:

```
package taschenrechnergui;

import java.awt.*;
import java.applet.*;

/**
 * Title:
 * @author J.M.Joller
 * @version 1.0
 */

public class TaschenrechnerGUIBeispiel extends Applet {

    private Button taste0,taste1,taste2,taste3,taste4,
                    taste5,taste6,taste7,taste8,taste9;
    private Button tasteGleich,tastePlus,tasteMinus,
                    tastePunkt,tasteMult,tasteDiv;
    private Panel buttonBereich;

    private TextField resultat;
    private boolean punktGesetzt = false;
    private boolean jetztLoeschen = false;
    private boolean spaeterLoeschen = false;
    private double ersteZahl = 0;
    private String zuletztVerwendeteMethode;

    public void init() {
        resultat = new TextField(15);
        resultat.setEditable(false);
        resultat.setBackground(Color.white);

        taste0 = new Button("0");
        taste1 = new Button("1");
        taste2 = new Button("2");
        taste3 = new Button("3");
        taste4 = new Button("4");
        taste5 = new Button("5");
        taste6 = new Button("6");
        taste7 = new Button("7");
        taste8 = new Button("8");
        taste9 = new Button("9");
        tasteGleich = new Button("=");
        tastePlus = new Button("+");
        tasteMinus = new Button("-");
        tasteMult = new Button("*");
        tasteDiv = new Button("/");
        tastePunkt = new Button(".");

        buttonBereich = new Panel();
        buttonBereich.setLayout(new GridLayout(4,4));

        buttonBereich.add(taste7);
        buttonBereich.add(taste8);
        buttonBereich.add(taste9);
        buttonBereich.add(tastePlus);
        buttonBereich.add(taste4);
        buttonBereich.add(taste5);
        buttonBereich.add(taste6);
        buttonBereich.add(tasteMinus);
        buttonBereich.add(taste1);
        buttonBereich.add(taste2);
        buttonBereich.add(taste3);
        buttonBereich.add(tasteMult);
        buttonBereich.add(taste0);
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        buttonBereich.add(tastePunkt);
        buttonBereich.add(tasteGleich);
        buttonBereich.add(tasteDiv);
    }

    public void start() {
        setLayout(new BorderLayout());
        add("North",resultat);
        add("Center",buttonBereich);
    }
}
```

## 1.2.10. Quiz

Hier einige Fragen als Test und um Sie zur eventuellen Wiederholung des Moduls zu bewegen, falls Sie grundlegendes noch nicht so richtig verstanden haben.

1. Wie können Sie dafür sorgen, mittels Layout Manager, dass die Breite und Position eines Buttons zwar veränderlich ist, nicht aber seine Höhe?<sup>1</sup>
  - a) North, South Panel in einem Border Layout
  - b) East, West Panel in einem Border Layout
  - c) Center Panel eines BorderLayouts
  - d) mittels GridLayout
  - e) mittels FlowLayout
2. Welche der folgenden sind Layout Manager in AWT?<sup>2</sup>
  - a) FlowLayout
  - b) BagLayout
  - c) BorderLayout
  - d) GridLayout
  - e) CardLayout
3. Welcher Layout Manager gestattet es Ihnen Komponenten von einer Zeile auf die nächste "fließen" zu lassen oder auf einer Zeile zu zentrieren?<sup>3</sup>
  - a) BorderLayout
  - b) GridLayout
  - c) FlowLayout
  - d) CardLayout
  - e) GridBagLayout
4. Welcher Layout Manager gestattet es Ihnen Zellen zu definieren und gestattet es den Komponenten individuelle Grössen zu spezifizieren?<sup>4</sup>
  - a) BorderLayout
  - b) GridLayout
  - c) FlowLayout
  - d) CardLayout
  - e) GridBagLayout
5. Welche der folgenden GUI Komponenten sind Unterklassen der Component Klasse?<sup>5</sup>
  - a) Button
  - b) Container
  - c) Menu
  - d) List
  - e) Scrollbar

---

<sup>1</sup> a)

<sup>2</sup> a) c) d) e)

<sup>3</sup> c)

<sup>4</sup> e)

<sup>5</sup> a) b) d) e)



## 1.2.11. Zusammenfassung

In diesem Modul haben Sie gelernt:

- aus welchen grundlegenden Klassen das AWT Package aufgebaut ist.
- aus welchen Komponenten das AWT Package besteht, aus welchen Komponenten.
- wie Container, Komponenten und Layout Manager grundsätzlich funktionieren und wie sie zusammen GUIs aufbauen.
- wie unterschiedliche Layout Manager flow, border, grid und card Layout Manager) eingesetzt werden können, um die gewünschten Layouts zu erreichen
- wie Frames und Panel Container eingesetzt werden
- wie Sie Panels in andere Container plazieren können, um komplexere Layouts erreichen zu können.

## 1.3. Modul 2 : Das AWT Event Modell

### In diesem Modul

- Modul 2 : Das AWT Event Modell
  - Modul Einleitung
  - Lektion 1 - Event Grundlagen
  - Lektion 2 - JDK 1.0 versus neues Event Modell
  - Lektion 3 - Java GUI Verhalten
  - Taschenrechner mit Event Handler
  - Praktische Übung
  - Quiz
  - Zusammenfassung

### 1.3.1. Modul Einleitung

In diesem Modul bauen wir auf dem Material des vorhergehenden Moduls auf und beschäftigen uns mit den Ereignissen, den *Events*, mit deren Hilfe die Dynamik in einem Windowing System gesteuert werden können. Java Events sind ein wichtiger Bestandteil des AWT Packages. Unter Event verstehen wir einen Mausklick, einen Tastendruck, das Scrollen eines Fensters, das Verschieben oder Verändern eines

Fensters.

#### 1.3.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

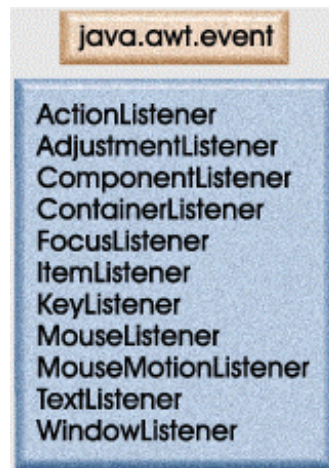
- Programmcode zu schreiben, mit dem Events behandelt werden können, welche im Benutzerinterface geschehen.
- die passenden Interfaces und *Handler* Methoden für unterschiedliche Ereignistypen einzusetzen.
- aus dem Event Objekt Details der Benutzeraktionen zu bestimmen, welche zu diesem Ereignis geführt hat.
- zu einem Event *Listener* passende *Adapter* Klassen auszuwählen.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.3.2. Lektion 1 - Event Grundlagen

### 1.3.2.1. Ein Ereignis ist ...

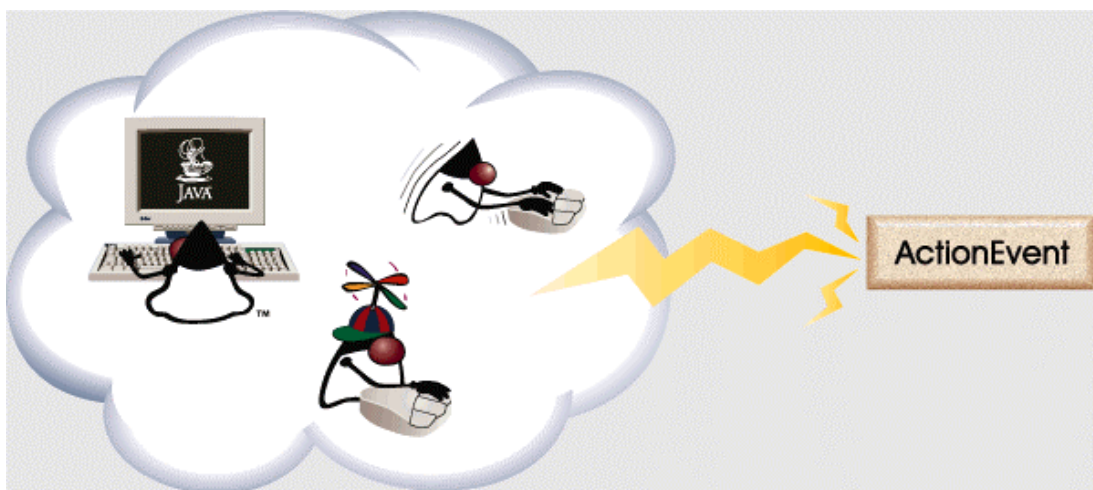
Immer wenn im Benutzerinterface eine Aktion stattfindet, wird ein *Event* generiert. Events sind Objekte, welche beschreiben, was geschieht. In der Grafik unten sehen Sie eine Liste mit Events, Eventklassen, mit deren Hilfe sehr unterschiedliche Kategorien von Benutzeraktionen beschrieben werden.



### 1.3.2.2. Ereignisquellen

Eine Ereignisquelle (bezogen auf Benutzeroberflächen) ist das Ergebnis irgend einer Benutzeraktion an einer AWT Komponente. Beispielsweise generiert ein Mausklick auf eine Button Komponente (als Quelle des Ereignisses) ein *ActionEvent*. Dieses *ActionEvent* ist ein Objekt, welches Informationen über den Status des Ereignisses enthält:

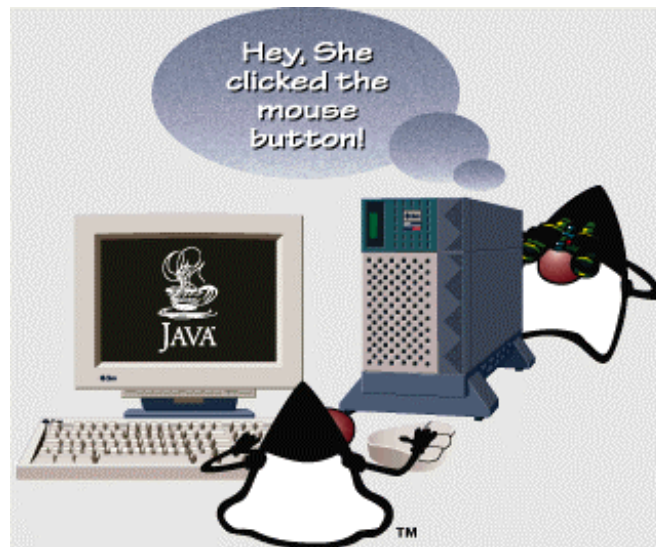
- *ActionCommand* - ein Befehl, der mit der Aktion vekiüpft ist
- *Modifiers* - irgend ein Modifier der Aktion



## 1.3.2.3. Event Handler

Falls ein Ereignis eintritt, empfängt die Komponente, mit der der Benutzer interagiert (Button, Slider, Textfeld,...), das Ereignisobjekt.

Ein *Event Handler* ist eine Methode, welche das Ereignisobjekt empfängt. Der Event Handler ist damit für eine Reaktion, die Bearbeitung des Ereignisobjekts, zuständig.

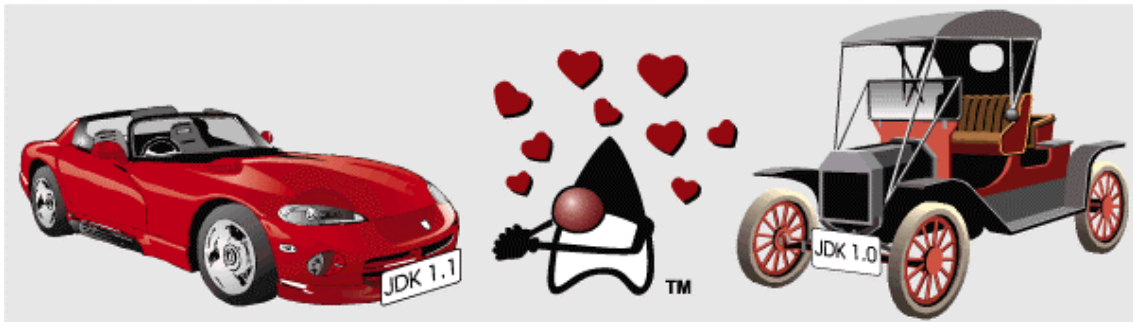


## 1.3.3. Lektion 2 - JDK 1.0 versus neues Event Modell

### 1.3.3.1. Wie werden Ereignisse verarbeitet?

Zwischen JDK1.0 und JDK1.1 bestehen sehr fundamentale Unterschiede in der Art und Weise, wie Ereignisse empfangen und verarbeitet werden. In diesem Abschnitt vergleichen wir das alte (JDK1.0) Event Modell mit dem aktuellen Ereignismodell aus JDK1.1.

In JDK 1.0 wurde ein hierarchisches Ereignismodell verwendet; in JDK 1.1 wurde ein sogenanntes Delegations- Ereignismodell eingeführt. Im Rahmen des Kurses halten wir uns an das aktuelle Delegationsmodell. Aber es ist wichtig den Unterschied der beiden Event Konzepte zu kennen.



## 1.3.3.2. Das hierarchische Modell (JDK1.0)

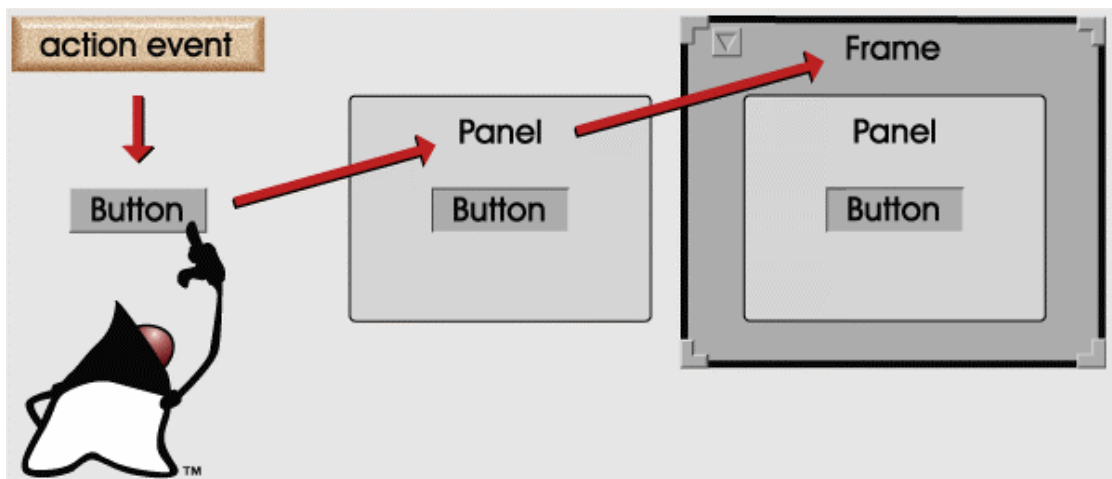
Beim hierarchischen Modell versucht man die Tatsache, dass einzelne Komponenten in anderen Komponenten oder Containern enthalten sind, als Basis für ein Ereignismodell zu verwenden:

- Ereignisse werden zuerst innerhalb der Komponenten behandelt, in denen Sie auftreten.
- Ereignisse, welche nicht innerhalb einer Komponente behandelt werden können, propagieren hierarchisch.

Das heisst, dass ein Ereignis, falls es nicht lokal abgehandelt werden kann, automatisch zum Container der Komponente weitergereicht wird.

### Bemerkung

Unter Container verstehen wir in diesem Zusammenhang nicht Objekte der Container Klasse sondern GUI-mässige Container.



Im obigen Beispiel sendet ein Mausklick auf den Button ein Action Event Objekt zuerst zum Button Objekt (welches sich in einem Panel eines Frame befindet)

Falls das Ereignis nicht vom Button behandelt wird, wird das Event Objekt an das Panel Objekt weitergeleitet, falls es dort nicht behandelt wird, wird das Event an das Frame Objekt gesandt.

Dieses hierarchische Modell besitzt folgende Vorteile:

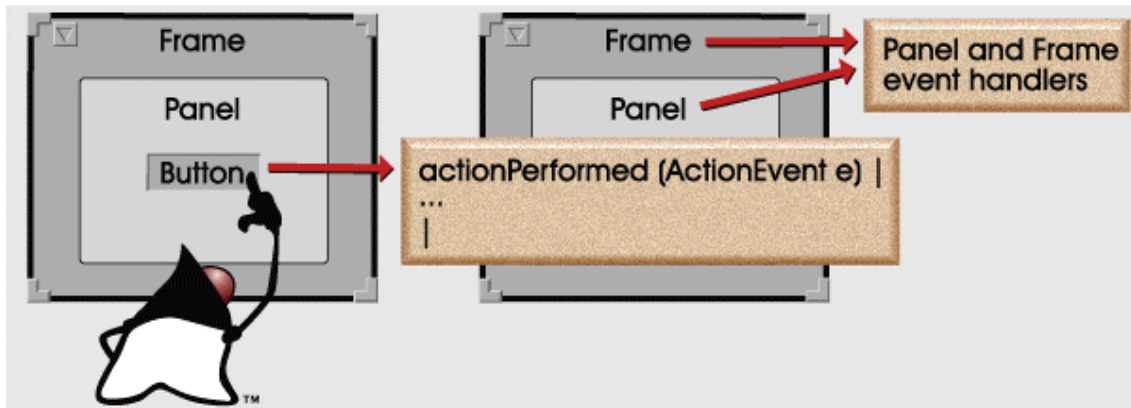
- es ist einfach und der objektorientierten Denkweise angepasst.

Das hierarchische Modell besitzt folgende Nachteile:

- es ist schwierig bestimmte Ereignisse herauszufiltern.
- auf der Klassenebene der Ereignisse können komplexe Hierarchien entstehen oder es muss auf einer sehr tiefen Stufe eine sehr umfangreiche `handleEvent()` Methode zur Verfügung stehen.

## 1.3.3.3. Das Delegationsmodell (JDK1.1)

In JDK 1.1 wurde ein Ereignismodell eingeführt, das Delegations- Eventmodell. In diesem Modell werden die Ereignisse, die Event Objekte, auch zuerst an die Quellkomponente gesandt. Aber es ist jeder Komponente überlassen, eine *Event Handler Methode* (einen *Listener*) zu definieren, um das Ereignis abzufangen. Damit ist es möglich, zur Behandlung des Ereignisses eine **eigene Klasse** zu definieren, welche das Ereignis behandeln kann.



Events werden nur von registrierten Listeners erkannt. Jedes Ereignis besitzt eine entsprechende Listener Klasse (Interface). Die Klasse, welche das Listener Interface implementiert, definiert die Methoden, welche ein Event Objekt empfangen können.

Komponenten, welche keinen Listener besitzen, empfangen auch keine Ereignisse und leiten keine weiter.

Im folgenden Beispiel sehen Sie ein einfaches `Frame` mit einem `Button` darauf. Die `ButtonHandler` Klasse, ist die Handler Klasse, an die die Behandlung des Ereignisses delegiert wird.

Worauf kommt es bei der Definition der `ButtonHandler` Klasse an?

- Die `Button` Klasse besitzt eine `addActionListener (ActionListener)` Methode.
- Das `ActionListener` Interface definiert eine einzige Methode, `actionPerformed`, welche ein `ActionEvent` empfängt.
- Falls ein `Button` Objekt kreiert wird, kann das Objekt einen Listener für das `ActionEvents` angeben (registrieren). Dies geschieht mittels der `addActionListener` Methode, welche das Objekt spezifiziert, welches das `ActionListener` Interface implementiert.
- Falls das `Button` Objekt angeklickt wird, wird ein `ActionEvent` an den `ActionListener` gesandt, der von der `actionPerformed` Methode angegeben wird.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
package buttonhandler;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class ButtonHandlerBeispiel extends Applet {
    public void start() {
        Frame f = new Frame("Button Handler Beispiel");
        Button b = new Button("Klick!");
        b.addActionListener(new EinfacherButtonHandler() );
        f.add("Center", b);
        f.setSize(100,100);
        f.setVisible(true);
    }
}
```

```
package buttonhandler;

import java.awt.event.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class EinfacherButtonHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Es ist ein Ereignis eingetreten");
    }
}
```

Ausführungsprotokoll:



```
Es ist ein Ereignis eingetreten
Es ist ein Ereignis eingetreten
Es ist ein Ereignis eingetreten
...
```

## Aufgaben

- 1) Fragen Sie mindestens 5 der für Ereignisse definierten Methoden im obigen Beispiel ab. Die aktuelle Version des obigen Beispiels (auf dem Server/Web/ der CD) enthält diese Modifikationen, ist also die Musterlösung.
- 2) Fügen Sie einen WindowHandler hinzu, mit dem das Fenster geschlossen und das Applet beendet werden kann (auch hier finden Sie die Musterlösung wie oben).



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

Das Delegationsmodell besitzt verschiedene Vorteile:

- Ereignisse werden nicht zufällig auf einem falschen Level behandelt: im hierarchischen Modell konnte dies passieren, weil Ereignisse weitergereicht wurden und eventuell auf einem Level abgehandelt wurden, auf dem man dies eigentlich nicht tun sollte oder wollte.
- Mit Hilfe von Filter (Adapter) Klassen können Sie Ereignisse und die zugeordneten Aktionen klassifiziert werden.
- Das Delegationsmodell verteilt die Arbeit unter den Komponenten.
- Das neue (Delegations. ) Modell unterstützt auch JavaBeans. JavaBeans sind wiederverwendbare Software Komponenten, welche in Java geschrieben wurden. JavaBeans werden visuell zusammengebaut, es handelt sich um spezielle Java Komponenten, die mit Hilfe eines speziellen Werkzeugs gebaut werden.

Das Delegationsmodell besitzt auch Nachteile und lässt Fragen offen:

- Das Modell ist komplexer und nicht so leicht verständlich wie das hierarchische Modell.
- Falls Sie eine Applikation mit JDK 1. 0 entwickelt haben ist es recht schwierig, die Applikation auf JDK 1.1+ zu migrieren.
- Eigentlich unterstützt das neue Ereignismodell auch noch das JDK1.0 Modell. Aber Sie sollten es unterlassen Modelle zu mischen. Im schlimmsten Fall resultiert ein chaotisches Verhalten.

## 1.3.4. Lektion 3 - Java GUI Verhalten

### 1.3.4.1. Die Event Kategorien

Der generelle Mechanismus zum Empfangen von Ereignissen von Komponenten haben wir bisher lediglich auf ein Ereignis bezogen besprochen, ausser Sie haben die Übung mit dem Window Handler gemacht.

Im `java.awt.event` Package sind bereits viele Ereignisse vordefiniert. Komponenten von Fremdherstellern oder Sie selbst, können weitere Ereignisse hinzufügen.

Für jede Kategorie von Ereignissen muss ein Interface definiert werden. Jede Klasse, welche Ereignisse dieser Kategorie empfangen (und bearbeiten) möchte, muss dieses Interface implementieren.

In der Regel definiert das Interface auch Methoden, welche implementiert werden müssen. Diese Methoden werden aufgerufen, falls ein bestimmtes Ereignis eintritt.

Als Beispiel kann die Musterlösung zur obigen Aufgabe dienen, in der Sie einen WindowListener implementieren mussten. WindowListener ist eine vordefinierte Kategorie von Ereignissen für Frames.

Die Tabelle auf der folgenden Seite fasst die Kategorien, deren Interfaces und die Methoden der Interfaces zusammen.

Die Methodennamen sind in der Regel selbstsprechend. Sie geben die Bedingungen an, unter denen eine Methode aufgerufen wird.

Falls Sie eigene Kategorien definieren, sollten Sie sich an diese Konvention halten.

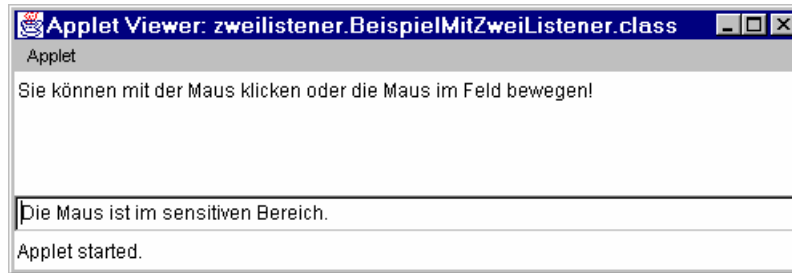
# JAVA AWT GUI UND APPLLET GRUNDLAGEN

Kategorie	Interface	Methoden
Action	ActionListener	actionPerformed (ActionEvent)
Item	ItemListener	itemStateChanged (ItemEvent)
Mouse Motion	MouseMotionListener	mouseDragged (MouseEvent) mouseMoved (MouseEvent)
Mouse Button	MouseListener	mousePressed (MouseEvent) mouseReleased (MouseEvent) mouseEntered (MouseEvent) mouseExited (MouseEvent) mouseClicked (MouseEvent)
Key	KeyListener	keyPressed (KeyEvent) keyReleased (KeyEvent) keyTyped (KeyEvent)
Focus	FocusListener	focusGained (FocusEvent) focusLost (FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved (ComponentEvent) componentHidden (ComponentEvent) componentResized (ComponentEvent) componentShown (ComponentEvent)
Window	WindowListener	windowClosing (WindowEvent) windowOpened (WindowEvent) windowIconified (WindowEvent) windowDeiconified (WindowEvent) windowClosed (WindowEvent) windowActivated (WindowEvent) windowDeactivated (WindowEvent)
Container	ContainerListener	componentAdded (ContainerEvent) componentRemoved (ContainerEvent)
Text	TextListener	textValueChanged (TextEvent)

## 1.3.4.2. Ein komplexeres Beispiel

Unten sehen Sie ein Beispiel, dessen Programmcode Sie darunter sehen. In diesem Beispiel können Sie zwei Events abfangen:

- falls die Maus in den sensiblen Bereich kommt, wird dies im Textfeld darunter angezeigt.
- falls Sie die Maustaste drücken und die Maus bewegen, sehen Sie die Position der Maus im sensitivenBereich.



Diese Ereignisse können von irgend einer Klasse abgefangen werden, welche das `MouseMotionListener` Interface implementiert. Dieses Interface definiert die Signatur für zwei Methoden, `mouseDragged()` und `mouseMoved()`. Wir sind lediglich an den Mausereignissen "bewegen der Maus im geklickten Zustand" [`mouseDragged`] und dem Betreten oder Verlassen des Bereiches interessiert.

Das Interface stellt jedoch diese Methoden zur Verfügung. Diese müssen wir auch, eventuell leer, implementieren. Die Methode `mouseMoved()` kann beispielsweise leer implementiert werden.

Um weitere Ereignisse abfangen zu können, beispielsweise Mausklicks, müssen wir das `MouseListener` Interface implementieren. Dieses Interface umfasst mehrere weitere Ereignisse, beispielsweise `mouseEntered`, `mouseExited`, `mousePressed`, `mouseReleased`, und `mouseClicked`.

Im Beispiel verwenden wir `mouseEntered` um feststellen zu können, ob sich die Maus im Appletbereich befindet. `mouseExited` zeigt das Verlassen der sensitiven Fläche des Applets an.

Die anderen drei Methoden benötigen wir im Moment nicht. Daher lassen wir sie leer, müssen Sie aber aufführen, da wir das Interface implementieren möchten.

Damit wir irgend etwas anzeigen können, schreiben wir die Position der Maus in das Textfeld.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.3.4.2.1. Erläuterungen zum Programm

Beachten Sie, dass wir bereits in der Definition der Klasse die Interfaces aufführen:

```
implements MouseMotionListener, MouseListener
```

Sie können mehrere Interfaces aufführen. Diese müssen einfach durch Kommata getrennt werden.

Beim Aufruf der Handler Methoden, beispielsweise `mouseDragged()`, werden weitere Informationen über das Ereignis, beispielsweise Informationen über die Quelle des Ereignisses, an die Methode übergeben. (Falls Sie die Aufgabe weiter oben gelöst haben, kennen Sie bereits mehrere Informationen, die Sie abfragen können).

Die vollständige Dokumentation der Ereignisse finden Sie unter `java.awt.event` (Package).

Die Methodenaufrufe:

```
this.addMouseListener(this);  
this.addMouseMotionListener(this);
```

rufen Methoden aus der aktuellen Klasse auf. Sie können so viele Eventquellen angeben wie Sie möchten.

## 1.3.4.2.2. Der Programmcode

```
package zweilistener;  
  
import java.awt.*;  
import java.awt.event.*;  
import java.applet.*;  
  
/**  
 * Title:  
 * Description:  
 * @author J.M.Joller  
 * @version 1.0  
 */  
  
public class BeispielMitZweiListener extends Applet implements  
MouseMotionListener, MouseListener {  
  
    private TextField tf;  
  
    public void init() {  
        this.setLayout(new BorderLayout());  
        this.add ("North", new Label ("Sie können mit der Maus klicken oder  
                                     die Maus im Feld bewegen!"));  
  
        tf = new TextField (30);  
        this.add ("South", tf);  
  
        this.addMouseMotionListener(this);  
        //this: die Methoden sind in dieser Klasse definiert  
        this.addMouseListener (this);  
        this.setSize(300, 100);  
        this.setBackground(Color.white);  
    }  
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public void start() {
    this.setVisible(true);
}

public void stop() {
    this.setVisible(false);
}

// Beachten Sie : alle Methoden des Interfaces müssen
//                implementiert werden,
//                selbst wenn dies leer geschieht

public void mouseDragged (MouseEvent e) {
    String s = "Maus Position:  X = " + e.getX() + " Y = " + e.getY();
    tf.setText (s);
}

public void mouseMoved (MouseEvent e) {}

// MouseListener Events
public void mouseClicked (MouseEvent e) {}

public void mouseEntered (MouseEvent e) {
    String s = "Die Maus ist im sensitiven Bereich.";
    tf.setText (s);
}

public void mouseExited (MouseEvent e) {
    String s = "Die Maus ist abgehauen!";
    tf.setText (s);
}

public void mousePressed (MouseEvent e) {}

public void mouseReleased (MouseEvent e) {}

}
```

## 1.3.4.3. Mehrere Listeners

Das AWT Event Listener Framework gestattet die Definition mehrerer Listener pro Komponente. Sie können also mehrere Listener an eine Komponente anhängen, wobei sich die Listener auf ein einziges Ereignis beziehen (zwei unterschiedliche Listener mit dem selben Ereignis).

Sie könnten ein solches Verhalten auch mit einem einzigen Handler behandeln. Aber es gibt Fälle, in denen Sie mehr Flexibilität benötigen, beispielsweise in kontextsensitiven Hilfesystemen.

Der Listener Mechanismus gestattet es Ihnen eine `addListener` Methode so oft Sie wollen aufzurufen, jeweils mit unterschiedlichen Listeners, je nach Bedarf. Alle registrierten Listener besitzen ihre eigenen Handler Methoden, welche aufgerufen werden, sobald das Ereignis eintritt.

Die Reihenfolge, in der die Handler Methoden aufgerufen werden, ist nicht fix vorgegeben. Die Überlegung ist einfach die:

- falls die Reihenfolge eine Rolle spielt, sollten Sie vermutlich das Problem anders lösen, beispielsweise mit einem Listener.
- falls Sie wirklich mehrere Listener verwenden, um klarere Programmstrukturen erreichen zu können, spielt wahrscheinlich die Reihenfolge sowieso keine Rolle.

## 1.3.4.4. Ereignis Adapter

Sie erinnern sich noch: da wir Interfaces implementieren, müssen wir alle dort definierten Methoden definieren, obschon wir eventuell nur einen Bruchteil davon benötigen. Die `Listener` Interfaces, speziell das `MouseListener` Interface und das `ComponentListener` Interface sind sehr umfangreich.

Das `MouseListener` Interface, zum Beispiel, definiert folgende Methoden:

- `mouseClicked (MouseEvent)`
- `mouseEntered (MouseEvent)`
- `mouseExited (MouseEvent)`
- `mousePressed (MouseEvent)`
- `mouseReleased (MouseEvent)`

Damit Sie nicht soviel Arbeit haben, stellt Java Ihnen zu jedem `Listener` Interface eine dazu passende *Adapter* Klasse, welche das entsprechende `Listener` Interface implementiert, die Implementation der Methoden offenlässt.

Im folgenden Beispiel sehen Sie ein Beispiel für den praktischen Einsatz der Adapterklassen. Sie können die Methoden der Adapterklassen jederzeit überschreiben oder auch direkt das Interface implementieren.

## 1.3.4.5. Programmbeispiel

Adapterklassen haben den Vorteil, dass Sie nicht alle Methoden implementieren müssen. Aber diese Klassen haben auch einen entscheidenden Nachteil: es sind Klassen. Sie können damit beispielsweise Applets nicht mehr so einfach implementieren, da Sie nicht mehrere Klassen erweitern können. Java kennt bekanntlich keine Mehrfachvererbung!

```
1  import java.awt.event.*;
2
3  public class MouseClickHandler extends MouseAdapter {
4
5      // Jetzt können wir einfach die benötigte Methoden
6      // und nur diese
7      // überschreiben.
8      public void mouseClicked (MouseEvent e) {
9          // was auch immer passiert...
10     }
11 }
```

## 1.3.5. Taschenrechner mit Event Handler - Praktische Übung

### 1.3.5.1. Einleitung

Im folgenden Beispiel ergänzen Sie durch Auswahl die jeweils passenden Programmteile zur Konstruktion eines Taschenrechners, diesmal mit Ereignissteuerung!

Die Logik des Taschenrechners ist recht einfach! Wir wollen lediglich einfache Operatione durchführen.

Am Ende der Übung finden Sie die Lösung, in leicht modifizierter Form, da die Lösung wächst und der Programmcode die jeweils aktuelle Version wiedergibt.



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public class CalcEvent extends  
    ActionListener implements Applet{  
}
```

```
public class CalcEvent extends Applet  
    implements ActionListener {  
}
```

```
public class CalcEvent extends  
    ActionListener {  
}
```

```
import java.awt.*;  
import java.awt.event.*;  
import java.applet.*;
```

```
key0.addActionListener(this);
```

```
key0.addActionListener();
```

```
addActionListener(key0);
```

```
key1 = new Button("1");  
key2 = new Button("2");  
key3 = new Button("3");  
key4 = new Button("4");  
key5 = new Button("5");  
key6 = new Button("6");  
key7 = new Button("7");  
key8 = new Button("8");  
key9 = new Button("9");  
keyequal = new Button("=");  
keyplus = new Button("+");  
keyminus = new Button("-");  
keymult = new Button("*");  
keydiv = new Button("/");  
keyperiod = new Button(".");
```

```
public void performAction (ActionEvent e) {  
    String bLabel = (String)e.getActionCommand();  
}
```

```
public void actionPerformed() {  
}
```

```
public void actionPerformed (ActionEvent e) {  
    String bLabel = (String)e.getActionCommand();  
}
```

```
buttonArea.add(keyminus);  
buttonArea.add(key1);  
buttonArea.add(key2);  
buttonArea.add(key3);  
buttonArea.add(keymult);  
buttonArea.add(key0);  
buttonArea.add(keyperiod);  
buttonArea.add(keyequal);  
buttonArea.add(keydiv);  
}
```

```
public void start() {  
    setLayout(new BorderLayout());  
    add("North",answer);  
    add("Center",buttonArea);  
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

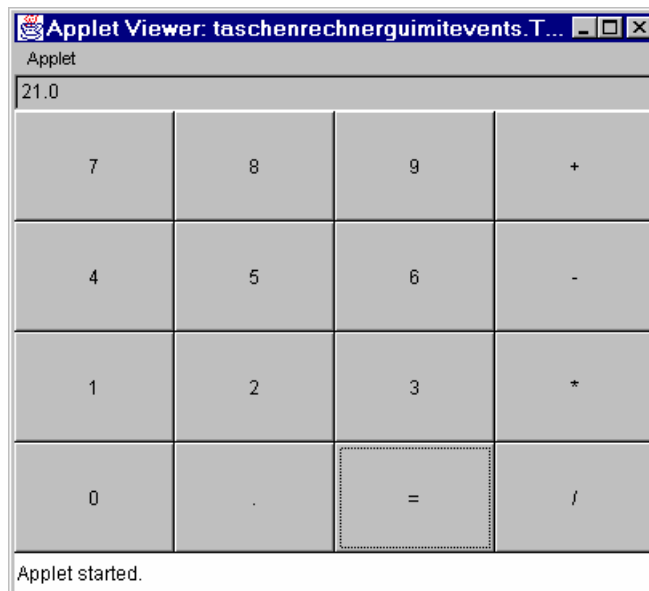
```
        buttonArea.add(keyminus);
        buttonArea.add(key1);
        buttonArea.add(key2);
        buttonArea.add(key3);
        buttonArea.add(keymult);
        buttonArea.add(key0);
        buttonArea.add(keyperiod);
        buttonArea.add(keyequal);
        buttonArea.add(keydiv);
    }

    public void start() {
        setLayout(new BorderLayout());
        add("North", answer);
        add("Center", buttonArea);
    }

    public void actionPerformed (ActionEvent e) {
        String bLabel = (String)e.getActionCommand();
    }
}
```

Nachdem Sie die Übung durchgearbeitet haben, wollen Sie sicher das fertige Applet ausführen und testen.

Auf den folgenden Seiten finden Sie den Programmcode für die Lösung. Auf dem Server / Web / der CD finden Sie das vollständige Projekt.



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.3.5.1.1. Der Taschenrechner

Das obige Bild stammt von folgendem Applet:

```
package taschenrechnerguimitevents;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * Title:
 * Description:
 * Copyright: Copyright (c) 2000
 * Company: Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

public class TaschenRechner extends Applet implements ActionListener{

    private Button taste0,taste1,taste2,taste3,taste4,taste5,
                    taste6,taste7,taste8,taste9;
    private Button tasteGleich,tastePlus,tasteMinus,tastePunkt,
                    tasteMult,tasteDiv;
    private Panel buttonArea;

    private TextField resultat;
    private boolean punktGesetzt = false;
    private boolean jetztLoeschen = false;
    private boolean spaeterLoeschen = false;
    private double ersteZahl = 0;
    private String zuletztVerwendeteMethode;

    public void init() {
        resultat = new TextField(15);
        resultat.setEditable(false);
        resultat.setBackground(Color.white);

        taste0 = new Button("0");
        taste1 = new Button("1");
        taste2 = new Button("2");
        taste3 = new Button("3");
        taste4 = new Button("4");
        taste5 = new Button("5");
        taste6 = new Button("6");
        taste7 = new Button("7");
        taste8 = new Button("8");
        taste9 = new Button("9");
        tasteGleich = new Button("=");
        tastePlus = new Button("+");
        tasteMinus = new Button("-");
        tasteMult = new Button("*");
        tasteDiv = new Button("/");
        tastePunkt = new Button(".");
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
taste0.addActionListener(this);
taste1.addActionListener(this);
taste2.addActionListener(this);
taste3.addActionListener(this);
taste4.addActionListener(this);
taste5.addActionListener(this);
taste6.addActionListener(this);
taste7.addActionListener(this);
taste8.addActionListener(this);
taste9.addActionListener(this);
tastePlus.addActionListener(this);
tasteMinus.addActionListener(this);
tasteMult.addActionListener(this);
tasteDiv.addActionListener(this);
tasteGleich.addActionListener(this);
tastePunkt.addActionListener(this);

buttonArea = new Panel();
buttonArea.setLayout(new GridLayout(4, 4));

buttonArea.add(taste7);
buttonArea.add(taste8);
buttonArea.add(taste9);
buttonArea.add(tastePlus);
buttonArea.add(taste4);
buttonArea.add(taste5);
buttonArea.add(taste6);
buttonArea.add(tasteMinus);
buttonArea.add(taste1);
buttonArea.add(taste2);
buttonArea.add(taste3);
buttonArea.add(tasteMult);
buttonArea.add(taste0);
buttonArea.add(tastePunkt);
buttonArea.add(tasteGleich);
buttonArea.add(tasteDiv);
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public void start() {
    setLayout(new BorderLayout());
    // addWindowListener (this);
    add("North",resultat);
    add("Center",buttonArea);
}

public void actionPerformed (ActionEvent e) {
    String bLabel = (String)e.getActionCommand();

    if (jetztLoeschen) {
        if (zuletztVerwendeteMethode == "equal" && (bLabel.equals("+")
            || bLabel.equals("-") || bLabel.equals("*") ||
            bLabel.equals("/"))) {
            jetztLoeschen = false;
        } else {
            resultat.setText("");
            jetztLoeschen = false;
        }
    }

    String current = resultat.getText();

    if (bLabel.equals(".")) {
        if (!punktGesetzt) {
            punktGesetzt = true;
            resultat.setText(current+".");
        }
    } else if (bLabel.equals("+")) {
        if (zuletztVerwendeteMethode == "plus") {
            ersteZahl += Double.valueOf(current).doubleValue();
        } else if (zuletztVerwendeteMethode == "minus") {
            ersteZahl -= Double.valueOf(current).doubleValue();
        } else if (zuletztVerwendeteMethode == "mult") {
            ersteZahl *= Double.valueOf(current).doubleValue();
        } else if (zuletztVerwendeteMethode == "div") {
            ersteZahl /= Double.valueOf(current).doubleValue();
        } else {
            ersteZahl = Double.valueOf(current).doubleValue();
        }
        zuletztVerwendeteMethode = "plus";
        punktGesetzt = false;
        jetztLoeschen = true;
    } else if (bLabel.equals("-")) {
        if (zuletztVerwendeteMethode == "plus") {
            ersteZahl += Double.valueOf(current).doubleValue();
        } else if (zuletztVerwendeteMethode == "minus") {
            ersteZahl -= Double.valueOf(current).doubleValue();
        } else if (zuletztVerwendeteMethode == "mult") {
            ersteZahl *= Double.valueOf(current).doubleValue();
        } else if (zuletztVerwendeteMethode == "div") {
            ersteZahl /= Double.valueOf(current).doubleValue();
        } else {
            ersteZahl = Double.valueOf(current).doubleValue();
        }
        zuletztVerwendeteMethode = "minus";
        punktGesetzt = false;
        jetztLoeschen = true;
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        else if (bLabel.equals("*")) {
            if (zuletztVerwendeteMethode == "plus") {
                ersteZahl += Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "minus") {
                ersteZahl -= Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "mult") {
                ersteZahl *= Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "div") {
                ersteZahl /= Double.valueOf(current).doubleValue();
            } else {
                ersteZahl = Double.valueOf(current).doubleValue();
            }
            zuletztVerwendeteMethode = "mult";
            punktGesetzt = false;
            jetztLoeschen = true;
        } else if (bLabel.equals("/")) {
            if (zuletztVerwendeteMethode == "plus") {
                ersteZahl += Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "minus") {
                ersteZahl -= Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "mult") {
                ersteZahl *= Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "div") {
                ersteZahl /= Double.valueOf(current).doubleValue();
            } else {
                ersteZahl = Double.valueOf(current).doubleValue();
            }
            zuletztVerwendeteMethode = "div";
            punktGesetzt = false;
            jetztLoeschen = true;
        } else if (bLabel.equals("=")) {
            if (zuletztVerwendeteMethode == "plus") {
                ersteZahl += Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "minus") {
                ersteZahl -= Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "mult") {
                ersteZahl *= Double.valueOf(current).doubleValue();
            } else if (zuletztVerwendeteMethode == "div") {
                ersteZahl /= Double.valueOf(current).doubleValue();
            }
            resultat.setText(new Float(ersteZahl).toString());
            ersteZahl = 0;
            jetztLoeschen = true;
            zuletztVerwendeteMethode = "equal";
        } else {
            resultat.setText(current+bLabel);
        }
    }
}
```

## 1.3.6. Quiz

1. Welche Information liefert die `handleEvent()` Methode?<sup>6</sup>
  - a) Ob das Event noch modifiziert werden muss.
  - b) Ob das Event zum Container der aktuellen Komponente weitergeleitet werden soll.
  - c) Ob das Event zur Oberklasse der aktuellen Komponente weitergeleitet werden soll.
  - d) Ob das Event an die Komponente innerhalb der aktuellen Komponente weitergeleitet werden soll.
2. Sie möchten das `MouseListener` Interface implementieren. Welche der folgenden Events müssen Sie implementieren?<sup>7</sup>
  - a) `mouseEntered`
  - b) `mouseExited`
  - c) `mousePressed`
  - d) `mouseDragged`
  - e) `mouseReleased`
3. Welche Interfaces müssen Sie implementieren, falls Sie die Mausbewegung und die Mausevents abfangen wollen?<sup>8</sup>
  - a) `MouseListener`
  - b) `MouseMotionListener`
  - c) `MouseListener`, `MouseMotionListener`
  - d) `MouseListener`, `ItemListener`
4. Ein ... ist das Ergebnis einer Benutzerinteraktion mit einer AWT Komponente<sup>9</sup>
  - a) `Event`
  - b) `EventHandler`
  - c) `EventListener`
  - d) `ExceptionHandler`
5. Welche Java Konstrukte liefern Templates für die Listener Interfaces und lassen die Implementation der Methoden offen.<sup>10</sup>
  - a) `Container`
  - b) `Event Modell`
  - c) `Event Handler`
  - d) `Event Adapter`

---

<sup>6</sup> c

<sup>7</sup> a, b, c, e

<sup>8</sup> c

<sup>9</sup> a

<sup>10</sup> d

## 1.3.7. Modul Zusammenfassung

In diesem Modul haben Sie gelernt:

- wie man Events abfangen kann, die im Benutzerinterface auftreten.
- wie passende Interfaces und Handler Methoden für unterschiedliche Ereignistypen bestimmt und implementiert werden.
- wie aus dem Event Objekt Informationen extrahiert werden können.



## 1.4. Modul 3 : Die AWT Komponenten Bibliothek

### In diesem Modul

- Modul 3 : Die AWT Komponenten Bibliothek
  - Modul Einleitung
  - Lektion 1 - AWT Schlüsselkomponenten
  - Lektion 2 - Kontrolle des Aussehens der Komponenten und Drucken
  - Kreieren eines GUIs für ein Zeichenprogramm
  - Praktische Übung
  - Quiz
  - Zusammenfassung

### 1.4.1. Modul Einleitung

Wie wir in Modul 1 und 2 bereits gesehen haben, stellt AWT eine grosse Anzahl Standard Hilfsmittel zur Verfügung. Damit Ihr User Interface ansprechend oder gemäss den Richtlinien Ihrer Firma oder Ihres Kunden entspricht, müssen Sie diese Komponenten optimal und korrekt kombinieren.

In diesem Modul betrachten wir einige der Komponenten, die wir im ersten Modul aufgelistet haben. Zudem werden wir einige

neue Komponenten einführen. An Hand einiger Applets werden Sie prüfen können, wie sich einzelne der grafischen Elemente konkret verhalten (Buttons, CheckBoxen, Datei Dialoge) und wie Sie komplexere grafische Oberflächen entwickeln können.

#### 1.4.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

- wichtige Schlüsselkomponenten von AWT zu kennen
- AWT Komponenten einzusetzen, um Benutzerinterfaces für reale Programme zu entwickeln.
- das Aussehen (Fonts, Farben) mit AWT Komponenten einzusetzen

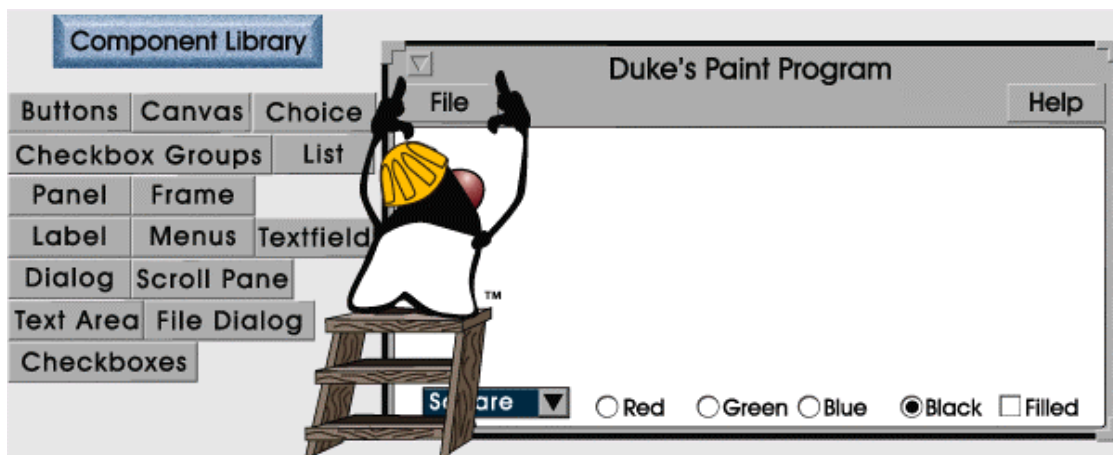
# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2. Lektion 1 - AWT Schlüsselkomponenten

### 1.4.2.1. Die AWT Komponenten Bibliothek

Java stellt eine breite Palette von Komponenten zur Verfügung, mit deren Hilfe Benutzerschnittstellen konstruiert werden können. Die folgenden Seiten präsentieren diese Komponenten in einigen Details und weisen auf Gefahren hin, die Ihnen bei der praktischen Arbeit behilflich sein können, Ihre Benutzeroberflächen zu optimieren.

Am Ende dieses Moduls werden Sie einen einfachen Rahmen für ein Zeichenprogramm schreiben. Daran arbeitet Duke in der Abbildung unten.



## 1.4.2.2. Buttons, Checkbox und Choice Komponenten

### 1.4.2.2.1. Buttons

Sie hatten bereits mehrfach Gelegenheit `Button` Komponenten kennen zu lernen. Ein Knopf / Button stellt eine einfache Schnittstelle dar, welche die zwei Zustände "gedrückt" und "nicht gedrückt" hat. Ein Button kann mit einem Label, einer einfachen Zeichenkette versehen werden, sozusagen als User Guide.

Die Grundstruktur ist aus folgendem Fragment erkennbar:

```
1  import java.awt.*;
2  import java.applet.*;
3
4  public class ButtonDemo extends Applet{
5      private Button b;
6
7      public void init() {
8          b = new Button("Textangabe");
9          this.add(b);
10     }
11 }
```

Die `actionPerformed()` Methode eines registrierten Listeners wird beim "drücken" des Knopfes (mit der Maus) ausgeführt.

Das `ActionListener` Interface wird eingesetzt, um diese Ereignisse abzufangen.

Die `getActionCommand()` Methode von `ActionEvent` (dieses Event wird ausgelöst, falls der Button gedrückt wird) liefert eine Zeichenkette, das Label des Buttons.

Mit `setActionCommand()` kann dieser Namen angepasst werden.

```
package button;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class ButtonBeispiel extends Applet implements ActionListener{
    private Button b;

    public void init() {
        b = new Button("Klick mich!");
        b.setActionCommand("Nochmal...");
        b.addActionListener(this);
        this.add(b);
    }

    public void actionPerformed(ActionEvent ae){
        System.out.println("ButtonBeispiel: der Button wurde angeklickt.");
        System.out.println("ButtonBeispiel: das Button ActionCommand ist " +
            ((Button) ae.getSource()).getActionCommand());
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## Ausgabe:

ButtonBeispiel: der Button wurde angeklickt.  
ButtonBeispiel: das Button ActionCommand ist Nochmal...  
ButtonBeispiel: der Button wurde angeklickt.  
...



### 1.4.2.2.2. Checkboxes

Die Klasse `Checkbox` stellt eine einfache "on/off" Alternative dar. Neben den Boxen kann ein Text Label platziert werden.

Das folgende Programm generiert drei Checkboxes mit den jeweiligen Labeln "eins", "zwei" und "drei".

Falls Sie ein Checkboxe anwählen, wird dies dem `ItemListener` Interface gemeldet.

Das `ItemEvent`, welches übergeben wird, enthält Informationen ob eine Checkbox ausgewählt oder abgewählt wurde.

Dieser Zustand kann mit der Methode `getStateChange()` bestimmt werden.

Die Methode liefert eine Konstante `ItemEvent.DESELECTED` oder `ItemEvent.SELECTED`.

Die Methode `getItem()` liefert eine Zeichenkette, ein `String` Objekt, welches das Label der entsprechenden Checkbox darstellt.

```
package checkbox;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class CheckboxBeispiel extends Applet {
    private Checkbox eins;
    private Checkbox zwei;
    private Checkbox drei;

    public Handler derHandler;

    public void init() {
        eins = new Checkbox("eins",true); // selektiert
        zwei = new Checkbox("zwei",false); // nicht selektiert
        drei = new Checkbox("drei",true); // selektiert

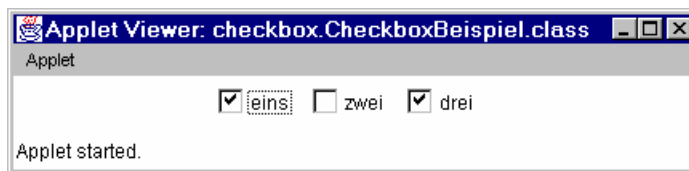
        derHandler = new Handler();

        eins.addItemListener(derHandler);
        zwei.addItemListener(derHandler);
        drei.addItemListener(derHandler);
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        this.add(eins);  
        this.add(zwei);  
        this.add(drei);  
    }  
}
```

```
class Handler implements ItemListener {  
    public void itemStateChanged(ItemEvent ev) {  
        String zustand = "deselektiert";  
  
        if (ev.getStateChange() == ItemEvent.SELECTED){  
            zustand = "selektiert";  
        }  
  
        System.out.println("CheckboxDemo: " +  
            ev.getItem() + " " + zustand);  
    }  
}
```



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.2.3. CheckboxGroup - Radio Buttons

Radio Buttons werden als Checkboxes kreiert, die zu einer Gruppe gehören. Dabei müssen Sie die CheckboxGroup als zusätzliches Argument angeben.

```
public void init() {
    cbg = new CheckboxGroup();

    eins = new Checkbox("Eins", cbg, false);
    zwei = new Checkbox("Zwei", cbg, false);
    drei = new Checkbox("Drei", cbg, true); // selektiert
}
```



An Stelle der viereckigen Multi-Selektionsboxen werden einfache Radio Buttons angezeigt.

```
package checkboxgroup;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class CheckboxGroupBeispiel extends Applet {
    private Checkbox eins;
    private Checkbox zwei;
    private Checkbox drei;

    public Handler derHandler;

    public CheckboxGroup cbg;

    public void init() {
        cbg = new CheckboxGroup();

        eins = new Checkbox("Eins", cbg, false);
        zwei = new Checkbox("Zwei", cbg, false);
        drei = new Checkbox("Drei", cbg, true);

        derHandler = new Handler();

        eins.addItemListener(derHandler);
        zwei.addItemListener(derHandler);
        drei.addItemListener(derHandler);

        this.add(eins);
        this.add(zwei);
        this.add(drei);
    }
}

class Handler implements ItemListener {
    public void itemStateChanged(ItemEvent ev) {

```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

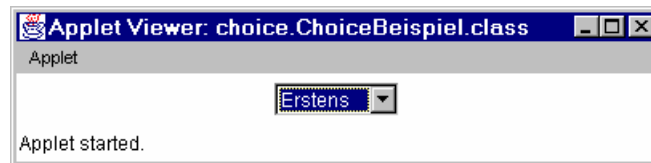
```
String zustand = "deselektiert";

if (ev.getStateChange() == ItemEvent.SELECTED){
    zustand = "selektiert";
}

System.out.println("CheckboxGroupDemo: " + ev.getItem() +
    " " + zustand);
}
}
```

## 1.4.2.2.4. Choice

Das folgende Applet zeigt die `Choice` Komponente. Diese stellt eine Liste von möglichen Werten dar.



Beim selektieren wird ein `String` Objekt ausgewählt. Dieses wird in der Demo Ausgabe angezeigt.

Das `ItemListener` Interface wird eingesetzt, um die Änderungen zu protokollieren. Im Prinzip verhält sich die Komponente wie eine `Checkbox`.

```
package choice;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class ChoiceBeispiel extends Applet {
    private Choice c;

    public Handler derHandler;

    public void init() {
        c = new Choice();

        c.addItem("Erstens");
        c.addItem("Zweitens");
        c.addItem("Drittens");

        derHandler = new Handler();

        c.addItemListener(derHandler);

        this.add(c);
    }
}

class Handler implements ItemListener {
    public void itemStateChanged(ItemEvent ev) {
        String zustand = "deselektiert";

        if (ev.getStateChange() == ItemEvent.SELECTED){
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        zustand = "selektiert";
    }

    System.out.println("Choice Beispiel: " + ev.getItem() +
        " " + zustand);
}
}
```

## 1.4.2.3. Canvas, Frame, Panel und ScrollPane

### 1.4.2.3.1. Canvas

Ein `Canvas` stellt eine einfache Fläche dar, in der Farbe der Hintergrundfarbe. Die Standardabmessungen sind null auf null. Somit müssen Sie zwingend dafür sorgen, beispielsweise mittels eines `Layout Managers`, dass die Komponente grösser als null wird.

Der `Canvas` Raum wird benutzt, um etwas zu zeichnen, zu beschreiben oder als Fläche, welche Mauseingaben oder Tastatureingaben empfangen kann. Der `Canvas` Raum kann entweder so eingesetzt werden, wie er ist, oder Sie können in die Fläche weitere Komponenten einbringen.

Der `Canvas` Raum kann auf alle Ereignisse hören, die einer generelle Komponente passieren können, beispielsweise `KeyListener`, `MouseMotionListener` oder `MouseListener` Objekte

Die Methoden dieser Listener können `KeyEvent`, `MouseMotionEvent` und `MouseEvent` Objekte empfangen und somit auch bearbeiten.

Damit ein `Canvas` Tastatur- Ereignisse empfangen und bearbeiten kann, muss die `requestFocus()` Methode des `Canvas` aufgerufen werden. Falls dies nicht geschieht, ist es kaum möglich Tastendrucke ins `Canvas` Objekt zu leiten. Sonst gehen die Tastendrucke unter Umständen einfach verloren oder irgendwohin umgeleitet.

#### **Bemerkung**

Ein `Canvas` wird oft als Plattform für eine zeichnende Komponente verwendet.

Das folgende Beispiel zeigt direkt im `Canvas` nichts an. Aber in der Ausgabe sehen Sie, ob ein Maus-Ereignis oder ein Tastatur- Ereignis eintrifft.

Dieses Beispiel verwendet die `addMouseListener` und `addKeyListener`, um die Ereignisse abzufangen und eine Ausgabe generieren zu können.

Wann immer eine Taste gedrückt wird, wird die Methode `keyTyped()` aufgerufen. Dabei wird der Buchstabe im `Canvas` angezeigt, also ein Echo generiert.

Falls die Maus angeklickt wird, wird die Methode `mouseClicked()` aufgerufen und der Fokus auf den `Canvas` gesetzt.

Alle anderen Methoden zeigen einfach den Namen der Methode auf der Konsole an, falls das entsprechende Ereignis eintritt.



```
mouseEntered
mousePressed
mouseReleased
```



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
mouseClicked
keyPressed
keyTyped
keyReleased
...
mouseExited

package canvas;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;

// Ausgabe = Methodennamen

public class CanvasBeispiel extends Applet implements KeyListener,
MouseListener {
    Canvas c;
    String s = "";

    public static void main(String args[]) {
        Frame f = new Frame("Canvas Beispiel");
        CanvasBeispiel mc = new CanvasBeispiel();
        mc.c = new Canvas();
        f.add("Center", mc.c);
        f.setSize(150, 150);
        mc.c.addMouseListener(mc);
        mc.c.addKeyListener(mc);
        f.setVisible(true);
    }

    public void keyTyped(KeyEvent ev) {
        System.out.println("keyTyped");
        s += ev.getKeyChar();
        // schlechtes Beispiel
        c.getGraphics().drawString(s, 0, 20);
    }

    public void mouseClicked(MouseEvent ev) {
        System.out.println("mouseClicked");
        // Focus auf Canvas
        c.requestFocus();
    }

    public void keyPressed(KeyEvent ev) {
        System.out.println("keyPressed");
    }

    public void keyReleased(KeyEvent ev) {
        System.out.println("keyReleased");
    }

    public void mousePressed(MouseEvent ev) {
        System.out.println("mousePressed");
    }

    public void mouseReleased(MouseEvent ev) {
        System.out.println("mouseReleased");
    }

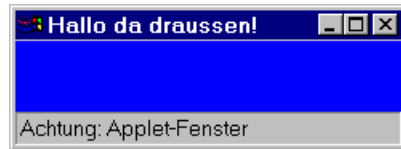
    public void mouseEntered(MouseEvent ev) {
        System.out.println("mouseEntered");
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public void mouseExited(MouseEvent ev) {  
    System.out.println("mouseExited");  
}  
}
```

## 1.4.2.3.2. Frames

Wir haben bereits im ersten Modul Frames betrachtet.



Frames sind "top-level" Windows. Zusätzlich kann ein `Frame` dekoriert werden, also mit Kopfzeile "resize Handles" versehen werden.

Die Grösse eines Frames kann mit der Methode `setSize()` gesetzt werden. In der Regel sollte Sie jedoch die Methode `pack()` einsetzen. Diese veranlasst den Layout Manager die Grösse so zu berechnen, dass alle Komponenten optimal ins `Frame` passen. Anschliessend wird die Grösse des `Frame` entsprechend gesetzt.

Die Ereignisse des `Frame` können mit Hilfe von Listener bearbeitet werden. Der `WindowListener` kann beispielsweise benutzt werden, wie im Übungsbeispiel weiter oben, die Methode `windowClosing()` aufzurufen, mit welcher der Quit Button im Window Manager Menu angewählt wird.

### Warnung

Sie sollten Keyboard Ereignisse von einem `Frame` nicht direkt auswerten. Obschon dies möglich wäre, theoretisch wenigstens, ist dies nicht zuverlässig. Wann immer Keyboard Events abgefangen werden sollen, ist es besser ein `Canvas`, `Panel` oder etwas ähnliches dem `Frame` hinzuzufügen und einen entsprechenden Listener zu verwenden.

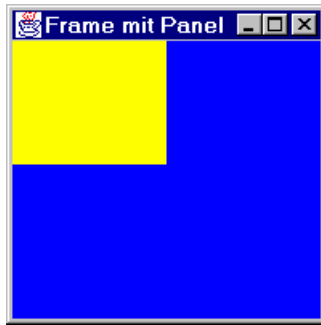
```
package einframe;  
  
import java.awt.*;  
import java.applet.*;  
  
public class MeinFrame extends Applet {  
    private Frame fr;  
    public void start ( ) {  
        fr = new Frame("Hallo da draussen!");  
        fr.setSize(500,500);  
        fr.setBackground(Color.blue);  
        fr.setVisible(true);  
    }  
    // ...  
}
```

## 1.4.2.3.3. Panels

Wir haben im Modul 1 ein einfaches `Panel` definiert und ins `Frame` eingefügt.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

Panels sind allgemein einsetzbare Container. Sie können nicht isoliert eingesetzt werden, wie beispielsweise Windows oder Dialoge.



Panels können Ereignisse behandeln, sofern der Fokus auf sie gesetzt wird.

```
package einframemitpanel;

import java.awt.*;
import java.applet.*;
public class FrameMitPanel extends Applet {
    private Frame fr;
    public void start() {
        fr = new Frame("Frame mit Panel");
        Panel pan = new Panel();
        fr.setSize(200,200);
        fr.setBackground(Color.blue);
        fr.setLayout(null);
        pan.setSize(100,100);
        pan.setBackground(Color.yellow);
        fr.add(pan);
        fr.setVisible(true);
    }
    //...
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

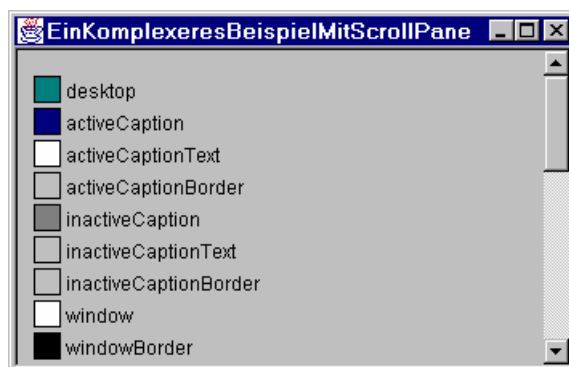
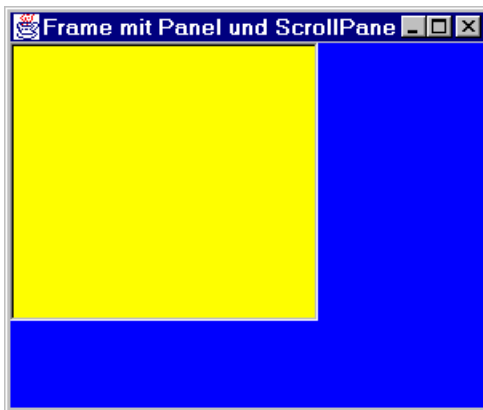
## 1.4.2.3.4. ScrollPane

Ein `ScrollPane` stellt einen allgemeinen Container zur Verfügung, der aber nicht isoliert eingesetzt werden kann. Ein `ScrollPane` stellt sozusagen ein Fenster, einen Viewport, in einen grösseren Bereich, mit Scrollbars zum Manipulieren dieses Fensters. Die `ScrollPane` kreiert und managed die Scrollbars, falls diese benötigt werden.

Ein `ScrollPane` kann lediglich eine einzige Komponente enthalten und kann den Layout Manager nicht selbst kontrollieren. Falls Sie einen Layout Manager benötigen, sollten Sie zusätzlich ein `Panel` verwenden und den Layout Manager für dieses `Panel` verwenden sowie Komponenten in dieses `Panel` plazieren.

Ereignisse werden in der Regel nicht für ein `ScrollPane` behandelt, sondern direkt auf der Stufe der Komponenten, die darin enthalten sind.

```
Frame f = new Frame("ScrollPane");
Panel p = new Panel();
ScrollPane sp = new ScrollPane();
p.setLayout(new GridLayout(3, 4));
.
.
.
sp.add(p);
f.add(sp, "Center");
f.setSize(200, 200);
f.setVisible(true);
```



## 1.4.2.4. Label, Text und List Komponenten

### 1.4.2.4.1. Label

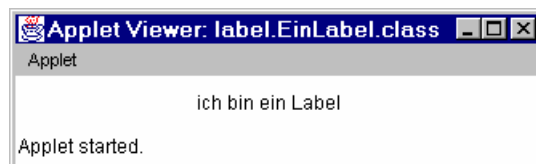
Ein `Label` Objekt zeigt eine einzige Textzeile an. Das Programm kann den Text verändern, aber der Benutzer selber nicht. Es ist auch nicht möglich den Rand des Labels zu dekorieren.

Labels werden üblicherweise eingesetzt, um etwas zu beschriften, nicht um Events abzufangen. Falls Sie den Fokus auf ein Label setzen, mit `requestFocus()`, könnten Sie auch (aber das ist kaum sinnvoll) wie beim `Canvas` Ereignisse auf ein Label umleiten.

```
import java.awt.*;
import java.applet.*;

public class LabelBeispiel extends Applet{
    private Label l;

    public void init() {
        l = new Label("ich bin ein Label");
        this.add(l);
    }
}
```



### 1.4.2.4.2. TextField

Das `TextField` ist ein einfaches Zeilentextdevice. Da lediglich eine einzelne Zeile vorhanden ist, kann mit `ActionListener` abgefragt werden, mit `actionPerformed()`, ob die Enter oder Return Taste gedrückt wurde. Andere Listener können natürlich ebenfalls eingesetzt werden. Neben dem `ActionListener` kann beispielsweise ein `TextListener` registriert werden, mit dem die einzelnen Tasten registriert werden können. Die Callback Methode hierfür ist `textValueChanged(TextEvent)`.

Wie bei der `TextArea`, die wir gleich diskutieren werden, kann der Text auf `ReadOnly` gesetzt werden, mit `setEditable(false)`.

Scrollbars werden keine angezeigt, weder horizontal noch vertikal. Allerdings kann man nach rechts scrollen, falls dies nötig wird (automatisch).

#### 1.4.2.4.2.1. TextField Beispiel ohne Listener

```
package textfield;
import java.awt.*;
import java.applet.*;

public class TextFieldBeispiel
extends Applet{
    private TextField f;
    public void init() {
        f = new TextField("TextField Demo Zeile...", 30);
        f.setEditable(true);
        f.requestFocus();
        this.add(f);
    }
}
```



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.4.2.2. *TextField Beispiel mit Listener*

```
package textlistener;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class TextFieldMitTextListener extends Applet implements
TextListener {
    private TextField f;

    public void init() {
        f = new TextField("Editierbare TextField Zeile mit
                          TextListener!", 30);
        f.setEditable(true);
        f.requestFocus();
        this.add(f);
        f.addTextListener(this);
    }
    public void textValueChanged(TextEvent te) {
        System.out.println("[TextFieldMitTextListener] Textzeile wurde
                          geändert");
    }
}
```



### Ausgabe

```
[TextFieldMitTextListener] Textzeile wurde geändert
[TextFieldMitTextListener] Textzeile wurde geändert
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.4.3. TextArea

Die `TextArea` stellt ein mehrzeiliges, mehrspaltiges Textinputdevice dar.

Falls Sie die Methode `setEditable(boolean)` verwenden, können Sie angeben, ob das Feld editierbar sein soll oder nicht.

Horizontale und vertikale Scrollbars werden automatisch eingeblendet, falls diese benötigt werden.

Sie können einen generellen Listener der `TextArea` hinzufügen. Aber weil das Textfeld aus mehreren Zeilen und Spalten besteht, können Sie lediglich feststellen, ob / dass ein neues Zeichen eingetippt wurde. Falls Sie wissen möchten, ob / dass die Eingabe beendet ist, müssen Sie einen Knopf hinzufügen, etwa "Eingabe vollständig" oder "Commit" oder sowas ähnliches.

Beide Komponenten, `TextArea` und `TextField`, sind an zwei Stellen dokumentiert: den ersten Teil finden Sie unter `TextComponent`; den zweiten Teil unter `TextArea` bzw. `TextField`. Beide sind Unterklassen der `TextComponent`.

Die Konstruktoren für die `TextArea` und `TextField` Klassen gestatten die Angabe der Anzahl Zeilen, die angezeigt werden sollen. Dies ist eine Angabe an den Layout Manager, der für die Darstellung verantwortlich ist. Daher kann die Zeilenangabe zu Problemen oder unschönem Aussehen führen, falls Sie auch noch mit Fonts und ähnlichen Klassen spielen.

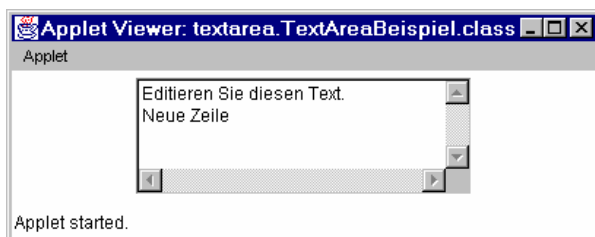
```
package textarea;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class TextAreaBeispiel extends Applet implements TextListener {
    private TextArea t;

    public void init() {
        t = new TextArea("Editieren Sie diesen Text.", 4, 30);
        t.setEditable(true);
        t.requestFocus();
        this.add(t);
        t.addTextListener(this);
    }

    public void textValueChanged(TextEvent te) {
        System.out.println("[TextAreaBeispiel] Der Text wurde verändert");
    }
}
```



```
[TextAreaBeispiel] Der Text wurde verändert
[TextAreaBeispiel] Der Text wurde verändert
[TextAreaBeispiel] Der Text wurde verändert
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.4.4. List

Ein `List` Objekt gestattet es Ihnen, Text in eine "Liste" einzutragen und zeilenweise zu selektieren. Die `List` ist scrollable und unterstützt einfache und mehrfache Selektion.

Das Beispiel zeigt deutlicher als aller Text, um was es geht:



Dem Konstruktor kann die bevorzugte Anzahl Zeilen mitgegeben werden, welche angezeigt werden soll.

Ein `ActionEvent`, vom `ActionListener` Interface abgefangen.

```
package list;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class ListBeispiel extends Applet {
    private List l;

    public void init() {
        l = new List(4,true);
        l.add("Erste");
        l.add("Zweite");
        l.add("Dritte");
        l.add("Vierte");
        l.add("Fünfte");
        l.add("Sechste");
        this.add(l);
    }
}
```

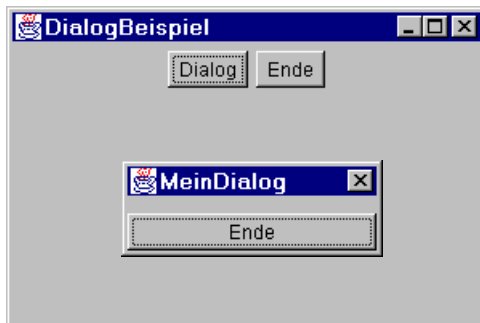


## 1.4.2.5. Dialog und FileDialog Komponenten

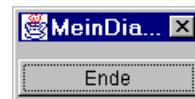
### 1.4.2.5.1. Dialog

Eine `Dialog` Komponente ist ähnlich wie ein `Frame`, kann also allein eingesetzt werden, mit einem `Window` plus Dekoration.

Die Komponente unterscheidet sich von einem `Frame` insofern, als weniger Dekorationen bereits mitgeliefert werden. Sie können auch "modale" Dialoge zulassen, wie in diesem Beispiel:



Sie können das Dialog Fenster auch aus dem Rahmen herausziehen.



Dialoge werden normalerweise nicht gleich dem Benutzer sichtbargemacht, sondern erst nachdem der Benutzer irgend eine Aktion ausgeführt hat. Im Beispiel oben muss der Benutzer auf die `Dialog` Taste Drücken, um das Dialogfenster zu kreieren. Dies geschieht in der Regel mit einem Konstrukt von der Form:

```
public void  
    actionPerformed(ActionEvent ev) {  
        d.setVisible(true);  
    }
```

Dialog sollten als wiederverwendbare Devices angesehen werden. Mit andern Worten: falls Sie ein Dialogobjekt nicht mehr benötigen, sollten Sie es einfach eventuell unsichtbar stehen lassen. Falls Sie es echt nicht mehr benötigen, wird sich der Garbage Collector darum kümmern; sollten Sie es wieder benötigen, steht es Ihnen sehr schnell wieder zur Verfügung. Denn das Kreieren eines Dialoge benötigt seine Zeit!

Einen `Dialog` machen Sie mit `setVisible(false)` unsichtbar. Dies geschieht typischerweise mit einem `WindowListener`. Falls die `windowClosing()` Method des Listeners aufgerufen wird, können Sie den Dialog schliessen. Das geschieht analog wie beim `Frame`.

### **Bemerkung**

Die `listener` Klasse kann den `WindowAdapter` erweitern oder den `WindowListener` implementieren.

Das Beispiel oben ist etwas umfangreicher als sonst, und eine Applikation. Deswegen finden Sie den Programmcode auf der folgenden Seite.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.5.1.1. Dialog Beispiel Anwendung

```
package dialog;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

// Dialog Fenster
class MeinDialog extends Dialog implements ActionListener {
    public MeinDialog(Frame parent) { // Konstruktor
        super(parent, "MeinDialog", true);
        Point parloc = parent.getLocation();
        setBounds(parloc.x + 30, parloc.y + 30, 400, 300);
        setBackground(Color.lightGray);
        setLayout(new BorderLayout());
        //Panel
        Panel panel = new Panel();
        customizeLayout(panel);
        add("Center", panel);
        //Ende-Button
        Button button = new Button("Ende"); // Dialog schliessen
        button.addActionListener(this);
        add("South", button);
        //Window-Listener
        addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent event) {
                    endDialog(); // siehe weiter unten
                }
            );
        pack();
    }

    private void customizeLayout(Panel panel) {
        //Beispielcode hier
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getActionCommand().equals("Ende")) {
            endDialog();
        }
    }

    void endDialog() {
        setVisible(false);
        dispose();
        ((Window)getParent()).toFront();
        getParent().requestFocus();
    }
}

public class DialogBeispiel extends Frame implements ActionListener {
    public static void main(String[] args) {
        DialogBeispiel wnd = new DialogBeispiel();
        wnd.setSize(300, 200);
        wnd.setVisible(true);
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public DialogBeispiel() { // Konstruktor
    super("DialogBeispiel");
    setBackground(Color.lightGray);
    setLayout(new FlowLayout());
    //Dialog-Button
    Button button = new Button("Dialog");
    button.addActionListener(this);
    add(button);
    //Ende-Button
    button = new Button("Ende");
    button.addActionListener(this);
    add(button);
    //Window-Listener
    addWindowListener(
        new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                setVisible(false);
                dispose();
                System.exit(0);
            }
        }
    );
}

public void actionPerformed(ActionEvent event) {
    String cmd = event.getActionCommand();
    if (cmd.equals("Dialog")) {
        MeinDialog dlg = new MeinDialog(this);
        dlg.setVisible(true);
    } else if (cmd.equals("Ende")) {
        setVisible(false);
        dispose();
        System.exit(0);
    }
}
```

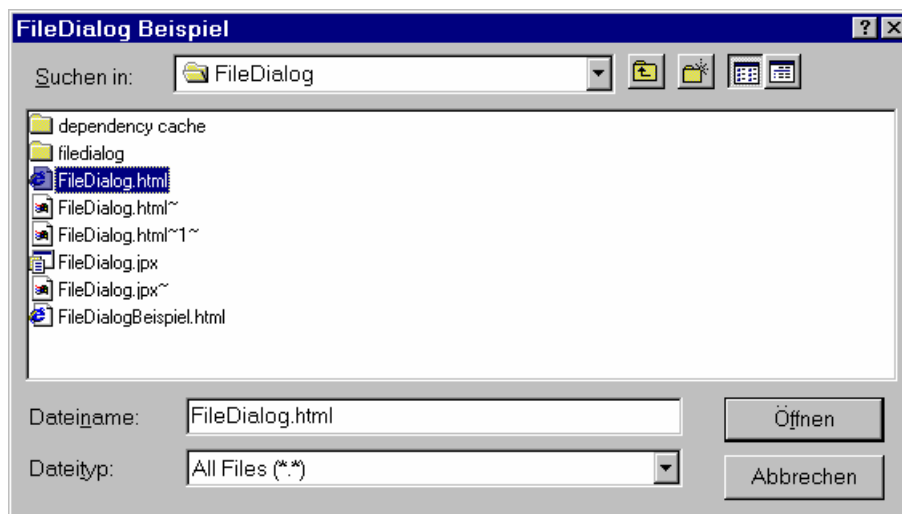
# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.5.2. FileDialog

Der `FileDialog` implementiert ein Device, mit dem Dateien ausgewählt werden können. Die Komponente besitzt ein eigenes Fenster, mit Dekoration und gestatte dem Benutzer das Dateisystem anzuschauen und bestimmte Dateien auszuwählen.

Events werden normalerweise nicht abgefangen.

Mit `setVisible(true)` wird blockiert, bis der Benutzer ein OK bestätigt und einen Dateinamen auswählt. Witzig ist, dass im Ausführungsprotokoll erst dann die Anzeige quittiert wird, wenn das Dialogfenster wieder geschlossen wird.



```
package filedialog;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class FileDialogBeispiel extends Applet{
    private Frame f;
    private FileDialog d;

    public void init() {
        f = new Frame();
        d = new FileDialog(f, "FileDialog Beispiel");
    }

    public void start() {
        d.setVisible(true);
        System.out.println("[FileDialogBeispiel]
                           FileDialog wird angezeigt");
    }

    public void stop() {
        d.setVisible(false);
        System.out.println("[FileDialogBeispiel]
                           FileDialog wird versteckt");
    }
}
```

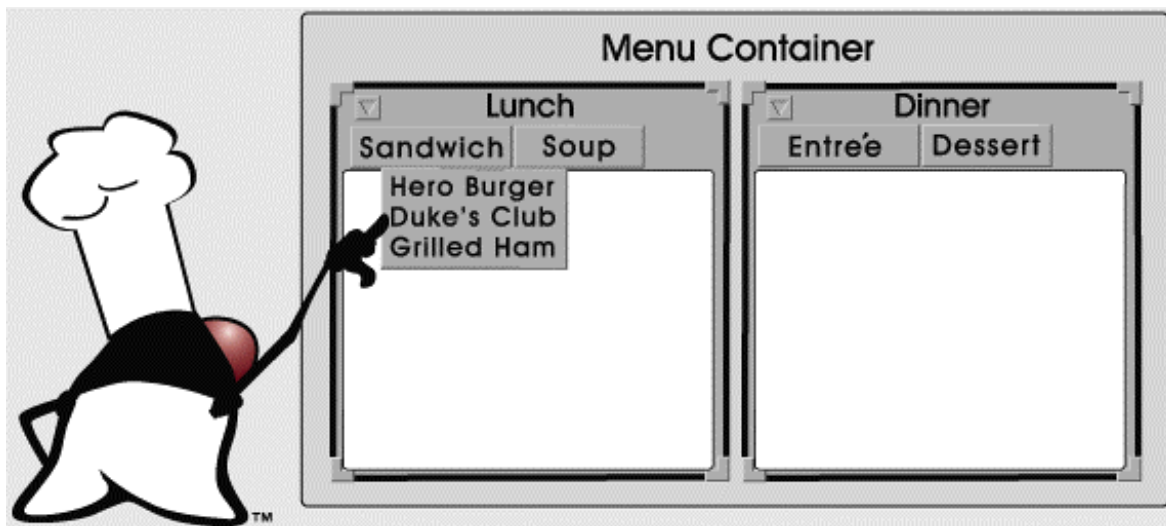
## 1.4.2.6. Menü Komponenten

### 1.4.2.6.1. Menus

Menus unterscheiden sich von anderen Komponenten in mehrer Beziehung und zwar grundlegend. Menps können andern Containern nicht einfach hinzugefügt werden. Auch das Layout geschieht nicht einfach durch den Layout Manager, wie bei den andern Komponenten. Menüs können Sie lediglich zu speziellen, den *Menu Containern* hinzufügen.

Sie können einen Menübaum nur starten, indem Sie eine Menübar in ein Frame einfügen, mit Hilfe der `setMenuBar()` Methode. Ab dann können Sie Menüs zur Menübar hinzufügen und Menüs zu Menüs oder Menüitems zu Menüs hinzufügen.

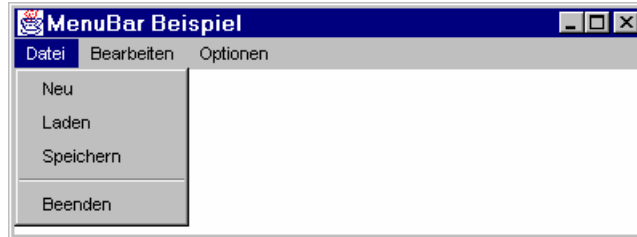
Pop-up Menus sind eine Ausnahme und benötigen keinen Layout.



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.6.2. MenuBar

Eine `MenuBar` Komponente ist ein horizontales Menu. Sie kann lediglich einem `Frame` Objekt hinzugefügt werden und bildet die Wurzel für alle Menübäume.



Die `MenuBar` Komponente selbst kennt keine Listener. Der Grund ist sehr einfach der, dass man davon ausgehen kann, dass das Verhalten durch die Menüs bestimmt wird.

Ein Spezialfall ist das Hilfsmenü. Dieses können Sie speziell mit der `MenuBar` Methode `setHelpMenu(Menu)` speziell gekennzeichnet werden. Das Menü selbst wird einfach dem `MenuBar` hinzugefügt, aber auf Plattform spezifische Art und Weise. Im Falle von X/Motif steht ein Hilfsmenü automatisch rechts im Menübar. Unter Windows sieht es folgendermassen aus:



Sie können eine `MenuBar` Komponente entweder direkt instanzieren:

```
1     ....
2     Frame f = new Frame("MenuBar");
3     MenuBar mb = new MenuBar();
4     f.setMenuBar(mb);
5     ....
```

oder aber eine Klasse definieren, welche diese Klasse erweitert:

```
class Hauptmenu extends MenuBar {
    ...

    public Hauptmenu()    {
        Menu m;

        //Datei
        m = new Menu("Datei");
        m.add(new MenuItem("Neu"));
        ...
    }
}
```

Wir werden auf ein vollständiges Beispiel zurück kommen, sobald wir weitere Menükomponenten besprochen haben.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.6.3. Menu

Die `Menu` Klasse stellt die üblichen pull-down Menus zur Verfügung. Die Komponente kann entweder zu einer `MenuBar` oder zu einem anderen `Menu` hinzugefügt werden.



Sie könnten auch einen `ActionListener` einem `Menu` Objekt hinzufügen. Aber das ist eher unüblich. Normalerweise wird der Dialog durch die `MenuItems` bestimmt, die Sie anwählen können und auf Grund deren Auswahl bestimmte Aktivitäten ausgeführt werden.

```
public Hauptmenu()    {
    Menu m;

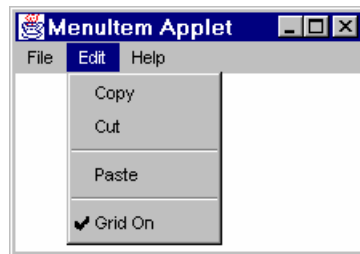
    //Datei
    m = new Menu("Datei");
    m.add(new MenuItem("Neu"));
    m.add(new MenuItem("Laden"));
    m.add(new MenuItem("Speichern"));
    m.addSeparator();
    m.add(new MenuItem("Beenden"));
    add(m);
    //Bearbeiten
    m = new Menu("Bearbeiten");
    m.add((miRueck = new MenuItem("Rueckgaengig")));
    m.addSeparator();
    m.add(new MenuItem("Ausschneiden"));
    m.add(new MenuItem("Kopieren"));
    m.add(new MenuItem("Einfuegen"));
    m.add(new MenuItem("Loeschen"));
    add(m);
    //Optionen
    m = new Menu("Optionen");
    m.add(new MenuItem("Einstellungen"));
    m.add((miFarbe = new CheckboxMenuItem("Farbe")));
    add(m);
    //Rueckgaengig deaktivieren
    enableRueckgaengig(false);
    //Farbe anschalten
    setFarbe(true);
    //Hilfe
    m = new Menu("Hilfe");
    m.add(new MenuItem("About"));
    this.setHelpMenu(m);
    add(m);
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.6.4. MenuItem

Die einzelnen Menüpositionen (auch für Applets) innerhalb eines Menüs werden der Reihe nach durch Sie vorgegeben.

Schauen wir uns das Ganze an Hand eines MenuApplet an:



mit dem Programmcode:

```
package menuitem;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class MenuItemApplet extends Applet implements ActionListener{
    private Frame f;
    private MenuBar mb;
    private Menu m1;
    private Menu m2;
    private Menu m3;
    private MenuItem mi1;
    private MenuItem mi2;
    private MenuItem mi3;
    private MenuItem mi4;
    private MenuItem mi5;
    private MenuItem mi6;
    private MenuItem mi7;
    private CheckboxMenuItem cbmi1;

    public void init() {
        f = new Frame("MenuItem Applet");
        mb = new MenuBar();
        f.setMenuBar(mb);

        m1 = new Menu("File");
        m2 = new Menu("Edit");
        m3 = new Menu("Help");
        mb.add(m1);
        mb.add(m2);
        mb.add(m3);
        mb.setHelpMenu(m3);

        mi1 = new MenuItem("Save");
        mi2 = new MenuItem("Load");
        mi3 = new MenuItem("Quit");
```



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
mi4 = new MenuItem("Copy");
mi5 = new MenuItem("Cut");
mi6 = new MenuItem("Paste");
mi7 = new MenuItem("About");

mi1.addActionListener(this);
mi2.addActionListener(this);
mi3.addActionListener(this);
mi4.addActionListener(this);
mi5.addActionListener(this);
mi6.addActionListener(this);
mi7.addActionListener(this);

cbmil = new CheckboxMenuItem("Grid On");

m1.add(mi1);
m1.add(mi2);
m1.addSeparator();
m1.add(mi3);

m2.add(mi4);
m2.add(mi5);
m2.addSeparator();
m2.add(mi6);
m2.addSeparator();
m2.add(cbmil);

m3.add(mi7);

f.setSize(150,150);
}

public void start() {
    f.setVisible(true);
}

public void stop() {
    f.setVisible(false);
}

public void actionPerformed(ActionEvent ae){
    System.out.println("[MenuItemApplet] " +
        ae.getActionCommand() + " selektiert.");
}
}
```

und der Protokollausgabe:

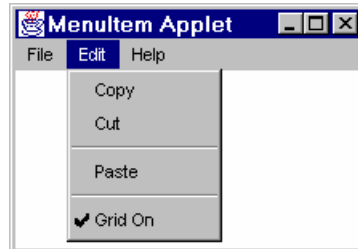
```
[MenuItemApplet] About selektiert.
[MenuItemApplet] Copy selektiert.
[MenuItemApplet] Save selektiert.
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.6.5. CheckboxMenuItem

Ein `CheckboxMenuItem` ist eine selektierbare Menüposition mit zwei möglichen Werten:

- Selektionen - *on* oder *off*
- Choices - aufgelistet in einem Menü



Die aktuelle Auswahl wird mit dem `ItemListener` Interface überprüft. Die Methode `itemStateChanged()` wird jeweils aufgerufen, falls die Wahl geändert wird.

```
public void init() {
    f = new Frame("MenuItem Applet");
    mb = new MenuBar();
    f.setMenuBar(mb);

    ...

    m2 = new Menu("Edit");
    ...

    mb.add(m2);
    mb.add(m3);
    mb.setHelpMenu(m3);

    mi1 = new MenuItem("Save"); // mi = menu item
    ...

    mi7 = new MenuItem("About");

    mi1.addActionListener(this);
    ...

    mi7.addActionListener(this);

    cbmi1 = new CheckboxMenuItem("Grid On");

    m1.add(mi1);
    m1.add(mi2);
    m1.addSeparator();
    m1.add(mi3);

    m2.add(mi4);
    m2.add(mi5);
    m2.addSeparator();
    m2.add(mi6);
    m2.addSeparator();
    m2.add(cbmi1);

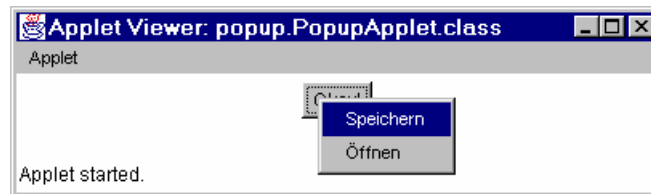
    m3.add(mi7);
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.2.6.6. PopupMenu

Die `PopupMenu` Komponente liefert ein standalone Menu, welches auf jeder Komponente geöffnet werden kann. Sie können Menu Items oder Menüs zu einem Popup Menü hinzufügen.

Das `PopupMenu` *muss* einer Oberkomponente hinzugefügt werden. Dies kann beispielsweise ein Applet sein.



Das `PopupMenu` wird mit Hilfe der `show` Methode angezeigt, typischerweise getriggert durch einen Button (wie im Beispiel oben).

Die Originalkomponente, der das Popup angehängt wird, muss in der selben Container Komponente enthalten sein wie das PopUp selbst. Im obigen Beispiel sind der Button und das PopUp beide im Applet (`this`) enthalten.

```
package popup;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class PopupApplet extends Applet implements ActionListener{
    private Button b;
    private PopupMenu p;
    private MenuItem s;
    private MenuItem l;

    public void init() {
        b = new Button("Okay!");
        p = new PopupMenu("Popup");
        s = new MenuItem("Speichern");
        l = new MenuItem("Öffnen");
        b.addActionListener(this);
        this.add(b);
        p.add(s);
        p.add(l);
        this.add(p);
    }

    public void actionPerformed(ActionEvent ev) {
        p.show(b,10,10);
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.3. Lektion 2 - Kontrolle des Aussehens der Komponenten und Drucken

### 1.4.3.1. Kontrolle der Farbe der Komponenten

Sie können die Farbe und Aussehen der AWT Komponenten mittels Hintergrund und Vordergrund Farbe kontrollieren, ebenso beim Text, auch dessen Font!

AWT stellt Ihnen dafür zwei Methoden für Komponenten zur Verfügung:

- `setForeground()`
- `setBackground()`

Beide Methoden benutzen als Argumente jeweils Instanzen der Klasse `java.awt.Color`. Sie können mit den Konstanten, beispielweise `Color.red`, `Color.blue`, etc. arbeiten. Die Klasse `java.awt.Color` kennt dreizehn vordefinierte Farben:

*black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white und yellow.*

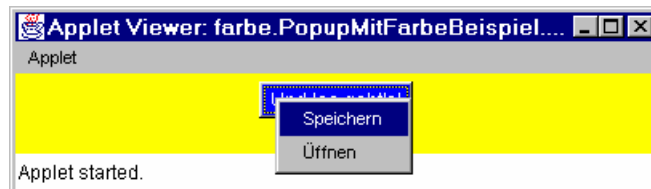
Aber Sie können auch selber weitere Farben definieren:

```
int r = 255, g = 255, b = 0;
Color c = new Color(r, g, b);
```

Dieser Konstruktor kreiert Farben basierend auf rot/grün/blau jeweils im Bereich 0 bis 255. Beachten Sie aber, dass Ihr Ausgabemedium eventuell eine der so konstruierten Farben nicht kennt.

### 1.4.3.2. Verändern der Komponentenfarbe - Beispiel

Hier ein (weiteres) Beispiel für eine modifizierte Version des `PopupMenu` Beispiels von weiter oben, einfach mit etwas Farbe



Hintergrund- und Vordergrundfarben werden hier mit `setForeground()` und `setBackground()` Methoden gesetzt.

```
package farbe;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/**
 * @author J.M.Joller
 * @version 1.0
 */

public class PopupMitFarbeBeispiel extends Applet implements
ActionListener{
    private Button b;
    private PopupMenu p;
    private MenuItem s;
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
private MenuItem l;  
  
public void init() {  
    b = new Button("Und los geht's!");  
    p = new PopupMenu("Popup");  
    s = new MenuItem("Speichern");  
    l = new MenuItem("Öffnen");  
    b.setForeground(Color.white);  
    b.setBackground(Color.blue);  
    b.addActionListener(this);  
    this.add(b);  
    this.setForeground(Color.black);  
    this.setBackground(Color.yellow);  
    p.add(s);  
    p.add(l);  
    this.add(p);  
}  
  
public void actionPerformed(ActionEvent ev) {  
    p.show(b,10,10);  
}  
}
```

### 1.4.3.3. Ändern der Fonts der Komponenten

Mit der Methode `setFont()` können Sie Charakteristiken der Fonts in Texten der Komponenten verändern. Auch hier gilt die selbe Warnung wie eben: passen Sie auf, dass das Ausgabemedium die Fonts auch tatsächlich unterstützt.

Die Klasse, mit der Fonts beschrieben werden, ist `java.awt.Font`. In der Klasse selbst sind keine Konstanten definiert für die einzelnen Fonts. Sie geben diese einfach als Text ein inklusive Grösse und weiteren Charakteristiken:

```
Font f = new Font("TimesRoman", Font.PLAIN, 14);
```

Gültige Fonts sind:

- Dialog
- Helvetica
- TimesRoman - ohne Zwischenraum
- Courier

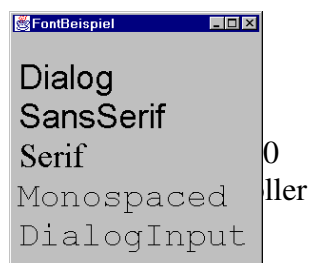
Eine vollständige Liste erhalten Sie mit der Methode `getFontList()` angewandt auf ein Toolkit Objekt (siehe Beispiel). Das aktuelle Toolkit erhalten Sie nachdem eine Komponente angezeigt wurde, mit der Methode `getToolkit()`. Alternativ können Sie das Standard Toolkit verwenden:

```
Toolkit.getDefaultToolkit().
```

Font Style Konstanten, eigentlich `int` Werte, sind:

- `Font.BOLD`
- `Font.ITALIC`
- `Font.PLAIN`
- `Font.BOLD + Font.ITALIC`

Die Grösse der Zeichen wird in Punkten / Points angegeben, als `int` Werte.



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.3.4. Drucken in AWT

Das Drucken in AWT geschieht analog zur Anzeige auf dem Bildschirm. Es werden spezielle `java.awt.Graphics` Objekte definiert, welche die Zeichenbefehle an den Drucker (Treiber) senden.

Wenn Sie ein solches Objekt drucken, erscheint der übliche Druckerdialog und Sie können den Drucker auswählen.

Pro Seite wird ein grafisches Objekt mit `f.printComponents(g);` ausgegeben.

```
1     ....
2     Frame f = new Frame("Print Beispiel");
3     Toolkit t = f.getToolkit();
4     PrintJob job = t.getPrintJob(f, "MeinPrintJob", null);
5     Graphics g = job.getGraphics();
6     ....
```

Ein komplexeres Beispiel sehen Sie hier:

```
package print;
import java.awt.*;
import java.awt.event.*;

public class PrintBeispiel extends Frame {
    public static void main(String[] args) {
        PrintBeispiel wnd = new PrintBeispiel();
    }

    public PrintBeispiel() {
        super("PrintBeispiel");
        addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent event)
                { System.exit(0); }
            }
        );
        setBackground(Color.lightGray);
        setSize(400,400);
        setVisible(true);
        //Ausdruck in 2 Sekunden starten
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            //nichts
        }
        printTestPage();
    }

    public void paint(Graphics g){
        paintGrayBoxes(g, 40, 50);
    }

    public void printTestPage() {
        PrintJob pjob = getToolkit().getPrintJob(
            this,
            "Testseite",
            null
        );
        if (pjob != null) {
            //Metriken
            int pres = pjob.getPageResolution();
            int sres = getToolkit().getScreenResolution();
            Dimension d2 = new Dimension(
                (int)((21.0 - 2.0) / 2.54) * sres,
                (int)((29.7 - 2.0) / 2.54) * sres
            );
        }
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
);
//Ausdruck beginnt
Graphics pg = pjob.getGraphics();
if (pg != null) {
    //Rahmen
    pg.drawRect(0, 0, d2.width, d2.height);
    //Text
    pg.setFont(new Font("TimesRoman",Font.PLAIN,24));
    pg.drawString("Testseite",40,70);
    pg.drawString(
        "Druckerauflösung : " + pres + " dpi",
        40,
        100
    );
    pg.drawString(
        "Bildschirmauflösung : " + sres + " dpi",
        40,
        130
    );
    pg.drawString(
        "Seitengröße : " + d2.width + " * " + d2.height,
        40,
        160
    );
    //Graustufenkästchen
    paintGrayBoxes(pg, 40, 200);
    //Seite ausgeben
    pg.dispose();
}
pjob.end();
}
}

private void paintGrayBoxes(Graphics g, int x, int y)
{
    for (int i = 0; i < 16; ++i) {
        for (int j = 0; j < 16; ++j) {
            int level = 16 * i + j;
            g.setColor(Color.black);
            g.drawRect(x + 20 * j, y + 20 * i, 20, 20);
            g.setColor(new Color(level, level, level));
            g.fillRect(x + 1 + 20 * j, y + 1 + 20 * i, 19, 19);
        }
    }
}
}
```

Die Ausgabe wurde in den PDF Writer umgeleitet und befindet sich auf dem Server.  
Wenn Sie die `print()` Methode aufrufen, dann fragt diese die Komponente ab und druckt sie.  
Mit der `printAll()` Methode wird der Container und alle Komponenten darin veranlasst, sich auf dem Drucker darzustellen, also zu drucken.

Mit `dispose()` wird die Seite an den Drucker gesandt.

```
g.dispose();
```

Und schliesslich wird das Drucken des Objekts mit der `end()` Methode abgeschlossen. Damit wird der Druckjob an den Spooler übergeben.

```
job.end();

1    ....
2    Frame f = new Frame("Print Beispiel");
3    Toolkit t = f.getToolkit();
4    PrintJob job = t.getPrintJob(f, "MeinPrintJob", null);
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
5    Graphics g = job.getGraphics();  
6    f.printAll(g);  
7    ....
```



## 1.4.4. Die *paint ()* Methode und der *Graphics* Context

Wir sind bereits verschiedentlich auf die `paint()` Methode getroffen. In der Regel werden die Inhalte eines Applets mit Hilfe der `paint()` Methode auf den Bildschirm gezeichnet:

```
package einfuehrendesbeispiel;

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

public class EinfacherKreis extends Applet {
    boolean isStandalone = false;
    /**Get a parameter value*/
    public String getParameter(String key, String def) {
        return isStandalone ? System.getProperty(key, def) :
            (getParameter(key) != null ? getParameter(key) : def);
    }

    /**Construct the applet*/
    public EinfacherKreis() {
    }
    /**Initialize the applet*/
    public void init() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    /**Component initialization*/
    private void jbInit() throws Exception {
        // hier passiert gar nichts
    }
    /**Get Applet information*/
    public String getAppletInfo() {
        return "Applet Information";
    }
    /**Get parameter info*/
    public String[][] getParameterInfo() {
        return null;
    }
    public void paint(Graphics g) {
        g.setColor(Color.black);
        g.fillRect(0,0,300,300);
        g.setColor(Color.white);
        g.fillOval(30,30,50,50);
    }
}
```

Wichtig bei der `paint()` Methode ist zu wissen, dass die `paint()` Methode lediglich zur Verfügung gestellt wird. Sie sehen keinen Methodenaufruf von `paint()` im obigen Programm! Der exakte Zeitpunkt, zu dem die `paint()` Methode aufgerufen wird, können Sie selber kaum bestimmen, denn dafür ist ein `GUIThread` zuständig.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

Das Zeichnen geschieht dadurch, dass Methoden eines *Graphics-Contextes* aufgerufen werden. Grapische Kontexte können auf

- Components:
  - Applet
  - Canvas
  - Frame
  - Panel
- Images
- Printers

dargestellt werden. Wir werden uns mit der Darstellung eines graphischen Contextes auf Komponenten, und am Schluss auf/in Bildern darstellen.

Der Stoff gehört weder zur Certification noch wird er üblicherweise im Unterricht durchgenommen.

Die drei Grundoperationen der Graphics Klasse sind:

- Auswahl einer Farbe
- Auswahl eines Fonts
- Zeichnen und Ausfüllen
- Clipping (eingeschränken der Region eines grafischen Kontextes, welche verändert werden kann).

## 1.4.4.1. Setzen der Farbe

Zum Setzen einer Farbe wird die `setColor()` Methode verwendet. Als Argument können Sie `r,g,b` oder eine Instanz der `Color` Klasse verwenden. Es gibt insgesamt 13 vordefinierte Farbklassen, als `final` definiert.

```
g.setColor(Color.black);  
...  
g.setColor(Color.white);  
...
```

Die Definition einer eigenen Farbe geschieht mit `(r,g,b)` Levels:

```
Color(int redlevel, int greenlevel, int bluelevel)
```

Beispiele:

```
new Color(0,0,0);           // Color.black  
new Color(255,0,0);         // Color.red  
new Color(255,255,255);     // Color.white
```

Sie können die verschiedenen Farben leicht in einem Applet definieren und sich ansehen. Nachdem die Farbe gesetzt wurde, werden alle weiteren Objekte in dieser Farbe gezeichnet!

## 1.4.4.2. Fonts - Die Auswahl eines Fonts

Die Definition eines Fonts geschieht analog zur Definition einer Farbe. Die Fonts müssen jedoch definiert sein, bevor Sie diese anwenden können.

Der Konstruktor:

```
Font(String fontname, int style, int size)
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

Die verfügbaren Fonts sind Plattform- abhängig. Sie können sich jedoch eine Liste generieren lassen, indem Sie die Fontliste abfragen:



```
package einführendesbeispiel;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;

public class FontListe extends Applet {
    boolean isStandalone = false;
    /**Get a parameter value*/
    public String getParameter(String key, String def) {
        return isStandalone ? System.getProperty(key, def) :
            (getParameter(key) != null ? getParameter(key) : def);
    }

    /**Construct the applet*/
    public FontListe() {
    }
    /**Initialize the applet*/
    public void init() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    /**Component initialization*/
    private void jbInit() throws Exception {
    }
    /**Get Applet information*/
    public String getAppletInfo() {
        return "Applet Information";
    }
    /**Get parameter info*/
    public String[][] getParameterInfo() {
        return null;
    }
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
public void paint(Graphics g) {
    String fontnamen[] = Toolkit.getDefaultToolkit().getFontList();
    System.out.println("Fontliste:");
    for (int i=0; i<fontnamen.length; i++) {
        g.setFont(new Font(fontnamen[i], Font.BOLD, 25));
        g.drawString(fontnamen[i], 50, 50+i*25);
    }
}
```

Die üblicherweise durch Java aufgezwungenen Fonts sind:

- "Serif"
- "SansSerif"
- "Monospaced"

(oder "TimesNewRoman", "Helvetica", "Courier").

Sie können auch Style Kombination kreieren, indem Sie beispielsweise

`Font.BOLD + Font.ITALIC`

kombinieren.

## 1.4.4.3. Zeichnen und Malen von Flächen

Der Nullpunkt jeder Zeichnung ist ganz oben links; die x-Achse zeigt nach rechts, y nach unten.

Das einfache Graphics Objekt besitzt nur eine limitierte Anzahl möglicher geometrischer Konstrukte. Weitere finden Sie beispielsweise unter Java2D oder als Graphics2D Objekt.

Die Grundfunktionen von Graphics Objekten sind:

- `drawLine()`
- `drawRect()` und `fillRect()`
- `drawOval()` und `fillOval()`
- `drawArc()` und `fillArc()`
- `drawPolygon()` und `fillPolygon()`
- `drawPolyline()`
- `drawString()`
- `drawImage()`

Schauen wir uns die einzelnen Methoden kurz an.

### 1.4.4.3.1. `drawLine()`

Diese Methode zeichnet einfach eine Verbindung zwischen zwei Punkten. Die Farbe wird aus der bereits gesetzten Graphics Contextfarbe übernommen:

```
g.drawLine(10,10,50,50);
```

### 1.4.4.3.2. `drawRectangle()` und `fillRectangle()`; `draw3Drect()`, `drawRoundRect()`

Mit diesen zwei Methoden können Sie ein Rechteck, leer oder gefüllt malen. Dabei wird der Nullpunkt und die Höhe plus Breite (nicht zwei entgegengesetzte Punkte!) angegeben.

```
g.drawRect(xStart,yStart,iWidth,iHeight)
```

## 1.4.4.3.3. drawOval() und fillOval()

Der Unterschied zum Rechteck ist die Darstellung. Parameter und Aufruf sind identisch, auch die Bedeutung der Parameter!

```
g.drawOval(xStart,yStart,iWidth,iHeight)
```

## 1.4.4.3.4. drawArc() und fillArc()

Ein Arc ist ein Segment eines Ovals. Dabei müssen Sie besonders auf die Parameter achten: zuerst müssen Sie die Parameter für das Oval angeben (Nullpunkt, Höhe, Breite), dann die Winkelangaben (Startwinkel und wieviele Grade). Der Winkel zählt ab Richtung Ost / drei Uhr, also nicht im geografischen Sinne.

## 1.4.4.3.5. drawPolygon() und fillPolygon()

Ein Polygon ist eine geschlossene geometrische Figur. Als Parameter verwenden die Methoden zwei Arrays, mit den x und den y Koordinaten. Der dritte Parameter ist die Anzahl Punkte. Warum der dritte Parameter?

Falls die Dimension der Koordinatenfelder nicht mit dem letzten Parameter, der Anzahl Winkel übereinstimmt, wird eine `ArrayIndexOutOfBoundsException` geworfen. Man kann den Parameter also nicht verstehen.

## 1.4.4.3.6. drawPolyline()

Zeichnet ein nicht geschlossenes Polygon.

## 1.4.4.3.7. drawString()

Zeichnet Ihnen die Zeichenkette auf die Komponente, auf der das Graphics Objekt definiert ist.

Beispiele sehen Sie weiter oben, beispielsweise die Ausgabe der Fonts.

## 1.4.4.3.8. drawImage()

Fügt ein Bild dem grafischen Context hinzu. Allerdings benötigen Sie neben dem Bild selbst noch einen `ImageObserver`.

```
package einführendesbeispiel;

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

public class drawImage extends JApplet {
    boolean isStandalone = false;
    Image im_Bild;
    int i_MouseX=10, i_MouseY=10;
    /**Get a parameter value*/
    public String getParameter(String key, String def) {
        return isStandalone ? System.getProperty(key, def) :
            (getParameter(key) != null ? getParameter(key) : def);
    }

    /**Construct the applet*/
    public drawImage() {
    }
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
/**Initialize the applet*/
public void init() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

/**Component initialization*/
private void jbInit() throws Exception {
    this.setSize(new Dimension(400,300));
    this.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            this_mouseClicked(e);
        }
    });

    // Laden des Bildes
    im_Bild = Toolkit.getDefaultToolkit().createImage("island1.jpg");
}

/**Get Applet information*/
public String getAppletInfo() {
    return "Applet Information";
}

/**Get parameter info*/
public String[][] getParameterInfo() {
    return null;
}

//static initializer for setting look & feel
static {
    try {

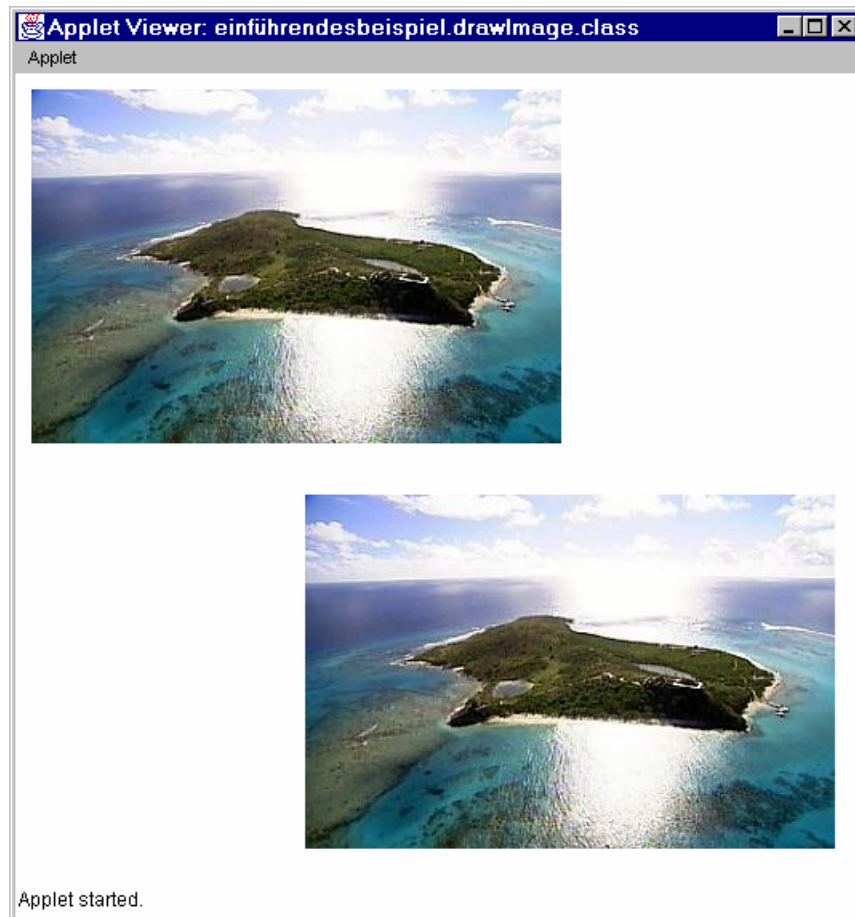
//UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

//UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName()
);
    }
    catch(Exception e) {
    }
}

public void paint(Graphics g) {
    g.drawImage(im_Bild,i_MouseX,i_MouseY,Color.blue,this);
}

void this_mouseClicked(MouseEvent e) {
    i_MouseX=e.getX();
    i_MouseY=e.getY();
    repaint();
    //return true;
}
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN



Das Bild wird immer ab dem Maus Klickpunkt neu gezeichnet, wobei das alte Bild nicht gelöscht wird (`repaint()`). Falls Sie das Applet verstecken und anschliessend wieder hervorholen, wird nur noch die neue Version gezeichnet.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

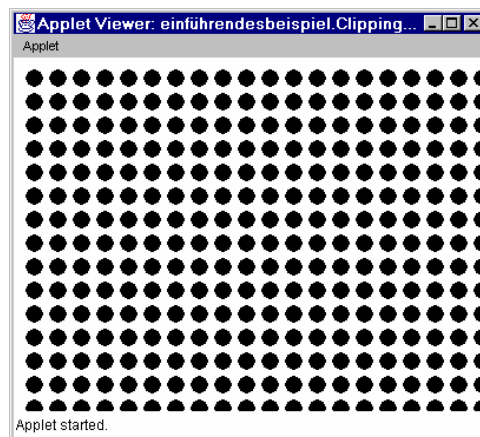
## 1.4.4.4. Clipping

Sie können entweder den gesamten Bereich des grafischen Kontextes modifizieren oder auch nur Teile davon. Falls Sie den modifizierbaren Bereich einschränken, spricht man von einer Clip Region, also einem Teilbereich des grafischen Kontextes.

Falls Sie `drawXXX()` oder `fillXXX()` verwenden, wird jeweils nur jener Bereich modifiziert, welcher in der Clipping Region liegt.

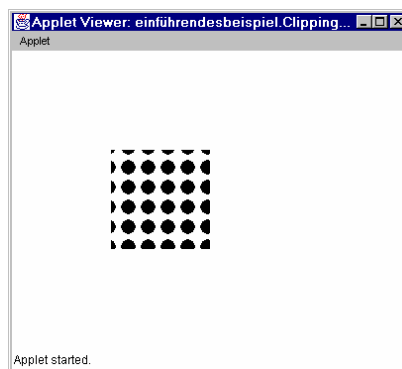
Grundsätzlich wird die Clip Region mit der Methode

```
public void setClip(int n, int m, int l, int k)
```



```
public void paint(Graphics g) {  
    for (int i=10; i<500; i+=20)  
        for (int j=10; j<500; j+=20)  
            g.fillOval(i, j, 15, 15);  
}
```

Jetzt das selbe Applet mit einer Clipping Region, die kleiner ist als der gesamte Graphics Context:



Und hier ist der entsprechende Programmcode:

```
public void paint(Graphics g) {  
    g.setClip(100, 100, 100, 100);  
    for (int i=10; i<500; i+=20)  
        for (int j=10; j<500; j+=20)  
            g.fillOval(i, j, 15, 15);  
}
```

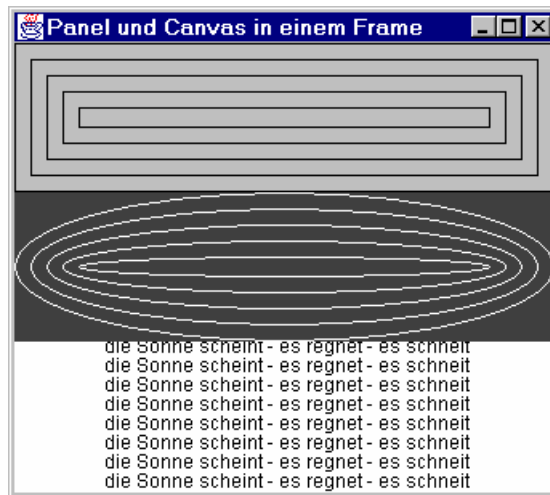


# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.4.5. Verschachtelte Aufrufe der `paint()` Methode

Falls wir in einem Applet oder einer Anwendung in mehreren Klassen die `paint()` Methode definiert und implementiert haben, müssen wir beachten, dass jeweils beim Aufruf der `paint()` Methode durch die Umgebung (Browser,...) alle `paint()` Methoden ausgeführt werden. Das folgende Beispiel zeigt eine solche Verschachtelung und das Ergebnis auf dem Bildschirm als Applikation. Die einzelnen Bereiche sind überlappend: das unterste Frame, auf das Text geschrieben wird, wird von den Canvas überdeckt.

Beim Aufruf der `paint()` Methode wird der darunterliegende Text durch die Grafik verdeckt.



```
package einführendesbeispiel;
import java.awt.*;

public class NestedPaints extends Frame {

    public NestedPaints() {
        super("Panel und Canvas in einem Frame");
        setSize(350,500);
        setLayout(new GridLayout(3,1) );
        add(new RectsPanel() );
        add(new OvalsCanvas() );
    }

    public static void main(String[] args) {
        NestedPaints nestedPaints = new NestedPaints();
        nestedPaints.setVisible(true);
    }

    public void paint(Graphics g) {
        Rectangle bounds = getBounds();
        int y=12;
        while(y<bounds.height) {
            g.drawString("die Sonne scheint - es regnet - es schneit",60,y);
            y+=12;
        }
    }
}

class RectsPanel extends Panel {
    public RectsPanel() {
        setBackground(Color.lightGray);
    }

    public void paint(Graphics g) {
        Rectangle bounds = getBounds();
        int x = 0;
        int y = 0;
        int w = bounds.width - 1;
        int h = bounds.height - 1;
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        for (int i=0; i<10; i++) {
            g.drawRect(x,y,w,h);
            x += 10;
            y += 10;
            w -= 20;
            h -= 20;
        }
    }
}

class OvalsCanvas extends Canvas {
    public OvalsCanvas() {
        setForeground(Color.white);
        setBackground(Color.darkGray);
    }
    public void paint(Graphics g) {
        Rectangle bounds = getBounds();
        int x = 0;
        int y = 0;
        int w = bounds.width - 1;
        int h = bounds.height - 1;
        for (int i=0; i<10; i++) {
            g.drawOval(x,y,w,h);
            x += 10;
            y += 10;
            w -= 20;
            h -= 20;
        }
    }
}
```

## 1.4.5. Der GUI Thread und die `repaint()` Methode

Im Hintergrund jeder Grafik Applikation läuft ein Grafik GUIThread. Sie können diesen abfragen, indem Sie beispielsweise einfach alle Threads anzeigen.

Im obigen Beispiel wurde diese Abfrage eingebaut. Sie liefert als Ergebnis:

```
Anzahl enumerate Threads = 4
Thread 0 ist main
Thread 1 ist AWT-EventQueue-0
Thread 2 ist SunToolkit.PostEventQueue-0
Thread 3 ist AWT-Windows
```

Der dazugehörige Programmcode lautet:

```
Thread[ ] th_Array = new Thread[Thread.activeCount()];
int n = Thread.enumerate(th_Array);
System.out.println("Anzahl enumerate Threads = " + n);
for (int i = 0; i < n; i++) {
    System.out.println("Thread "+i+" ist "+th_Array[i].getName());
}
```

Offensichtlich ist einer der Threads für die Behandlung der Events zuständig, hoffentlich mit hoher Priorität:

```
Thread 0 ist main                Priorität 5
Thread 1 ist AWT-EventQueue-0    Priorität 6
Thread 2 ist SunToolkit.PostEventQueue-0 Priorität 5
Thread 3 ist AWT-Windows        Priorität 5
```

Daneben ist ein Thread mit normaler Priorität für die Steuerung des `paint()`, `repaint()` zuständig. Aufrufe von `paint()` können nicht nur durch die Umgebung generiert werden. Sie können `paint()` auch direkt aufrufen (aber sollten dies gefälligst nicht tun). Besser ist der Aufruf der `repaint()` Methode, die ihrerseits die `paint()` Methode aufruft.

Wir müssen also unterscheiden:

- spontanes Aufrufen der `paint()` Methode durch die Umgebung
- programmgesteuertes Aufrufen der `paint()` Methode.

## 1.4.5.1. Spontaner Aufruf der `paint()` Methode

Sie kennen diesen Effekt bereits: wann immer Sie ein Applet oder ein Grafikprogramm in den Hintergrund und wieder in der Vordergrund bringen, wird der Inhalt der Zeichnungsfläche neu gezeichnet. Dies geschieht mittels eines Aufrufes der `paint()` Methode.

Dies geschieht mit Hilfe des GUI Threads. Dieser ruft die `paint()` Methode auf:

- nach dem Aufbau
- nach der de-Ikonifizierung
- nach der ersten Anzeige
- wann immer Sie wieder zur bestimmten Webseite zurückkehren, selbst wenn die Seite lokal im Cache ist.

Jeder Graphics Context benötigt eine Clipping Area. Diese wird vom Umgebungsthread per Default gesetzt, ausser Sie haben eine eigene Clipping Area definiert, sonst ist es die maximal darstellbare Fläche.

## 1.4.5.2. Die `repaint()` Methode

Falls Sie im Programm komplexe Bereiche grafisch verändern, kann es sinnvoll sein, die grafische Oberfläche Ihrer Applikation oder Ihres Applets neu zu zeichnen. Falls Ihre Komponente einen Graphics Context besitzt, können Sie darauf zugreifen. Sie erhalten diesen Context mit Hilfe der Methode

`getGraphics()`

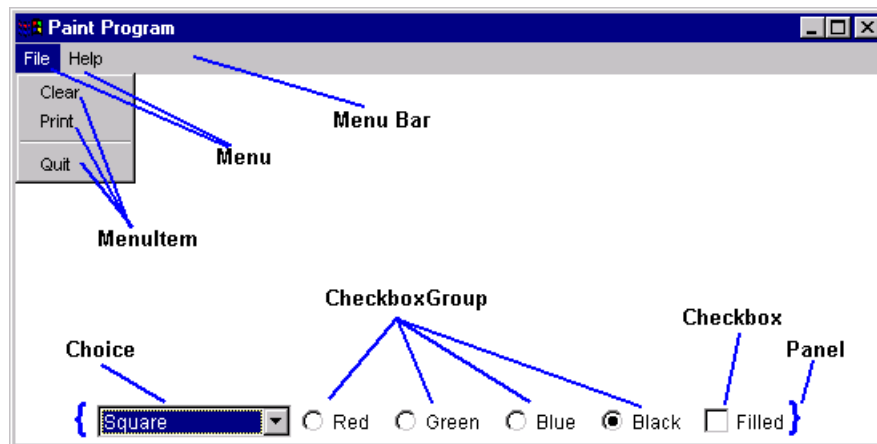
Hier die Beispielausgabe aus dem Programm `NestedPanels`:

```
Graphics Context main: null
Graphics Context NestedPaints: sun.awt.windows.WGraphics
Graphics Context RectsPanel: sun.awt.windows.WGraphics
Graphics Context OvalsCanvas: sun.awt.windows.WGraphics
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.6. Kreieren eines GUIs für ein Zeichenprogramm Praktische Übung

In dieser Übung erstellen Sie ein Rahmenprogramm für ein Zeichenprogramm, alles aus AWT Komponenten. Das eigentliche Zeichenprogramm werden Sie allerdings nicht erstellen. Aber Sie können das ja als Übung machen.




# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
bar = new MenuBar();  
file = new Menu ("File");
```

```
bar = new MenuBar();  
file = new MenuBar("File");
```


```
bar = new Menu();  
file = new Menu("File");
```

```
public class PaintGUI extends Applet {  
    Frame f;  
    canvas paintCanvas;  
    MenuBar bar;  
    Menu file, help;  
    MenuItem clear, print, quit;  
    Panel toolPanel;  
    Choice shapesChoice;  
    CheckboxGroup colorGroup;  
    Checkbox red, green, blue, black;  
    Checkbox filled;  
  
    public void init() {  
        f = new Frame ("Paint Program");  
  
          
    }  
}
```

```
file.add("Clear");  
file.add("Print");  
file.add(Separator);
```

```
file.add(clear);  
file.add(print);  
file.add(Separator);
```


```
file.add(clear);  
file.add(print);  
file.addSeparator();
```

```
MenuItem clear, print, quit;  
Panel toolPanel;  
Choice shapesChoice;  
CheckboxGroup colorGroup;  
Checkbox red, green, blue, black;  
Checkbox filled;  
  
public void init() {  
    f = new Frame ("Paint Program");  
  
    bar = new MenuBar();  
    file = new Menu ("File");  
    clear = new MenuItem("Clear");  
    print = new MenuItem("Print");  
    quit = new MenuItem("Quit");  
  
      
}
```

```
f.add(bar);
```

```
f.setMenuBar(bar);
```

```
f.setMenuBar(file);
```

```
bar = new MenuBar();  
file = new Menu ("File");  
clear = new MenuItem("Clear");  
print = new MenuItem("Print");  
quit = new MenuItem("Quit");  
file.add(clear);  
file.add(print);  
file.addSeparator();  
file.add(quit);  
bar.add(file);  
  
help = new Menu("Help");  
bar.setHelpMenu(help);  
  
      
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
shapesChoice = new Choice();  
shapesChoice.addItem("Square");
```

```
shapesChoice = new Choice();  
shapesChoice.addItem("Square");
```

```
shapesChoice = new Choice();  
shapesChoice.addSquare();
```

```
clear = new MenuItem("Clear");  
print = new MenuItem("Print");  
quit = new MenuItem("Quit");  
file.add(clear);  
file.add(print);  
file.addSeparator();  
file.add(quit);  
bar.add(file);
```

```
help = new Menu("Help");  
bar.setHelpMenu(help);
```

```
f.setMenuBar(bar);
```



```
}  
}
```

```
colorGroup = new CheckboxGroup();  
red = new Check(red, colorGroup, true);
```

```
colorGroup = new CheckboxGroup();  
red = new Checkbox("Red", colorGroup, false);
```

```
colorGroup = new CheckboxGroup("Red");
```

```
file.add(quit);  
bar.add(file);
```

```
help = new Menu("Help");  
bar.setHelpMenu();
```

```
f.setMenuBar(bar);
```

```
toolpanel = new Panel();
```

```
shapesChoice = new Choice();  
shapesChoice.addItem("Square");  
shapesChoice.addItem("Circle");  
shapesChoice.addItem("Line");  
shapesChoice.addItem("Rounded Square");
```



```
}  
}
```

```
file.add(quit);  
bar.add(file);
```

```
help = new Menu("Help");  
bar.setHelpMenu();
```

```
f.setMenuBar(bar);
```

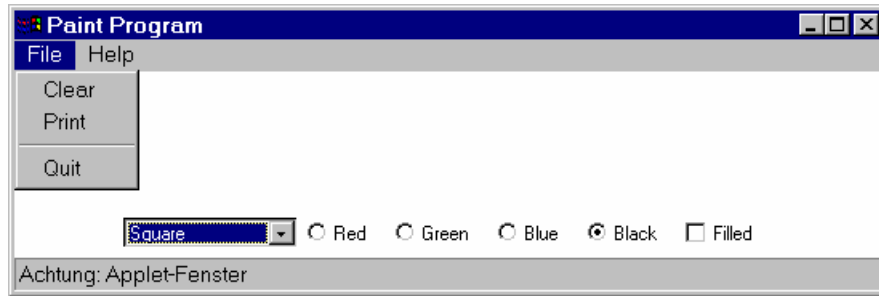
```
toolpanel = new Panel();
```

```
shapesChoice = new Choice();  
shapesChoice.addItem("Square");  
shapesChoice.addItem("Circle");  
shapesChoice.addItem("Line");  
shapesChoice.addItem("Rounded Square");  
colorGroup = new CheckboxGroup();  
red = new Checkbox("Red", colorGroup, false);
```

```
}  
}
```

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

## 1.4.6.1.1.1. Programmcode für den Applikationsrahmen



```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class PaintGUI extends Applet {

    Frame f;
    Canvas paintCanvas;
    MenuBar bar;
    Menu file, help;
    MenuItem clear, print, quit;
    Panel toolPanel;
    Choice shapesChoice;
    CheckboxGroup colorGroup;
    Checkbox red, green, blue, black;
    Checkbox filled;

    public void init() {
        f = new Frame ("Paint Program");

        bar = new MenuBar ();
        file = new Menu ("File");
        clear = new MenuItem ("Clear");
        print = new MenuItem ("Print");
        quit = new MenuItem ("Quit");
        file.add (clear);
        file.add (print);
        file.addSeparator();
        file.add (quit);
        bar.add (file);

        help = new Menu ("Help");
        bar.setHelpMenu (help);

        f.setMenuBar (bar);

        toolPanel = new Panel ();

        shapesChoice = new Choice ();
        shapesChoice.addItem ("Square");
        shapesChoice.addItem ("Circle");
        shapesChoice.addItem ("Line");
        shapesChoice.addItem ("Rounded Square");
```



# JAVA AWT GUI UND APPLLET GRUNDLAGEN

```
        colorGroup = new CheckboxGroup();
        red = new Checkbox("Red", colorGroup, false);
        green = new Checkbox("Green", colorGroup, false);
        blue = new Checkbox("Blue", colorGroup, false);
        black = new Checkbox("Black", colorGroup, true);

        filled = new Checkbox ("Filled", false);

        toolPanel.add (shapesChoice);
        toolPanel.add (red);
        toolPanel.add (green);
        toolPanel.add (blue);
        toolPanel.add (black);
        toolPanel.add (filled);

        f.add ("South", toolPanel);
        paintCanvas = new Canvas();
        f.add ("Center", paintCanvas);
        f.setSize (550, 300);
    }

    public void start () {
        f.setVisible (true);
    }

    public void stop () {
        f.setVisible (false);
    }
}
```

## 1.4.7. Quiz

Hier einige Fragen zur Wiederholung oder damit Sie sich den Stoff nochmals einprägen können:

1. Welcher der folgenden Komponenten können Sie einem Container *nicht* anhängen?<sup>11</sup>
  - a) Applet
  - b) Component
  - c) Container
  - d) Menu
  - e) Panel
2. Sie haben eine TextArea mit folgendem Programmfragment<sup>12</sup>

```
TextField t = new TextArea("12345", 5, 5);
```

Welche der folgenden Aussagen trifft zu?
  - a) es werden genau fünf Zeichen pro Zeile angezeigt, sofern nicht irgendwo etwas anderes festgelegt wird.
  - b) der Text wird fünf Zeilen hoch angezeigt
  - c) die maximale Anzahl Zeichen pro Zeile beträgt fünf
  - d) der Benutzer wird den Text editieren können
  - e) die angegebene Zeichenkette kann mit mehreren Fonts angezeigt werden
3. Welche der folgenden Komponenten sind standalone Komponenten?<sup>13</sup>
  - a) Panel
  - b) Frame
  - c) Window
  - d) Dialog
  - e) ScrollPane
4. Welche der folgenden Komponenten benötigt ein setFocus() um auf Benutzereingaben reagieren zu können?<sup>14</sup>
  - a) Menu
  - b) Canvas
  - c) FileDialog
  - d) Button
  - e) List
5. Welche Menükomponente kann jeder Komponente hinzugefügt werden?<sup>15</sup>
  - a) HelpMenu
  - b) CheckboxMenuItem
  - c) PopupMenu
  - d) MenuItem
  - e) MenuBar

---

<sup>11</sup> d

<sup>12</sup> b d

<sup>13</sup> b c d

<sup>14</sup> b

<sup>15</sup> c

## 1.4.8. Zusammenfassung Modul

In diesem Modul haben Sie gelernt,

- mit den wichtigsten AWT Komponenten umzugehen.
- mit den AWT komplexere Benutzerinterfaces zu bauen.
- Fonts und Farbgebung der AWT Komponenten zu kontrollieren.

## 1.5. Zusammenfassung

Nachdem Sie diese Kursunterlagen durchgearbeitet haben, sollten Sie in der Lage sein:

- das AWT Package und seine Komponenten zu beschreiben.
- Frame und Panel Container einzusetzen.
- Panels innerhalb anderer Container zu platzieren und damit komplexere Layouts zu kreieren.
- Ereignisse abzufangen und den Programmablauf ereignisgesteuert zu gestalten.
- passende Interfaces und Handler Methoden einzusetzen, um die Ereignisse zu beschreiben.
- AWT Komponenten für den Bau von GUIs einzusetzen, auch komplexeren.
- Farbgebung und Fonts der AWT Komponenten zu kontrollieren.

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

<b>JAVA GUI UND APPLLET GRUNDLAGEN .....</b>	<b>1</b>
1.1. KURSÜBERSICHT .....	1
1.1.1.1. Lernziele .....	1
1.2. MODUL 1 : DIE ENTWICKLUNG VON JAVA GUIs .....	2
1.2.1. Modul Einleitung .....	2
1.2.1.1. Lernziele .....	2
1.2.2. Lektion 1 - Java GUI Grundlagen .....	3
1.2.2.1. Das java.awt Package .....	3
1.2.2.2. Applet .....	4
1.2.2.3. Panel Window .....	4
1.2.2.4. Dialog Frame .....	4
1.2.2.5. TextArea, TextField .....	4
1.2.2.6. Window .....	5
1.2.2.7. Panel .....	5
1.2.2.8. Button .....	5
1.2.2.9. Checkbox .....	5
1.2.2.10. CheckBoxGroup .....	5
1.2.2.11. Choice .....	5
1.2.2.12. Container .....	6
1.2.2.13. TextComponent .....	6
1.2.2.14. Component .....	6
1.2.2.15. GridLayout .....	6
1.2.2.16. FlowLayout .....	6
1.2.2.17. BorderLayout .....	7
1.2.2.18. Event .....	7
1.2.2.19. Exception .....	7
1.2.2.20. Error .....	7
1.2.3. Container und Komponenten .....	8
1.2.3.1. Positionieren von Komponenten .....	8
1.2.4. Frames .....	9
1.2.5. Panels .....	10
1.2.6. Container Layouts .....	11
1.2.6.1. Ein einfaches Layout Manager Beispiel .....	12
1.2.7. Lektion 2 - Layout Manager .....	14
1.2.7.1. Flow Layout Manager .....	14
1.2.7.1.1. Flow Layout Manager Beispiel .....	15
1.2.7.2. BorderLayout Manager .....	16
1.2.7.2.1. BorderLayout Manager Beispiel .....	17
1.2.7.3. Der GridLayout Manager .....	18
1.2.7.3.1. GridLayout Manager Beispiel .....	19
1.2.7.4. Der CardLayout Manager .....	20
1.2.7.4.1. CardLayout Manager Beispiel .....	20
1.2.7.5. GridBagLayout Manager .....	23
1.2.7.5.1. GridBagLayout Manager Beispiel .....	23
1.2.8. Lektion 3 - Komplexere Layouts .....	24
1.2.8.1. Containers - Frames und Panels anders betrachtet .....	24
1.2.8.2. Kreieren von Panels und komplexen Layouts .....	24
1.2.8.3. Komplexeres Layout Beispiel .....	25
1.2.9. Praktische Übung .....	26
1.2.9.1. Kreieren Sie ein GUI für einen Taschenrechner .....	26
1.2.9.2. Einführung in die Aufgabe .....	26
1.2.9.3. Musterlösung für das Taschenrechner GUI .....	30
1.2.10. Quiz .....	32
1.2.11. Zusammenfassung .....	33
1.3. MODUL 2 : DAS AWT EVENT MODELL .....	34
1.3.1. Modul Einleitung .....	34
1.3.1.1. Lernziele .....	34
1.3.2. Lektion 1 - Event Grundlagen .....	35
1.3.2.1. Ein Ereignis ist .....	35
1.3.2.2. Ereignisquellen .....	35
1.3.2.3. Event Handler .....	36
1.3.3. Lektion 2 - JDK 1.0 versus neues Event Modell .....	37
1.3.3.1. Wie werden Ereignisse verarbeitet? .....	37

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

1.3.3.2.	Das hierarchische Modell (JDK1.0) .....	38
1.3.3.3.	Das Delegationsmodell (JDK1.1) .....	39
1.3.4.	<i>Lektion 3 - Java GUI Verhalten</i> .....	42
1.3.4.1.	Die Event Kategorien .....	42
1.3.4.2.	Ein komplexeres Beispiel .....	44
1.3.4.2.1.	Erläuterungen zum Programm .....	45
1.3.4.2.2.	Der Programmcode .....	45
1.3.4.3.	Mehrere Listeners .....	47
1.3.4.4.	Ereignis Adapter .....	47
1.3.4.5.	Programmbeispiel .....	48
1.3.5.	<i>Taschenrechner mit Event Handler - Praktische Übung</i> .....	48
1.3.5.1.	Einleitung .....	48
1.3.5.1.1.	Der Taschenrechner .....	51
1.3.6.	<i>Quiz</i> .....	55
1.3.7.	<i>Modul Zusammenfassung</i> .....	56
1.4.	<b>MODUL 3 : DIE AWT KOMPONENTEN BIBLIOTHEK</b> .....	57
1.4.1.	<i>Modul Einleitung</i> .....	57
1.4.1.1.	Lernziele .....	57
1.4.2.	<i>Lektion 1 - AWT Schlüsselkomponenten</i> .....	58
1.4.2.1.	Die AWT Komponenten Bibliothek .....	58
1.4.2.2.	Buttons, Checkbox und Choice Komponenten .....	59
1.4.2.2.1.	Buttons .....	59
1.4.2.2.2.	Checkboxes .....	60
1.4.2.2.3.	CheckboxGroup - Radio Buttons .....	62
1.4.2.2.4.	Choice .....	63
1.4.2.3.	Canvas, Frame, Panel und ScrollPane .....	64
1.4.2.3.1.	Canvas .....	64
1.4.2.3.2.	Frames .....	66
1.4.2.3.3.	Panels .....	66
1.4.2.3.4.	ScrollPane .....	68
1.4.2.4.	Label, Text und List Komponenten .....	69
1.4.2.4.1.	Label .....	69
1.4.2.4.2.	TextField .....	69
1.4.2.4.2.1.	TextField Beispiel ohne Listener .....	69
1.4.2.4.2.2.	TextField Beispiel mit Listener .....	70
1.4.2.4.3.	TextArea .....	71
1.4.2.4.4.	List .....	72
1.4.2.5.	Dialog und FileDialog Komponenten .....	73
1.4.2.5.1.	Dialog .....	73
1.4.2.5.1.1.	Dialog Beispiel Anwendung .....	74
1.4.2.5.2.	FileDialog .....	76
1.4.2.6.	Menü Komponenten .....	77
1.4.2.6.1.	Menus .....	77
1.4.2.6.2.	MenuBar .....	78
1.4.2.6.3.	Menu .....	79
1.4.2.6.4.	MenuItem .....	80
1.4.2.6.5.	CheckboxMenuItem .....	82
1.4.2.6.6.	PopupMenu .....	83
1.4.3.	<i>Lektion 2 - Kontrolle des Aussehens der Komponenten und Drucken</i> .....	84
1.4.3.1.	Kontrolle der Farbe der Komponenten .....	84
1.4.3.2.	Verändern der Komponentenfarbe - Beispiel .....	84
1.4.3.3.	Ändern der Fonts der Komponenten .....	85
1.4.3.4.	Drucken in AWT .....	86
1.4.4.	<i>Die paint () Methode und der Graphics Context</i> .....	89
1.4.4.1.	Setzen der Farbe .....	90
1.4.4.2.	Fonts - Die Auswahl eines Fonts .....	90
1.4.4.3.	Zeichnen und Malen von Flächen .....	92
1.4.4.3.1.	drawLine() .....	92
1.4.4.3.2.	drawRectangle() und fillRectangle(); draw3Direct(), drawRoundRect() .....	92
1.4.4.3.3.	drawOval() und fillOval() .....	93
1.4.4.3.4.	drawArc() und fillArc() .....	93
1.4.4.3.5.	drawPolygon() und fillPolygon() .....	93
1.4.4.3.6.	drawPolyline() .....	93
1.4.4.3.7.	drawString() .....	93
1.4.4.3.8.	drawImage() .....	93
1.4.4.4.	Clipping .....	96
1.4.4.5.	Verschachtelte Aufrufe der paint () Methode .....	97

# JAVA AWT GUI UND APPLLET GRUNDLAGEN

1.4.5.	Der GUI Thread und die <i>repaint()</i> Methode.....	99
1.4.5.1.	Spontaner Aufruf der <i>paint()</i> Methode .....	100
1.4.5.2.	Die <i>repaint()</i> Methode.....	100
1.4.6.	Kreieren eines GUIs für ein Zeichenprogramm <i>Praktische Übung</i> .....	101
1.4.6.1.1.1.	Programmcode für den Applikationsrahmen .....	104
1.4.7.	Quiz .....	106
1.4.8.	Zusammenfassung Modul.....	107
1.5.	ZUSAMMENFASSUNG .....	107

© Java : Sun Microsystems

© Duke : Sun Microsystems