TECHNISCHE UNIVERSITÄT DARMSTADT

Kapitel 10: GUIs

Karsten Weihe

TECHNISCHE UNIVERSITÄT DARMSTADT

Window Manager

Als erstes ein paar Worte zu einem wichtigen Bestandteil von GUIs, der uns beim Schreiben von GUIs einen Haufen Arbeit abnimmt: der Window Manager, also deutsch so etwas wie Fensterverwalter.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

Hier in der FOP reichen in der Tat ein paar erläuternde Worte als Hintergrundwissen zum Window Manager.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Systemprozess**
  - **Wird in der Regel beim Booten hochgefahren**
  - **Läuft permanent im Hintergrund (*service*, *daemon*)**
- **Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.**
  - **Öffnen, Schließen, (De-)ikonifizieren, Größe ändern**
  - **Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet**

***Erinnerung*: In Kapitel 09, zu Beginn des Abschnitts zu Runnable und Thread, hatten wir schon echte Prozesse im Gegensatz zu Threads kurz besprochen. Siehe auch die *Zusammenfassung Programme und Prozesse*.**

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

Um den Start des Window Managers müssen Sie sich in der Regel nicht kümmern.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

Etliche Prozesse laufen auf den üblichen Arbeitsplatzrechnern die ganze Zeit im Hintergrund mit, darunter eben auch der Window Manager.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

In der Windows-Welt spricht man in der Regel von Services, in der UNIX-Welt eher von Daemons.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

Generell hat man das Thema GUI-Programmierung im Laufe der Jahrzehnte sehr gut verstanden und weiß gut, was alle GUI-Applikationen im Prinzip so alles gemeinsam haben. Diese Gemeinsamkeiten kann man dann in zentrale Dienste herausfaktorisieren, ein für allemal implementieren und dann allen Applikationen anbieten. Dazu gehören auch die Funktionalitäten des Window Managers.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

Hier sehen Sie ein paar Beispiele für das, was der Window Manager Ihnen alles so abnimmt, so dass Sie sich beim Schreiben einer GUI-Applikation überhaupt nicht darum kümmern müssen. Sie können in Ihrer Applikation beispielsweise entscheiden, *wann* ein Fenster, das zu Ihrer GUI-Applikation gehört, geöffnet oder geschlossen wird, welches Bild nach der Ikonifizierung zu sehen ist (also wenn anstelle des Fensters nur noch ein kleines Symbol = icon in der Symbolleiste zu sehen ist), und was zu tun ist, um den Inhalt des Fensters nach einer Größenänderung wieder aufzubauen. Aber die umfangreichen, schwierig zu implementierenden und hochgradig plattformabhängigen Schritte, die für alle diese Operationen realisiert werden müssen und immer wieder mehr oder weniger gleich sind – das nimmt Ihnen der Code ab, der im Window Manager enthalten ist.

Window Manager

TECHNISCHE UNIVERSITÄT DARMSTADT

- Systemprozess
  - Wird in der Regel beim Booten hochgefahren
  - Läuft permanent im Hintergrund (*service*, *daemon*)
- Stellt die generelle, anwendungs*un*spezifische Funktionalität zur Verfügung, z.B.
  - Öffnen, Schließen, (De-)ikonifizieren, Größe ändern
  - Der Rahmen eines Fensters und der Bildschirm-hintergrund werden vom Window Manager verwaltet

Am augenfälligsten tritt der Window Manager in Erscheinung in den Fenstern, die er selbst verwaltet. Konzeptuell gesehen, ist der Bildschirmhintergrund auch ein Fenster. Und der Rahmen rund um ein Fenster – also, das., was bei jedem Fenster absolut identisch aussieht – besteht ebenfalls aus einzelnen Fenstern.

TECHNISCHE UNIVERSITÄT DARMSTADT

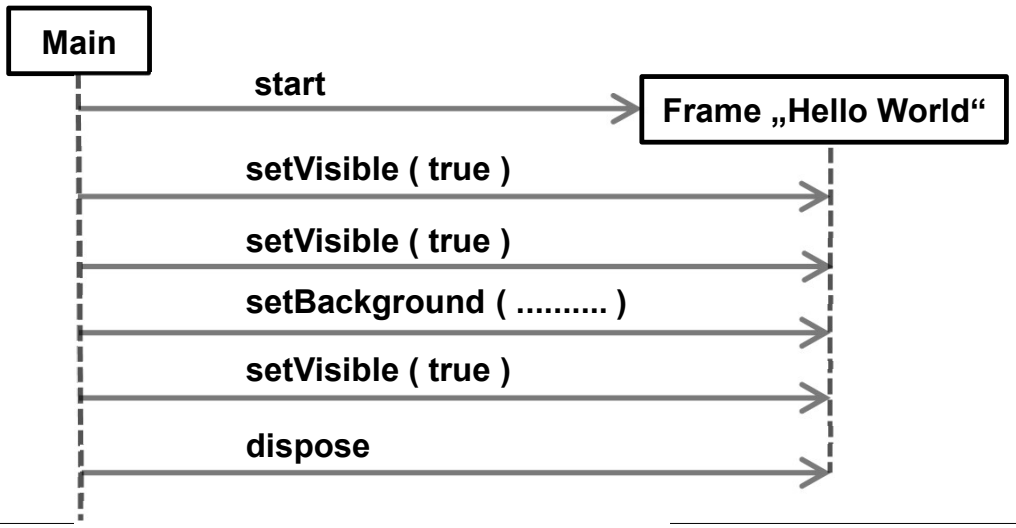
Fenster mit Rahmen: Klasse Frame

Jetzt öffnen wir Fenster aus einer GUI-Applikation heraus. Dazu gibt es zwar auch eine Klasse mit dem passenden Namen Window, aber das wäre dann ein Fenster ohne Rahmen. Für Fenster *mit* Rahmen ist Klasse Frame von Klasse Window abgeleitet. Die zusätzliche Funktionalität von Klasse Frame gegenüber Klasse Window ist, dass eben ein vom Window Manager verwalteter Rahmen hinzugefügt ist.

Wenn nichts anderes gesagt ist, gehört jede Klasse und jedes Interface zum Package java.awt; awt ist die Abkürzung für abstract window toolkit.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT



Im UML-Sequenzdiagramm sehen wir, dass ein Fenster, das aus einem Java-Programm heraus geöffnet wird, ein eigener Thread ist. Das Gesamtbild ist in diesem ersten Beispiel sehr einfach: Es gibt nur zwei Threads insgesamt, nur Nachrichten vom ersten zum zweiten Thread und keine Antworten auf die Nachrichten.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Nun der Java-Code des Main-Threads.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Erinnerung: Die Klassenmethode sleep von Klasse Thread hält den Thread, in dem sie aufgerufen wird, um die Anzahl Millisekunden an, die im Parameter spezifiziert ist. Damit wird also der Main-Thread aus dem UML-Sequenzdiagramm angehalten.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Für ein Fenster mit Rahmen wird ein Objekt der Klasse Frame eingerichtet. In dem hier aufgerufenen Konstruktor wird der neue Thread gestartet.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Dieser Konstruktor von Klasse Frame hat einen einzelnen Parameter vom formalen Typ String. Das ist der Titel, der bei gängigen Window Managern oben mittig im Rahmen angezeigt wird.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Ein Fenster ist entweder sichtbar oder unsichtbar. Bei der Einrichtung des Frame-Objektes ist das Fenster erst einmal unsichtbar und muss mit Methode setVisible erst sichtbar gemacht werden.

Der Sinn dahinter ist, dass häufig eine ganze Menge zu tun ist, um den Inhalt eines Fensters aufzubauen, und solange sollte ein Fenster besser noch nicht sichtbar sein.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Warum hat Methode setVisible einen booleschen Parameter? Nun, mit true wird das Fenster sichtbar, mit false unsichtbar.

Die Anwendungen von setVisible auf dieser Folie haben natürlich keinen speziellen Sinn, sondern sind rein illustrativ.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Mit der Methode setBackground wird die Hintergrundfarbe des Fensters auf die Farbe gesetzt, die als aktualer Parameter übergeben wird.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Klasse Color hat unter anderem einen Konstruktor mit drei int-Parametern, die den Rot-, Grün- und Blauanteil der neu einzurichtenden Farbe jeweils im Bereich 0 bis 255 spezifizieren. Dabei bedeutet 255 maximale Farbstärke. Durch die additive Farbmischung ergibt dreimal 255 also weiß, dreimal 0 ist schwarz. Die hier spezifizierte Farbe ist also ungefähr ein sehr dunkles, blaustichiges Türkis.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setVisible ( false );
Thread.sleep ( 2000 );
frame.setBackground ( new Color ( 20, 30, 40 ) );
frame.setVisible ( true );
frame.dispose();
```

Sobald die Programmausführung an einer Stelle angekommen ist, an der ein Frame-Objekt nicht mehr benötigt wird, sollte sicherheitshalber immer Methode dispose aufgerufen werden. Alle Ressourcen, die vom Fenster und allen seinen Bestandteilen mit Beschlag belegt sind, werden dadurch wieder freigegeben, und der Thread des Fensters wird beendet.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.ICONIFIED );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.NORMAL );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.MAXIMIZED_HORIZ );
frame.dispose();
```

Noch eine kleine Variation dieses einführenden Beispiels, um ein paar weitere Methoden von Frame zu sehen.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.ICONIFIED );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.NORMAL );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.MAXIMIZED_HORIZ );
frame.dispose();
```

Methode setExtendedState setzt den Status des Fensters auf eine von mehreren vordefinierten Möglichkeiten, die durch Klassenkonstanten von Klasse Frame kodiert sind. Wie der Name schon sagt, sorgt Konstante ICONIFIED dafür, dass das Fenster ikonifiziert wird.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.ICONIFIED );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.NORMAL );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.MAXIMIZED_HORIZ );
frame.dispose();
```

Diese Konstante de-ikonifiziert das Fenster wieder.

Frame

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
frame.setVisible ( true );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.ICONIFIED );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.NORMAL );
Thread.sleep ( 2000 );
frame.setExtendedState ( Frame.MAXIMIZED_HORIZ );
frame.dispose();
```

Noch ein letztes Beispiel dazu: Damit wird das Fenster horizontal auf die gesamte Bildschirmbreite ausgedehnt.

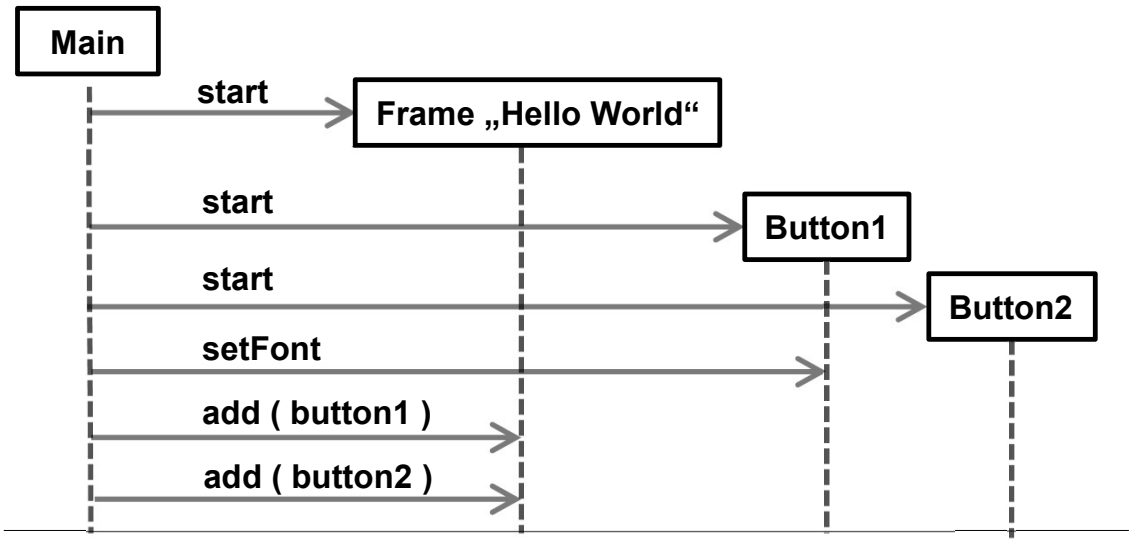
TECHNISCHE UNIVERSITÄT DARMSTADT

Button und ActionListener

Bisher haben Fenster bei uns noch keinen Inhalt. Als erste Art des Inhalts schauen wir uns Buttons, also Knöpfe an; später dann auch andere, komplexere Komponententypen von Fenstern.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT



Das UML-Sequenzdiagramm ist beim einführenden Beispiel etwas komplizierter. Nicht nur das Fenster, sondern auch die beiden Buttons, die wir im Fenster platzieren werden, sind jeweils separate Threads.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Für jeden Komponententyp in Fenstern wie etwa Buttons gibt es jeweils eine eigene Klasse mit aussagekräftigem Namen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Bei Buttons heißt die Klasse einfach genau so.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Klasse Button hat einen Konstruktor mit einem einzelnen String-Parameter. Das ist der Text, der auf dem Button angezeigt wird.

***Nebenbemerkung*: Die Größe des Buttons wird im Wesentlichen durch die Länge des Strings sowie Schriftart und Schriftgröße determiniert. Aber auch die Größe des Fenster und die Größen der anderen GUI-Komponenten, die neben diesen beiden Buttons vielleicht noch im Fenster angezeigt werden sollen, beeinflussen die Größe eines Buttons.**

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Selbstverständlich hat Klasse Button auch eine Methode, um Schriftart und Schriftgröße festzulegen. Das englische Wort font heißt Schriftart. Dafür gibt es eine Klasse mit passendem Namen, und ein Objekt dieser Klasse als aktualer Parameter definiert die neue Schriftart.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Eine Schriftart setzt sich aus Schriftfamilie und Schnitt zusammen. Hier wird die gängige Schriftfamilie Helvetica verwendet, und der Schnitt ist Fettdruck, englisch bold. Es gibt auch kursiv, englisch italic, normal und fett-kursiv zugleich.

Da es nur wenige Schnitte gibt, bietet es sich an, für jeden davon eine eigene Klassenkonstante von FONT anzubieten. Schriftarten hingegen gibt es wie Sand am Meer, und permanent kommen neue hinzu, da geht es nur mit dem Namen der Schriftart als String.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Der dritte Parameter ist die Schriftgröße.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

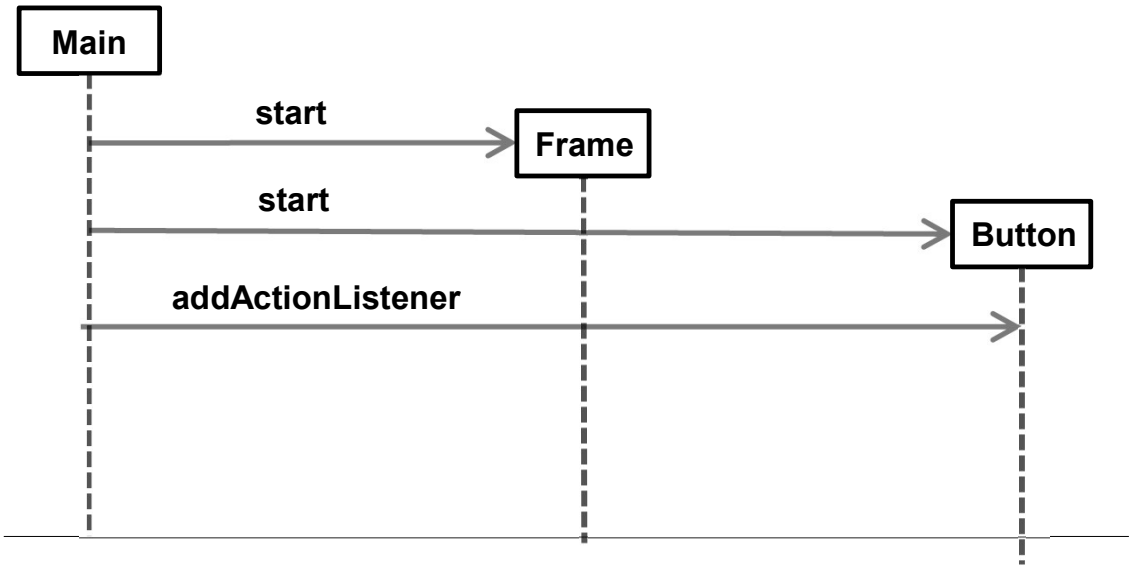
```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
button1.setFont ( new Font ( “Helvetica“, Font.BOLD, 20 ) );
frame.add ( button1 );
frame.add ( button2 );
```

Mit Methode add werden beide Buttons in das Fenster eingefügt. Wir haben es nicht in der Hand, wie die Buttons im Fenster platziert werden.

***Vorgriff*: Später, beim Thema LayoutManager, werden wir sehen, wie wir die Platzierung der Komponenten in einem Fenster steuern können.**

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT



Buttons sollen aber nicht nur angezeigt werden, sondern es soll eine spezifische Aktion passieren, wenn man auf einen Button klickt. Hier kommt der ActionListener ins Spiel, der zweite Bestandteil des Titels dieses Abschnitts. Was Sie in diesem Diagramm sehen, ist grob gesprochen das, was wir in Main dazu zu schreiben haben. Das sehen wir uns im Detail auf der nächsten Folie in Java an.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Color color = new Color ( 20, 30, 40 );
Button button = new Button ( “Change background color“ );
ActionListener listener
        = new BackgroundColorListener ( frame, color );
button.addActionListener ( listener );
```

Hier sehen Sie also die Anweisungen, die in Main notwendig sind.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Color color = new Color ( 20, 30, 40 );
Button button = new Button ( “Change background color“ );
ActionListener listener
        = new BackgroundColorListener ( frame, color );
button.addActionListener ( listener );
```

Ein Fenster und eine Farbe als Beispielmaterial, ...

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Color color = new Color ( 20, 30, 40 );
Button button = new Button ( “Change background color“ );
ActionListener listener
        = new BackgroundColorListener ( frame, color );
button.addActionListener ( listener );
```

... der Button, bei dem wir den ActionListener registrieren werden, ...

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Color color = new Color ( 20, 30, 40 );
Button button = new Button ( “Change background color“ );
ActionListener listener
        = new BackgroundColorListener ( frame, color );
button.addActionListener ( listener );
```

... und der ActionListener von der gleich noch zu erstellenden Klasse BackgroundColorListener.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

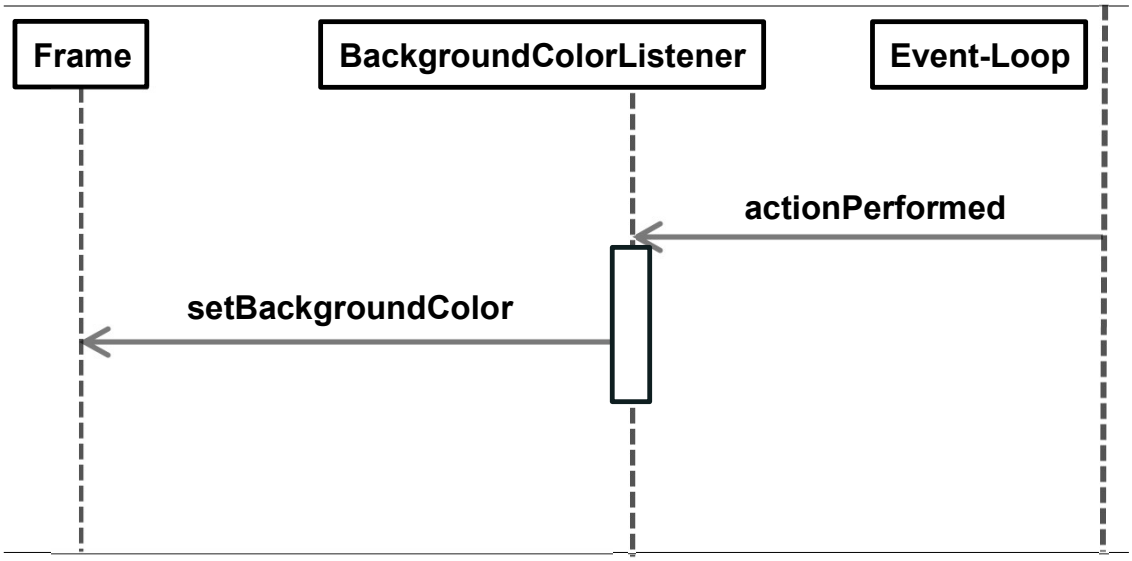
```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Color color = new Color ( 20, 30, 40 );
Button button = new Button ( “Change background color“ );
ActionListener listener
        = new BackgroundColorListener ( frame, color );
button.addActionListener ( listener );
```

Komponenten wie Button haben eine eigene Methode mit aussagekräftigen Namen, um Listener von verschiedenen Arten zu registrieren. Die Konsequenz von addActionListener ist: Wann immer auf den Button geklickt wird, wird Methode actionPerformed des Listeners aufgerufen. Man kann auch mehrere ActionListener bei demselben Button mit Methode addActionListener anhängen, dann wird Methode actionPerformed bei jedem davon aufgerufen, wann immer auf den Button geklickt wird.

Dahinter steht ein Thread, der automatisch eingerichtet wird und den sicherlich passenden Namen Event Dispatch Thread trägt. Dieser Thread wartet permanent auf Eingaben von Maus und Tastatur und ruft bei jedem Event die Listener auf, die an der Komponente registriert sind, in der der Mauszeiger zum Zeitpunkt des Events ist.

Button und ActionListener

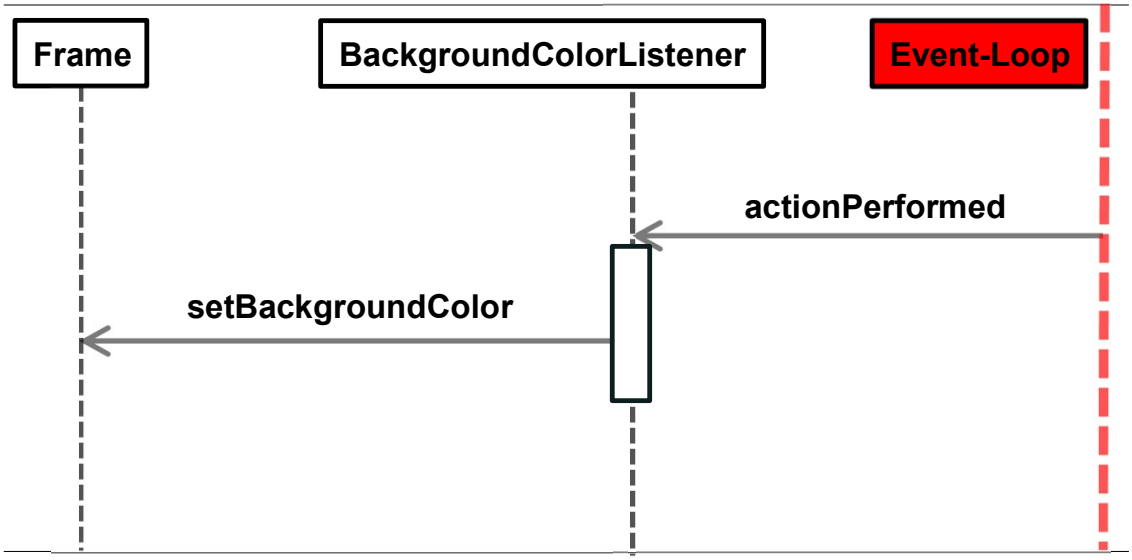
TECHNISCHE UNIVERSITÄT DARMSTADT



Hier sehen Sie nun, was im Hintergrund passiert und nicht von Ihnen zu implementieren ist. Dazu sollte allerdings gesagt werden, dass dieses Bild nicht hundertprozentig der Realität im Java-Laufzeitsystem entspricht, aber durch die Konformität zur bisher gewählten Bildsprache (hoffentlich) verständlicher als die Detailrealität ist und auch ausreichend nah an der Realität sein sollte, um das korrekte Grundverständnis zu vermitteln.

Button und ActionListener

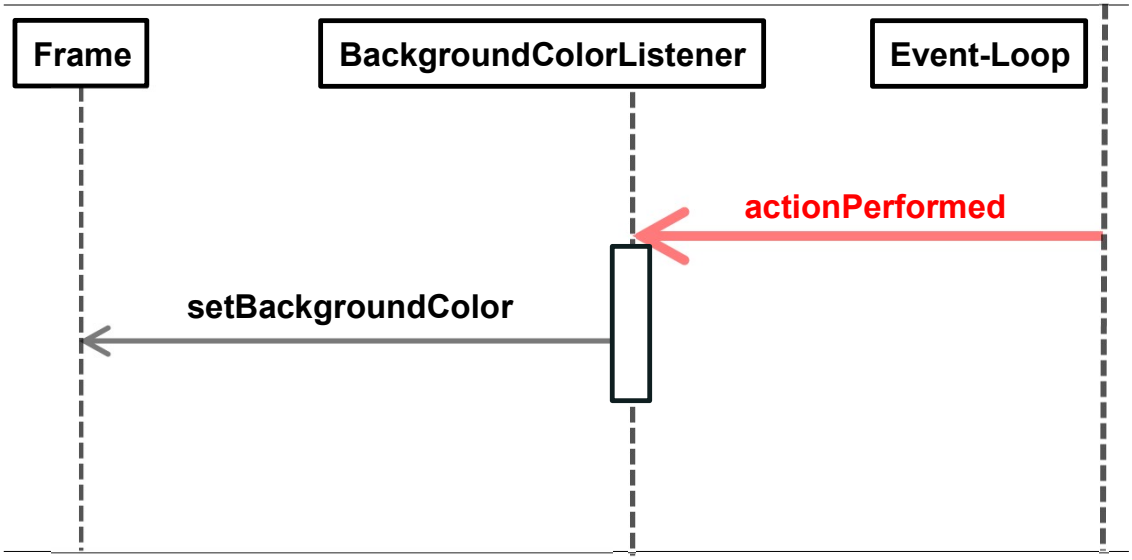
TECHNISCHE UNIVERSITÄT DARMSTADT



Neben den Threads, die wir selbst einrichten, richtet das Laufzeitsystem für unser Java-Programm immer auch ein paar weitere Threads ein. Einer davon ist die Event-Loop. Die Event-Loop überwacht unter anderem, ob der Nutzer der Programms mit der Tastatur oder mit der Maus etwas macht, und sorgt dafür, dass daraufhin der dafür vorgesehene Code ausgeführt wird.

Button und ActionListener

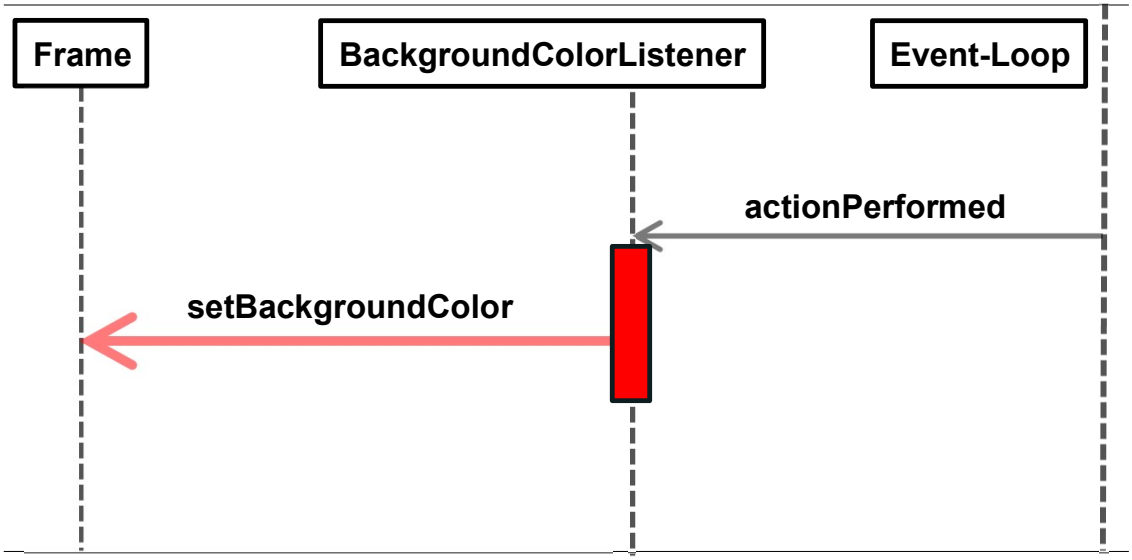
TECHNISCHE UNIVERSITÄT DARMSTADT



Zum Beispiel bei einem Mausklick werden die momentanen Koordinaten des Mauzeigers protokolliert, und es wird berechnet, in welcher graphischen Komponente diese Koordinaten sind. Wir nehmen hier an, dass der Mausklick in dem Button mit Titel „Change background color“ passiert, den wir auf der letzten Folie in das Fenster eingefügt hatten. Für diesen Button haben wir mit addActionListener ein Objekt von ChangeBackgroundListener registriert. Intern wird dieses Objekt bei der Event-Loop registriert. Und die ruft dann die Methode actionPerformed von ChangeBackgroundListener auf, die für diesen Fall vorgesehen ist.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT



Wir werden diese Methode actionPerformed gleich implementieren. Hier sehen wir aber schon den wesentlichen Punkt: Das Listener-Objekt speichert als Attribut einen Verweis auf das Fenster und ruft bei diesem Fenster die Methode auf, mit der die Hintergrundvariable geändert werden kann.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BackgroundColorListener implements ActionListener {
    Frame frame;
    Color color;
    public BackgroundColorListener ( Frame frame, Color color ) {
        this.frame = frame;
        this.color = color;
    }
    .........
}
```

Nun implementieren wir also noch die Klasse BackgroundColorListener.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BackgroundColorListener implements ActionListener {
   Frame frame;
   Color color;
   public BackgroundColorListener ( Frame frame, Color color ) {
      this.frame = frame;
      this.color = color;
   }
   .........
}
```

ActionListener ist ein Interface und findet sich in Package java.awt.event.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BackgroundColorListener implements ActionListener {
  Frame frame;
  Color color;
  public BackgroundColorListener ( Frame frame, Color color ) {
    this.frame = frame;
    this.color = color;
  }
  .....
}
```

Wenn der Button angeklickt wird, soll BackgroundColorListener die Hintergrundfarbe neu setzen. Dafür braucht die Klasse einerseits einen Verweis auf das Fenster, ...

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BackgroundColorListener implements ActionListener {
   Frame frame;
   Color color;
   public BackgroundColorListener ( Frame frame, Color color ) {
      this.frame = frame;
      this.color = color;
   }
   .........
}
```

... andererseits die Farbe, wieder kodiert in einem Objekt der Klasse Color.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BackgroundColorListener implements ActionListener {
    .....
    public void actionPerformed ( ActionEvent event ) {
        frame.setBackground ( color );
    }
}
```

ActionListener ist ein funktionales Interface, hat also nur eine Objektmethode, die nicht default ist. Tatsächlich hat es auch nur die funktionale Methode, und die heißt actionPerformed wie schon gesehen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BackgroundColorListener implements ActionListener {
    .....
    public void actionPerformed ( ActionEvent event ) {
        frame.setBackground ( color );
    }
}
```

Der Parameter wird von der Event-Loop erzeugt und initialisiert mit Detailinformationen zum Event. Diesen Parameter schauen wir uns erst im nächsten Beispiel an. Die Implementation von actionPerformed im Beispiel auf der vorliegenden Folie benutzt ihn noch nicht. Die Implementation auf dieser Folie sollte nach dem bisher Gesagten selbsterklärend sein.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   Frame frame;
   PrintStream pout;
   public QuitListener ( Frame frame, PrintStream pout ) {
      this.frame = frame;
      this.pout = pout;
   }
   .........
}
```

Dies ist schon das angekündigte zweite Beispiel.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   Frame frame;
   PrintStream pout;
   public QuitListener ( Frame frame, PrintStream pout ) {
      this.frame = frame;
      this.pout = pout;
   }
   ..........
}
```

Eine weitere selbstgebaute ActionListener-Klasse. Die Methode actionPerformed soll jetzt das Fenster schließen und den Zeitpunkt, zu dem sie aufgerufen wurde, protokollieren.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   Frame frame;
   PrintStream pout;
   public QuitListener ( Frame frame, PrintStream pout ) {
      this.frame = frame;
      this.pout = pout;
   }
   .........
}
```

Ein Verweis auf das Fenster muss auch hier wieder als Attribut gespeichert werden, um in Methode actionPerformed darauf zuzugreifen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   Frame frame;
   PrintStream pout;
   public QuitListener ( Frame frame, PrintStream pout ) {
      this.frame = frame;
      this.pout = pout;
   }
   .........
}
```

Wie bisher auch und wie es guter Praxis entspricht, lassen wir offen, auf welchen PrintStream der Zeitpunkt geschrieben werden soll.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   .........
   public void actionPerformed ( ActionEvent event ) {
      frame.dispose();
      Timestamp stamp = new Timestamp ( event.getWhen() );
      pout.print ( stamp.getHour() + ":" + stamp.getMinute() );
   }
}
```

Jetzt fehlt nur noch die Methode actionPerformed selbst.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   ..........
   public void actionPerformed ( ActionEvent event ) {
     frame.dispose();
     Timestamp stamp = new Timestamp ( event.getWhen() );
     pout.print ( stamp.getHour() + ":" + stamp.getMinute() );
   }
}
```

Das ist die Hauptaktion: Das Fenster wird geschlossen, und alle von ihm potentiell reservierten Ressourcen werden wieder freigegeben.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   .........
   public void actionPerformed ( ActionEvent event ) {
      frame.dispose();
      Timestamp stamp = new Timestamp ( event.getWhen() );
      pout.print ( stamp.getHour() + ":" + stamp.getMinute() );
   }
}
```

Alle Arten von Events, und so auch Klasse ActionEvent, haben eine Methode getWhen, die als long-Wert den Zeitpunkt zurückliefert, zu dem der Event geschehen ist. Konkret hier ist das also der Zeitpunkt, zu dem der Klick auf den Button vom Event Dispatch Thread registriert wurde. Genauer gesagt, ist für diesen Zeitpunkt die Zeitdifferenz seit Beginn des Jahres 1970 kodiert, und zwar in Millisekunden.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   .........
   public void actionPerformed ( ActionEvent event ) {
      frame.dispose();
      Timestamp stamp = new Timestamp ( event.getWhen() );
      pout.print ( stamp.getHour() + ":" + stamp.getMinute() );
   }
}
```

Klasse Timestamp in Package java.sql nimmt uns die Arbeit ab, diese long-Zahl in die übliche Darstellung mit Jahr, Monat, Tag, Stunde, Minute, Sekunde und Millisekunde umzurechnen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   .........
   public void actionPerformed ( ActionEvent event ) {
      frame.dispose();
      Timestamp stamp = new Timestamp ( event.getWhen() );
      pout.print ( stamp.getHour() + ":" + stamp.getMinute() );
   }
}
```

Dies sind zwei Beispiele für die Methoden von Timestamp, mit denen man die einzelnen Bestandteile derjenigen Zeitangabe erhalten kann, die im Konstruktor von Timestamp als long übergeben wurde; sollte eigentlich selbsterklärend sein.

***Erinnerung*: Der Plusoperator bedeutet Konkatenation bei zwei Strings beziehungsweise so wie hier bei einem String und einem Wert eines primitiven Datentyps.**

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitListener implements ActionListener {
   ..........
   public void actionPerformed ( ActionEvent event ) {
      frame.dispose();
      Timestamp stamp = new Timestamp ( event.getWhen() );
      pout.print ( stamp.getHour() + ":" + stamp.getMinute() );
   }
}
```

Wohin die Daten geschrieben werden, also mit welcher Datensenke der PrintStream verbunden ist, war ja im Konstruktor offengelassen und als Attribut pout im QuitListener-Objekt gespeichert worden.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
ActionListener listener1
   = new BackgroundColorListener ( frame, new Color ( 20, 30, 40 ) ) );
ActionListener listener2 = new QuitListener ( frame, System.err );
button1.addActionListener ( listener1 );
button2.addActionListener ( listener2 );
frame.add ( button1 );
frame.add ( button2 );
```

Jetzt ein Beispiel mit *zwei* Buttons und entsprechenden Listenern.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
ActionListener listener1
    = new BackgroundColorListener ( frame, new Color ( 20, 30, 40 ) ) );
ActionListener listener2 = new QuitListener ( frame, System.err );
button1.addActionListener ( listener1 );
button2.addActionListener ( listener2 );
frame.add ( button1 );
frame.add ( button2 );
```

Diese Schritte für den ersten Button hatten wir schon auf einer früheren Folie gesehen; das ist nichts Neues.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
ActionListener listener1
   = new BackgroundColorListener ( frame, new Color ( 20, 30, 40 ) ) );
ActionListener listener2 = new QuitListener ( frame, System.err );
button1.addActionListener ( listener1 );
button2.addActionListener ( listener2 );
frame.add ( button1 );
frame.add ( button2 );
```

Jetzt ein zweiter Button mit dem QuitListener. Wie Sie sehen, ist eigentlich alles absolut parallel.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button1 = new Button ( “Change Background Color“ );
Button button2 = new Button ( “Quit“ );
ActionListener listener1
   = new BackgroundColorListener ( frame, new Color ( 20, 30, 40 ) ) );
ActionListener listener2 = new QuitListener ( frame, System.err );
button1.addActionListener ( listener1 );
button2.addActionListener ( listener2 );
frame.add ( button1 );
frame.add ( button2 );
```

Als PrintStream wählen wir den Standard Error Stream.

Erinnerung: Kapitel 08, Abschnitt zu Bytedaten.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   private Button quitButton;
   public QuitFrame () {
      super ( “Hello World“ );
      quitButton = new Button ( “Quit“ );
      add ( quitButton );
      button.addActionListener ( new QuitGoodByeListener() );
   }
   .........
}
```

Wenn auf einen Button geklickt wird, dann soll oft ja auch der Button selbst sich ändern, beispielsweise einen anderen Text bekommen. Dieses Ziel ist eine gute Gelegenheit, um im nächsten Beispiel die Arbeit mit Frames jetzt zu verallgemeinern. Der typische Umgang mit Klasse Frame ist eigentlich eher so, wie Sie es *hier* sehen, ...

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
    private Button quitButton;
    public QuitFrame () {
        super ( “Hello World“ );
        quitButton = new Button ( “Quit“ );
        add ( quitButton );
        button.addActionListener ( new QuitGoodByeListener() );
    }
    .....
}
```

... nämlich indem man eine Klasse von Frame ableitet. Diese neue Klasse bekommt alle Funktionalität von Klasse Frame vererbt, aber man kann noch Funktionalität hinzufügen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
  private Button quitButton;
  public QuitFrame () {
    super ( “Hello World“ );
    quitButton = new Button ( “Quit“ );
    add ( quitButton );
    button.addActionListener ( new QuitGoodByeListener() );
  }
  ..........
}
```

Zum Beispiel können wir Verweise auf die Komponenten, die wir ins Fenster packen, für späteren Zugriff als zusätzliche Attribute speichern.

Wenn auf den Button geklickt wird, dann soll Folgendes passieren: Zuerst soll „Good bye“ anstelle von „Quit“ auf dem Button angezeigt werden, und erst zwei Sekunden später soll sich dann das Fenster schließen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   private Button quitButton;
   public QuitFrame () {
      super ( “Hello World“ );
      quitButton = new Button ( “Quit“ );
      add ( quitButton );
      button.addActionListener ( new QuitGoodByeListener() );
   }
   .........
}
```

Selbstverständlich kann der ActionListener auch registriert werden, *nachdem* der Button im Fenster eingefügt wurde. Die Klasse QuitGoodByeListener müssen wir noch schreiben, dann ist das Beispiel vollständig.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   private Button quitButton;
   public QuitFrame () {
      super ( “Hello World“ );
      quitButton = new Button ( “Quit“ );
      add ( quitButton );
      button.addActionListener ( new QuitGoodByeListener() );
   }
   .....
}
```

Aber vorher machen wir uns klar, dass die Situation jetzt offenbar grundlegend anders sein muss: Weder das Fenster noch der Button werden als Parameter beim Konstruktoraufruf übergeben, obwohl der Listener ja auf beide zugreifen soll: auf den Button, um seinen Text zu ändern, und auf das Fenster, um es zu schließen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   .........
   private class QuitGoodByeListener implements ActionListener {
      public void actionPerformed ( ActionEvent event ) {
         quitButton.setLabel ( “Good bye!“ );
         thread.sleep ( 2000 );
         dispose();
       }
   }
   .........
}
```

Des Rätsels Lösung: QuitGoodByeListener ist in QuitFrame eingebettet.

***Erinnerung*: In Kapitel 09 gab es einen Abschnitt zu verschachtelten Klassen, also Klassen wie QuitGoodByeListener, die innerhalb von anderen Klassen wie hier QuitFrame definiert sind.**

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
    ..........
    private class QuitGoodByeListener implements ActionListener {
        public void actionPerformed ( ActionEvent event ) {
            quitButton.setLabel ( "Good bye!" );
            thread.sleep ( 2000 );
            dispose();
        }
    }
    ..........
}
```

Wir haben bei der Einführung von verschachtelten Klassen in Kapitel 09 gesehen, dass ein Objekt der inneren Klasse innerhalb der äußeren Klasse ganz normal mit new und Klassennamen GoodByeListener erzeugt werden kann. Auf der Eingangsfolie zu diesem Beispiel haben wir das Listener-Objekt genau so im Konstruktor von Klasse Frame erzeugt, das passt also. Wir wissen auch, dass das so erzeugte Objekt der inneren Klasse GoodByeListener mit dem Objekt der äußeren Klasse Frame verbunden ist und auf dessen Attribute und Methoden zugreifen kann, als wären es die eigenen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   ..........
   private class QuitGoodByeListener implements ActionListener {
      public void actionPerformed ( ActionEvent event ) {
         quitButton.setLabel ( "Good bye!" );
         thread.sleep ( 2000 );
         dispose();
      }
   }
   ..........
}
```

Hier wird also die von Klasse Frame ererbte Methode dispose mit dem QuitFrame-Objekt aufgerufen, bei dessen Konstruktion das QuitGoodByeListener-Objekt erzeugt wurde.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   .........
   private class QuitGoodByeListener implements ActionListener {
      public void actionPerformed ( ActionEvent event ) {
         quitButton.setLabel ( “Good bye!“ );
         thread.sleep ( 2000 );
         dispose();
       }
   }
   .........
}
```

Und hier wird eben auf ein Attribut dieses Frame-Objektes zugegriffen.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class QuitFrame extends Frame {
   .........
   private class QuitGoodByeListener implements ActionListener {
      public void actionPerformed ( ActionEvent event ) {
         quitButton.setLabel ( “Good bye!“ );
         thread.sleep ( 2000 );
         dispose();
       }
   }
   .........
}
```

Bisher hatten wir den Text auf einem Button immer nur mit dem Konstruktor der Klasse Button gesetzt. Die Methode setLabel von Klasse Button erlaubt das spätere Überschreiben dieses Textes mit dem aktualen Parameter, natürlich beliebig oft.

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button = new Button ( “Hello“ );
frame.add ( button );
button.addActionListener ( ( e ) -> { System.out.print ( “Hello“ ); } );
```

Zum Abschluss dieses Abschnitts noch eine illustrative Beispielvariation mit einem Lambda-Ausdruck.

***Erinnerung*: Kapitel 04c, Abschnitt zu Functional Interfaces und Lambda-Ausdrücken in Java.**

Button und ActionListener

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( new String ( “Hello World“ ) );
Button button = new Button ( “Hello“ );
frame.add ( button );
button.addActionListener ( ( e ) -> { System.out.print ( “Hello“ ); } );
```

Interface ActionListener ist ja ein funktionales Interface, kann also durch einen Lambda-Ausdruck initialisiert werden, der auf Methode actionPerformed passt, also Rückgabetyp void und einen Parameter hat, dessen formaler Typ ActionEvent vom Compiler selbst bestimmt werden kann, ohne dass wir ihn hinschreiben müssten.

***Erinnerung*: Auch diese automatische Typbestimmung durch den Compiler haben wir in Kapitel 04c, Abschnitt zu Functional Interfaces und Lambda-Ausdrücken, gesehen.**

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere Listener- und Event-Typen

Für die meisten Zwecke, in denen Listener und Events eingesetzt werden sollen, sind ActionListener und ActionEvent zu einfach gestrickt. Daher gibt es diverse weitere Listener-Interfaces und Event-Klassen in der Java-Standardbibliothek, die auf verschiedene Situationen sind und beileibe nicht alle mit GUIs zu tun haben. Wir schauen uns beispielhaft die Listener-Interfaces und Event-Klassen in Package java.awt an.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

KeyListener	KeyEvent
MouseListener	MouseEvent
MouseMotionListener	MouseEvent
MouseWheelListener	MouseWheelEvent
WindowFocusListener	WindowEvent
WindowListener	WindowEvent
WindowStateListener	WindowEvent

Hier sehen Sie links alle Listener-Interfaces und rechts die jeweils zugehörigen Event-Klassen. Wir sehen uns das gleich im Detail an. Hier können wir aber schon einmal festhalten, ...

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

KeyListener	**KeyEvent**
MouseListener	MouseEvent
MouseMotionListener	MouseEvent
MouseWheelListener	MouseWheelEvent
WindowFocusListener	WindowEvent
WindowListener	WindowEvent
WindowStateListener	WindowEvent

... es gibt ein Listener-Interface und eine Event-Klasse für die Tastatur, englisch keyboard, ...

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

KeyListener	KeyEvent
MouseListener	MouseEvent
MouseMotionListener	MouseEvent
MouseWheelListener	MouseWheelEvent
WindowFocusListener	WindowEvent
WindowListener	WindowEvent
WindowStateListener	WindowEvent

... drei Listener-Interfaces mit zugehörigen Event-Klassen für Nutzeraktionen mit der Maus, wobei man sich dafür entschieden hat, dass eine Event-Klasse für MouseListener und MouseMotionListener ausreicht, MouseWheelListener aber eine eigene Event-Klasse benötigt, ...

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

KeyListener	KeyEvent
MouseListener	MouseEvent
MouseMotionListener	MouseEvent
MouseWheelListener	MouseWheelEvent
WindowFocusListener	WindowEvent
WindowListener	WindowEvent
WindowStateListener	WindowEvent

... und schließlich noch drei Listener-Interfaces und eine Event-Klasse für Nutzerinteraktionen, die das Fenster als Ganzes betreffen.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.addKeyListener ( new MyKeyListener ( ..... ) );
frame.addMouseListener ( new MyMouseListener ( ..... ) );
frame.addWindowListener ( new MyWindowListener ( ..... ) );
```

Für jede Art von Listener gibt es eine eigene Registrierungsmethode. Der Name ist immer derselbe wie das Listener-Interface, nur ein add davor. Jede dieser Methoden hat genau einen Parameter, und dessen formaler Typ ist das zugehörige Listener-Interface.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.addKeyListener ( new MyKeyListener ( ..... ) );
frame.addMouseListener ( new MyMouseListener ( ..... ) );
frame.addWindowListener ( new MyWindowListener ( ..... ) );
```

Wie bei ActionListener, werden wir auch bei anderen Listener-Interfaces eigene Klassen definieren, die das jeweilige Interface implementieren, und deren Konstruktoren dann jeweils geeignete Parameterlisten haben werden.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface KeyListener {
   public void keyPressed ( KeyEvent event );
   public void keyReleased ( KeyEvent event );
   public void keyTyped ( KeyEvent event );
}
```

Zuerst zum KeyListener, der die Tastatur abhorcht.

Mit einer Taste auf der Tastatur kann man im Prinzip drei Dinge tun: Man kann sie herunterdrücken und dann gedrückt halten, man kann sie wieder loslassen, und man kann sie kurz antippen. Der Event Dispatch Thread unterscheidet zwischen diesen drei Fällen und ruft die entsprechende Methode auf.

Beachten Sie, dass KeyListener – wie wohl die meisten Listener-Interfaces in der Java-Standardbibliothek – *kein* Functional Interface ist.

Jede der drei Methoden können Sie sich wie actionPerformed bei ActionListener vorstellen, nur dass sie eben in jeweils spezifischen Situationen aufgerufen werden.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Dies ist ein schematisches, rein illustratives Beispiel für eine selbstgebaute KeyListener-Klasse. Phantasielos, wie wir sind, nennen wir sie einfach MyKeyListener.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Zuallererst: Zu jedem Listener-Interface in der Java-Standardbibliothek, das kein Functional Interface ist, gibt es eine zugehörige Adapter-Klasse. Die Adapter-Klasse zu KeyListener heißt KeyAdapter. Eine solche Adapter-Klasse implementiert das zugehörige Listener-Interface, aber sämtliche Methoden sind leer, das heißt, enthalten keinerlei Anweisungen.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Der Sinn dahinter zeigt sich in diesem Beispiel: Wir wollen eine KeyListener-Klasse schreiben, die darauf reagiert, dass eine Taste heruntergedrückt und untengehalten wird, auf die beiden anderen Fälle soll ein Listener dieser Klasse nicht reagieren. Wenn wir nun die Klasse MyKeyListener direkt das Interface KeyListener implementieren lassen würden, müssten wir auch die anderen beiden Methoden implementieren, ebenfalls ohne Anweisungen. Genau das erhalten wir aber durch Vererbung geschenkt, wenn wir die Klasse MyKeyListener statt dessen von KeyAdapter ableiten. Die Adapter-Klassen zu den Listener-Interfaces bieten also ein kleines Stück Bequemlichkeit. Bei Functional Interfaces wie ActionListener machen solche Adapter natürlich *keinen* Sinn, daher gibt es sie dafür nicht.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

***Nebenbemerkung*: KeyAdapter und die Adapter-Klassen für andere Listener-Interfaces sind abstract, obwohl sie keine abstrakten Methoden enthalten. In Kapitel 02, Abschnitt zu abstrakten Methoden und Klassen, hatten wir schon darauf hingewiesen. Hintergrund ist, dass ein Listener-Objekt, dessen Methoden allesamt keine Anweisungen enthalten, keinen Effekt hat und die Einrichtung eines solchen Objektes sicherlich auf einem falschen Gedanken beruhen würde.**

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Erinnerung: In Kapitel 03c hatten wir die switch-Anweisung in einem gleichnamigen Abschnitt kennen gelernt.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Klasse KeyEvent hat unter anderem diese Methode, die eine Kodierung der gedrückten Taste zurückliefert.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Für jede Taste hat Klasse KeyEvent eine identifizierende Klassenkonstante vom Typ int, hier nur drei Beispiele.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Das ist beispielsweise die Kodierung für den Buchstaben A.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Das hingegen ist die Kodierung des Doppelpunktes.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyKeyListener extends KeyAdapter {
   public void keyPressed ( KeyEvent event ) {
      switch ( event.getKeyCode() ) {
         case KeyEvent.VK_A: ..... break;
         case KeyEvent.VK_COLON: ..... break;
         case KeyEvent.VK_BACKSPACE: ..... break;
      }
   }
}
```

Und das schließlich ist die Kodierung für die Enter-Taste. Wie Sie sehen, haben auch Funktionstasten eine Kodierung.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface MouseListener {
    public void mouseClicked ( MouseEvent event );
    public void mousePressed ( MouseEvent event );
    public void mouseReleased ( MouseEvent event );
    public void mouseEntered ( MouseEvent event );
    public void mouseExited ( MouseEvent event );
}
```

Wir machen weiter mit dem Interface MouseListener.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface MouseListener {
  public void mouseClicked ( MouseEvent event );
  public void mousePressed ( MouseEvent event );
  public void mouseReleased ( MouseEvent event );
  public void mouseEntered ( MouseEvent event );
  public void mouseExited ( MouseEvent event );
}
```

Wie bei der Tastatur gibt es auch bei den Mausbuttons drei unterschiedliche Fälle: erstens kurz klicken, zweitens herunterdrücken und dann gedrückt halten sowie drittens wieder loslassen.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface MouseListener {
   public void mouseClicked ( MouseEvent event );
   public void mousePressed ( MouseEvent event );
   public void mouseReleased ( MouseEvent event );
   public void mouseEntered ( MouseEvent event );
   public void mouseExited ( MouseEvent event );
}
```

Hinzu kommen noch zwei Fälle, die es bei der Tastatur nicht gibt, nämlich dass der Mauszeiger den Bereich, den der MouseListener abhorcht, entweder betritt oder verlässt.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface MouseMotionListener {
   public void mouseDragged ( MouseEvent event );
   public void mouseMoved ( MouseEvent event );
}

public interface MouseWheelListener {
   public void mouseWheelMoved ( MouseWheelEvent event );
}
```

Hier kurz die anderen beiden Listener-Interfaces für Interaktionen mit der Maus. Die Namen der Methoden sollten selbsterklärend sein.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMouseListener extends MouseAdapter {
   public void mouseClicked ( MouseEvent event ) {
      if ( event.getButton() == MouseEvent.BUTTON1 )
         System.err ( "(" + event.getX() + "," + event.getY() + ")" );
   }
   public void MouseWheelMoved ( MouseWheelEvent event ) {
      System.out ( event.getWheelRotation() );
   }
}
```

Ein kleines, rein illustratives Beispiel für eine Klasse, die MouseListener implementiert.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMouseListener extends MouseAdapter {
   public void mouseClicked ( MouseEvent event ) {
      if ( event.getButton() == MouseEvent.BUTTON1 )
         System.err ( "(" + event.getX() + "," + event.getY() + ")" );
   }
   public void MouseWheelMoved ( MouseWheelEvent event ) {
      System.out ( event.getWheelRotation() );
   }
}
```

Beziehungsweise wir leiten wieder von der zugehörigen Adapter-Klasse ab, da wir nicht auf alle potentiell auftretenden Fälle reagieren und daher nicht alle Methoden eigenhändig implementieren wollen.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMouseListener extends MouseAdapter {
   public void mouseClicked ( MouseEvent event ) {
      if ( event.getButton() == MouseEvent.BUTTON1 )
         System.err ( "(" + event.getX() + "," + event.getY() + ")" );
   }
   public void MouseWheelMoved ( MouseWheelEvent event ) {
      System.out ( event.getWheelRotation() );
   }
}
```

Die Klasse MouseAdapter steht allerdings nicht in Eins-zu-Eins-Korrespondenz zu einem einzelnen Listener-Interface, sondern implementiert MouseListener, MouseMotionListener und MouseWheelListener zugleich. Auch hier sind alle Methoden schon in der Adapter-Klasse implementiert, aber halt ohne Anweisungen.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMouseListener extends MouseAdapter {
   public void mouseClicked ( MouseEvent event ) {
      if ( event.getButton() == MouseEvent.BUTTON1 )
         System.err ( "(" + event.getX() + "," + event.getY() + ")" );
   }
   public void MouseWheelMoved ( MouseWheelEvent event ) {
      System.out ( event.getWheelRotation() );
   }
}
```

Bei einem MouseEvent kann unter anderem abgefragt werden, auf welchen Button der Maus gedrückt wurde.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMouseListener extends MouseAdapter {
   public void mouseClicked ( MouseEvent event ) {
      if ( event.getButton() == MouseEvent.BUTTON1 )
         System.err ( "(" + event.getX() + "," + event.getY() + ")" );
   }
   public void MouseWheelMoved ( MouseWheelEvent event ) {
      System.out ( event.getWheelRotation() );
   }
}
```

... sowie auch die Koordinaten relativ zum Ursprung des Bereichs auf dem Bildschirm, den der MouseListener abhorcht. Der Ursprung, also der Punkt mit x- und y-Koordinate 0, ist immer die linke obere Ecke, und die Koordinaten wachsen nach rechts und nach unten.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMouseListener extends MouseAdapter {
   public void mouseClicked ( MouseEvent event ) {
      if ( event.getButton() == MouseEvent.BUTTON1 )
         System.err ( "(" + event.getX() + "," + event.getY() + ")" );
   }
   public void MouseWheelMoved ( MouseWheelEvent event ) {
      System.out ( event.getWheelRotation() );
   }
}
```

Nur beispielhaft eine Methode von MouseWheelEvent. Mit getWheelRotation bekommt man zurückgeliefert, um wie viele Ticks das Mausrad bei der Nutzeraktion, durch die das Event ausgelöst wurde, gedreht wurde. (Bei heutigen Mäusen sind Begriffe wie „Rad“ und „Tick“ natürlich metaphorisch zu verstehen.)

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface WindowListener {
   public void windowOpened ( WindowEvent event );
   public void windowClosing ( WindowEvent event );
   public void windowClosed ( WindowEvent event );
   public void windowActivated ( WindowEvent event );
   public void windowDeactivated ( WindowEvent event );
   public void windowIconified ( WindowEvent event );
   public void windowDeiconified ( WindowEvent event );
}
```

Nun noch zu den Listener-Interfaces für ganze Fenster. Die Namen der Methoden von Interface WindowListener sollten eigentlich selbsterklärend sein.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface WindowListener {
    public void windowOpened ( WindowEvent event );
    public void windowClosing ( WindowEvent event );
    public void windowClosed ( WindowEvent event );
    public void windowActivated ( WindowEvent event );
    public void windowDeactivated ( WindowEvent event );
    public void windowIconified ( WindowEvent event );
    public void windowDeiconified ( WindowEvent event );
}
```

Nur ein Detail: Wenn ein Fenster die Anweisung bekommen hat, sich zu schließen, wird ein erstes Event ausgelöst und von Methode windowClosing behandelt; sobald das Fenster dann wirklich geschlossen ist, wird dann noch windowClosed mit einem zweiten Event aufgerufen.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface WindowStateListener {
   public void windowStateChanged ( WindowEvent event );
}

public interface WindowFocusListener {
   public void windowGainedFocus ( WindowEvent event );
   public void windowLostFocus ( WindowEvent event );
}
```

Wir streifen nur ganz kurz die beiden weiteren fensterbezogenen Listener-Interfaces. Auch hier sollten die Namen der Methoden selbsterklärend sein.

Weitere Listener und Events

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract public class KeyAdapter implements KeyListener

abstract public class MouseAdapter
    implements MouseListener,
               MouseMotionListener,
               MouseWheelListener

abstract public class WindowAdapter
    implements WindowFocusListener,
               WindowListener,
               WindowStateListener
```

Hier noch einmal die Beziehung zwischen Listener-Interfaces und Adapter-Klassen im Überblick. Nach dem bisher Gesagten sollte auch diese Folie selbsterklärend sein.

TECHNISCHE UNIVERSITÄT DARMSTADT

Andere Typen von Komponenten in einem Fenster

Klasse Button war nur unser erstes Beispiel für Komponenten von Fenstern. Jetzt schauen wir uns die weiteren Komponententypen an, die das Package java.awt bereitstellt.

Andere Komponententypen

TECHNISCHE UNIVERSITÄT DARMSTADT

Button	Label
Canvas	List
Checkbox	Scrollbar
Choice	TextComponent

Das sind die acht Klassen für verschiedene Komponententypen in Package java.awt. Klasse Button hatten wir schon gesehen.

Andere Komponententypen

TECHNISCHE UNIVERSITÄT DARMSTADT

Button	Label
Canvas	List
Checkbox	Scrollbar
Choice	TextComponent

Bei diesen beiden Klassen wird uns auch ein weiteres Listener-Interface mit Namen ItemListener begegnen, ...

Andere Komponententypen

TECHNISCHE UNIVERSITÄT DARMSTADT

Button	Label
Canvas	List
Checkbox	Scrollbar
Choice	TextComponent

... bei Klasse Scrollbar ein Listener-Interface namens AdjustmentListener ...

Andere Komponententypen

TECHNISCHE UNIVERSITÄT DARMSTADT

Button	Label
Canvas	List
Checkbox	Scrollbar
Choice	TextComponent

... und bei Klasse TextComponent dann TextListener.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyCanvas extends Canvas {
   public void paint ( Graphics graphics ) {
      .....
   }
}
```

```
frame.add ( new MyCanvas() );
```

Zuerst zu Klasse Canvas. Canvas ist das englische Wort unter anderem für Leinwand im Sinne von Zeichenfläche, und genau das bietet Klasse Canvas: eine abgegrenzte Zeichenfläche in einem Fenster.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyCanvas extends Canvas {
   public void paint ( Graphics graphics ) {
      ..........
   }
}
```

```
frame.add ( new MyCanvas() );
```

Ein Objekt der Klasse Canvas wird exakt genauso zu einem Fenster hinzugefügt wie ein Button-Objekt.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyCanvas extends Canvas {
    public void paint ( Graphics graphics ) {
        ..........
    }
}
```

```
frame.add ( new MyCanvas() );
```

Von Klasse Canvas wird typischerweise abgeleitet, ...

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyCanvas extends Canvas {
  public void paint ( Graphics graphics ) {
      ..........
  }
}
```

```
frame.add ( new MyCanvas() );
```

... weil die zentrale Methode von Canvas, paint, in Klasse Canvas leergelassen ist. Man leitet von Klasse Canvas ab, um paint mit der eigenen Zeichenlogik zu überschreiben. Weitere Methoden von Canvas brauchen wir in diesem kleinen Beispiel nicht zu überschreiben.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyCanvas extends Canvas {
   public void paint ( Graphics graphics ) {
      ..........
   }
}
```

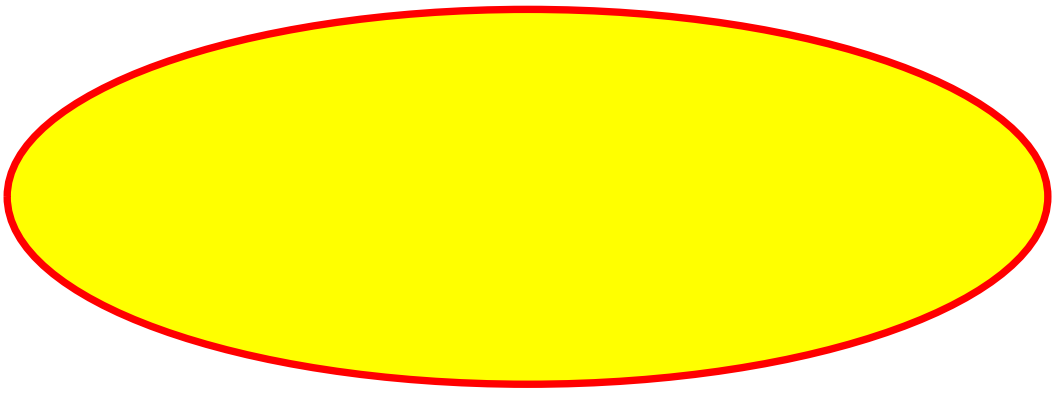
```
frame.add ( new MyCanvas() );
```

Erinnerung: Im Fallbeispiel GeomShape2D in Kapitel 02 hatten wir schon Klasse Graphics aus Package java.awt gesehen. Ein Objekt der Klasse Graphics ist allgemein mit einer Zeichenfläche verbunden, und diese Zeichenfläche wird durch Aufrufe der Methoden von Klasse Graphics mit Inhalt gefüllt.

Ein Canvas-Objekt erzeugt eine Zeichenfläche in dem Fenster, zu dem es addiert wurde. Alle Anweisungen zum Zeichnen, die an graphics gehen, werden auf dieser Zeichenfläche realisiert.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT



That‘s what an ellipse looks like!

Die Fläche zwischen den beiden schwarzen, waagrechten Linien soll die Zeichenfläche sein. Wir wollen eine Ellipse und einen Text so darin zeichnen, dass sie in der Horizontalen mittig platziert sind. Zu allen vier Rändern soll der Abstand des Gesamtbildes 10% der Breite beziehungsweise Höhe der Zeichenfläche sein. Auch zwischen Ellipse und Text soll der Abstand 10% der Gesamthöhe betragen. Diese Regeln sollen bei jeder beliebigen Breite und Höhe erfüllt werden.

***Achtung*: Die Zeichnung ist nicht maßstabsgetreu!**

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Der Inhalt der Methode paint, wie wir sie hier beispielhaft implementieren wollen, passt nicht so recht auf eine Folie. Hier sehen Sie erst einmal den ersten Teil der Anweisungen.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Mit der Objektmethode getFontMetrics erhält man von Graphics Informationen über die momentan auf der Zeichenfläche festgelegte Schriftart und Schriftgröße.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Diese beiden Werte geben an, wie weit Text in dieser Schriftart und Schriftgröße vertikal nach oben beziehungsweise nach unten von der Basislinie des Textes aus reichen kann. Die Summe aus beiden Werten ist natürlich die maximale vertikale Ausdehnung, in der ein Text garantiert verbleibt.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Klasse Rectangle in Package java.awt wird häufig nicht nur für Rechtecke als Zeichenobjekte benutzt, sondern auch für Umrissangaben. Das von Methode getClipBounds zurückgelieferte Rectangle-Objekt beschreibt das Zeichenfenster. Die Attribute x und y geben die Koordinaten des Ursprungs oben links an, und wie es die Namen schon sagen, geben width und height die Breite und die Höhe an.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Wir wollen ja beispielhaft eine achsenparallele Ellipse in die Zeichenfläche setzen. Allerdings wollen wir zu jedem Rand ungefähr zehn Prozent der Zeichenfläche freilassen. Daher addieren wir zehn Prozent der Breite der Zeichenfläche auf den x-Koordinatenwert des linken Randes der Zeichenfläche, um die x-Koordinate am linken Ende der Ellipse zu berechnen.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Und entsprechend umfasst die Breite der Ellipse ungefähr achtzig Prozent der Breite der Zeichenfläche.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Analog lassen wir oben zehn Prozent der Fensterhöhe frei.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
FontMetrics fontMetrics = graphics.getFontMetrics();
final int maxStringHeight = fontMetrics.getMaxAscent() + fontMetrics.getMaxDescent();
Rectangle area = graphics.getClipBounds();
final double marginFraction = 0.1;
final int horizontalMargin = (int) ( area.width * marginFraction );
final int verticalMargin = ( int ) ( area.height * marginFraction );
final int leftMarginOfEllipse = area.x + horizontalMargin;
final int widthOfEllipse = area.width – 2 * horizontalMargin;
final int topMarginOfEllipse = area.y + verticalMargin;
final int heightOfEllipse = area.height – 3 * verticalMargin – maxStringHeight;
```

Bei der vertikalen Höhe der Ellipse müssen wir aber noch etwas beachten. Wir wollen ja innerhalb der Zeichenfläche noch einen Untertitel unter die Ellipse setzen und müssen dafür entsprechend noch vertikalen Platz reservieren. Zudem wollten wir noch einen vertikalen Abstand von 10% der Höhe der Zeichenfläche zwischen der Ellipse und dem Text lassen, also insgesamt dreimal statt zweimal verticalMargin.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
graphics.setColor ( Color.YELLOW );
graphics.fillOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
graphics.setColor ( Color.RED );
graphics.drawOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
```

Jetzt sind wir soweit, dass wir die gelbrote Ellipse zeichnen können.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
graphics.setColor ( Color.YELLOW );
graphics.fillOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
graphics.setColor ( Color.RED );
graphics.drawOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
```

Dazu setzen wir die Zeichenfarbe zuerst auf Gelb und nach dem Zeichnen der Innenfläche dann auf Rot.

***Erinnerung*: Genau so sind wir in Kapitel 02 vorgegangen.**

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
graphics.setColor ( Color.YELLOW );
graphics.fillOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
graphics.setColor ( Color.RED );
graphics.drawOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
```

Wie ebenfalls aus Kapitel 02 bekannt, sind das die beiden Methoden von Klasse Graphics, um den Inhalt beziehungsweise den Rand einer Ellipse zu zeichnen.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
graphics.setColor ( Color.YELLOW );
graphics.fillOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
graphics.setColor ( Color.RED );
graphics.drawOval
    ( leftMarginOfEllipse, topMarginOfEllipse, widthOfEllipse, heightOfEllipse );
```

Und das sind die vier notwendigen Angaben. Die letzten beiden aktualen Parameter sind sicherlich selbsterklärend; die ersten beiden sind die x- und die y-Koordinate des Referenzpunktes der Ellipse.

***Erinnerung*: Den Referenzpunkt bei diversen geometrischen Objektarten – darunter auch Ellipsen – hatten wir im Fallbeispiel GeomShape2D in Kapitel 02 gesehen: die obere linke Ecke des kleinsten umschließenden achsenparallelen Rechtecks.**

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Nach dem Zeichnen der Ellipse müssen wir noch den Untertitel geeignet platzieren und zeichnen.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Das soll also der Inhalt des Untertitels werden.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Wir wollen den Untertitel horizontal zentriert platzieren. Dazu müssen wir wissen, wie breit er sein wird. Klasse FontMetrics hat genau für solche Fälle eine Objektmethode stringWidth. Wir hatten das FontMetrics-Objekt ja eben vom Graphics-Objekt via Methode getFontMetrics erhalten, so dass die momentan im Graphics-Objekt eingestellte Schriftart und Schriftgröße für die Berechnung verwendet werden.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Der Wert in x soll die x-Koordinate werden, an der der String beginnt, also die linke Begrenzung des Strings. Daher addieren wir zunächst die Hälfte der Breite der Zeichenfläche auf die linke Begrenzung der Zeichenfläche, das ergibt die x-Koordinate in der Mitte der Zeichenfläche. Davon ziehen wir die Hälfte der Breite des Untertitels ab. Die beiden dafür notwendigen Divisionen durch 2 sind hier mittels Distributivgesetz zusammengefasst (wobei streng genommen das Distributivgesetz bei ganzzahliger Division mit Rest nicht gilt, aber für unsere rein demonstrativen Zwecke passt das schon).

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Die y-Koordinate wird die Höhe der Basislinie des Untertitels sein. Da der Untertitel ganz unten in die Zeichenfläche platziert werden soll, bietet es sich an, vom unteren Rand der Zeichenfläche aus zu rechnen. Also addieren wir zuerst die Höhe der Zeichenfläche auf ihre *oberste* vertikale Koordinate, um die *unterste* vertikale Koordinate der Zeichenfläche zu erhalten. Damit der Untertitel garantiert nicht über den unteren Rand der Zeichenfläche hinausgeht, muss die Basislinie um den Wert von getMaxDescent erhöht werden, dieser Wert ist also abzuziehen, natürlich auch der Abstand zum unteren Rand der Zeichenfläche.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Nicht vergessen, die Farbe noch umzustellen.

Canvas

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final String subtitle = new String ( “That\‘s what an ellipse looks like!“ );
final int widthOfString = fontMetrics.stringWidth ( subtitle );
final int x = area.x + ( area.width – widthOfString ) / 2;
final int y = area.y + area.height – verticalMargin – fontMetrics.getMaxDescent();
graphics.setColor ( Color.BLACK );
graphics.drawString ( subtitle, x, y );
```

Jetzt sind wir soweit, dass wir den Untertitel mit Methode drawString tatsächlich an die richtige Position zeichnen können. Damit ist das Beispiel für Klasse Canvas beendet, und wir können uns der nächsten Art von graphischen Komponenten zuwenden.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Als nächstes dann Klasse Checkbox, wieder anhand eines kleinen Beispiels. Eine Checkbox besteht aus einem sehr kleinen Button, oft auch Pin genannt, und einem meist sehr kurzen Text unmittelbar daneben, darüber oder darunter. Der Button hat zwei graphisch unterschiedlich dargestellte Zustände, die man als an- oder ausgeschaltet interpretieren kann.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Über eine Checkbox wollen wir eine Maschine manuell an- und ausschalten können. Diese Maschine sei im Java-Programm durch ein Objekt einer Klasse repräsentiert, die wir phantasielos SomeKindOfMachine nennen. Dieses Objekt ist über irgendeinen Kommunikationskanal, der im Konstruktor spezifiziert werden müsste und hier nicht weiter interessiert, mit der eigentlichen Maschine in der realen Welt verbunden.

Der Einfachheit halber gehen wir davon aus, dass die Maschine nicht an der Checkbox vorbei an- und ausgehen kann, das heißt, das hier kurz angedeutete Java-Programm hat volle Kontrolle darüber, ob die Maschine an oder aus ist.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Wenn die Maschine an ist, soll die Checkbox durch ihren Text das Abschalten anbieten, wenn sie aus ist, zum Anschalten.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Wir gehen davon aus, dass die Maschine zu Beginn ausgeschaltet ist, also initialisieren wir die Checkbox mit dem String, der zum *An*schalten auffordert. Der Konstruktor von Checkbox, der einen einzelnen Parameter von Klasse String hat, setzt den Zustand der Checkbox auf aus, das passt also.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Wie angekündigt, kommt bei Klasse Checkbox ein weiteres Listener-Interface ins Spiel, genannt ItemListener. Wir werden gleich eine ItemListener-Klasse namens TurnOffListener davon ableiten und halten hier erst einmal fest, dass sie natürlich einen Verweis auf die Checkbox und einen auf die Maschine braucht, um einerseits den Text auf der Checkbox umzustellen und andererseits die Maschine an- beziehungsweise abzuschalten. Da der Listener auch den Text an der Checkbox jeweils passend ändern soll, braucht er natürlich auch die beiden Strings.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Klasse Checkbox hat dann eben auch eine Methode addItemListener, um ItemListener zu registrieren.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
SomeKindOfMachine machine = new SomeKindOfMachine ( ..... );
String askToTurnOn = new String ( “Tick to turn on!“ );
String askToTurnOff = new String ( “Tick to turn off“ );
Checkbox checkbox = new Checkbox ( askToTurnOn );
ItemListener listener
      = new TurnOnOffListener
          ( checkbox, machine, askToTurnOn, askToTurnOff );
checkbox.addItemListener ( listener );
frame.add ( checkbox );
```

Und auch hier wird die neue Komponente wieder auf dieselbe Weise eingefügt, wie wir es bei Button und Canvas schon gesehen hatten.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class TurnOnOffListener implements ItemListener {
  private Checkbox checkbox;
  private SomeKindOfMachine machine;
  String askToTurnOn;
  String askToTurnOff;
  public TurnOnOffListener
            ( Checkbox checkbox, SomeKindOfMachine machine,
              String askToTurnOn, String askToTurnOff ) {
    this.checkbox = checkbox;
    this.machine = machine;
    this.askToTurnOn = askToTurnOn;
    this.askToTurnOff = askToTurnOff;
  }
  ..........
}
```

Jetzt wie angekündigt die eigene ItemListener-Klasse. Die beiden private-Attribute, denen im Konstruktor die Parameterwerte zugewiesen werden, bieten eigentlich nichts Neues, das alles sollte selbsterklärend sein.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class TurnOnOffListener implements ItemListener {
  private Checkbox checkbox;
  private SomeKindOfMachine machine;
  String askToTurnOn;
  String askToTurnOff;
  public TurnOnOffListener
            ( Checkbox checkbox, SomeKindOfMachine machine,
              String askToTurnOn, String askToTurnOff ) {
     this.checkbox = checkbox;
     this.machine = machine;
     this.askToTurnOn = askToTurnOn;
     this.askToTurnOff = askToTurnOff;
   }
   .........
}
```

Der Name ist hoffentlich einigermaßen aussagekräftig.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void itemStateChanged ( ItemEvent event ) {
   if ( checkbox.isSelected() ) {
      machine.turnOn();
      checkbox.setLabel ( askToTurnOff );
   } else {
      machine.turnOff();
      checkbox.setLabel ( askToTurnOn );
   }
}
```

Interface ItemListener ist ein funktionales Interface, ...

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void itemStateChanged ( ItemEvent event ) {
    if ( checkbox.isSelected() ) {
        machine.turnOn();
        checkbox.setLabel ( askToTurnOff );
    } else {
        machine.turnOff();
        checkbox.setLabel ( askToTurnOn );
    }
}
```

... und das ist die funktionale Methode. Wir werden in diesem Beispiel nicht auf das als Parameter übergebene Event zugreifen, bei etwas komplexeren Aufgaben würden wir hingegen Informationen über das Event abrufen müssen, um die jeweilige Aufgabe zu lösen.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void itemStateChanged ( ItemEvent event ) {
   if ( checkbox.isSelected() ) {
      machine.turnOn();
      checkbox.setLabel ( askToTurnOff );
   } else {
      machine.turnOff();
      checkbox.setLabel ( askToTurnOn );
   }
}
```

Die Methode isSelected von Klasse Checkbox ist boolesch und liefert genau dann true zurück, wenn die Checkbox an ist.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void itemStateChanged ( ItemEvent event ) {
   if ( checkbox.isSelected() ) {
      machine.turnOn();
      checkbox.setLabel ( askToTurnOff );
   } else {
      machine.turnOff();
      checkbox.setLabel ( askToTurnOn );
   }
}
```

In diesem Fall wurde also die Checkbox durch das Event, das diesen Aufruf von itemStateChanged getriggert hat, angeschaltet. Das ist genau der Fall, in dem die Maschine anzuschalten ist, und der Text der Checkbox muss dann auch entsprechend ausgetauscht werden.

Checkbox

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void itemStateChanged ( ItemEvent event ) {
   if ( checkbox.isSelected() ) {
      machine.turnOn();
      checkbox.setLabel ( askToTurnOff );
   } else {
      machine.turnOff();
      checkbox.setLabel ( askToTurnOn );
   }
}
```

Dasselbe spiegelsymmetrisch dann im Fall, dass das letzte Event die Checkbox *aus*geschaltet hat. Damit ist das Beispiel für Checkbox auch schon vollendet.

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Choice choice = new Choice();
choice.add ( “Yes“ );
choice.add ( “No“ );
choice.add ( “Undecided“ );
choice.select ( 2 );
choice.addItemListener ( new BooleanMenuListener ( choice ) );
frame.add ( choice );
```

Das nächste Beispiel in alphabetischer Reihenfolge ist die Klasse Choice. Ein Choice-Objekt repräsentiert ein Auswahlmenü.

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Choice choice = new Choice();
choice.add ( “Yes“ );
choice.add ( “No“ );
choice.add ( “Undecided“ );
choice.select ( 2 );
choice.addItemListener ( new BooleanMenuListener ( choice ) );
frame.add ( choice );
```

Eine Choice ist bei Einrichtung einfach leer, daher leere Parameterliste.

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Choice choice = new Choice();
choice.add ( “Yes“ );
choice.add ( “No“ );
choice.add ( “Undecided“ );
choice.select ( 2 );
choice.addItemListener ( new BooleanMenuListener ( choice ) );
frame.add ( choice );
```

Die einzelnen Menüpunkte werden mit Methode add eingefügt und kommen in der Reihenfolge ihrer Einfügung auf die Positionen 0, 1, 2 und so weiter.

***Nebenbemerkung*: Es gibt auch eine Methode add mit der Position als zweitem Parameter, die den neuen Menüpunkt an eben dieser Position einfügt. Wie üblich beim Einfügen an bestimmten Positionen, steigt die Position daraufhin um 1 bei dem Menüpunkt, der vorher an dieser Position war, sowie bei allen Menüpunkten mit höheren Positionen.**

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Choice choice = new Choice();
choice.add ( “Yes“ );
choice.add ( “No“ );
choice.add ( “Undecided“ );
choice.select ( 2 );
choice.addItemListener ( new BooleanMenuListener ( choice ) );
frame.add ( choice );
```

Nicht nur der Nutzer kann den Menüpunkt auswählen, sondern das geht auch auf Programmebene mit Methode select. Typischerweise wird diese Möglichkeit so wie hier genutzt, nämlich zum Initialisieren.

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Choice choice = new Choice();
choice.add ( “Yes“ );
choice.add ( “No“ );
choice.add ( “Undecided“ );
choice.select ( 2 );
choice.addItemListener ( new BooleanMenuListener ( choice ) );
frame.add ( choice );
```

Eine für diese Choice zweckmäßige, eigene ItemListener-Klasse definieren wir gleich.

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Choice choice = new Choice();
choice.add ( “Yes“ );
choice.add ( “No“ );
choice.add ( “Undecided“ );
choice.select ( 2 );
choice.addItemListener ( new BooleanMenuListener ( choice ) );
frame.add ( choice );
```

Und auch hier geht das Hinzufügen so wie in allen Fällen bisher.

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BooleanMenuListener implements ItemListener {
   private Choice choice;
   public BooleanMenuListener ( Choice choice ) {
      this.choice = choice;
   }
   public void itemStateChanged ( ItemEvent event ) {
      System.out.print ( choice.getSelectedItem() );
   }
}
```

Jetzt wie angekündigt zu der ItemListener-Klasse. Während wir bei Canvas und Checkbox einigermaßen realistische Beispiele betrachtet hatten, bleibt dieses Beispiel jetzt kurz und rein illustrativ. Gegenüber der ItemListener-Klasse im Checkbox-Beispiel soeben gibt es hier eigentlich nichts Neues, ...

Choice

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class BooleanMenuListener implements ItemListener {
   private Choice choice;
   public BooleanMenuListener ( Choice choice ) {
      this.choice = choice;
   }
   public void itemStateChanged ( ItemEvent event ) {
      System.out.print ( choice.getSelectedItem() );
   }
}
```

... abgesehen von dieser Methode getSelectedItem, die den Text zu dem Menüpunkt, dessen Auswahl das Event getriggert hat, zurückliefert.

***Nebenbemerkung*: In einer echten Anwendung von Klasse Choice würden auf Basis der Information, welcher Menüpunkt ausgewählt wurde, natürlich weitere Aktionen folgen. Dann würde man wahrscheinlich auch eher getSelected*Index* statt getSelected*Item* wählen, was die Position des ausgewählten Menüpunktes zurückliefert.**

Damit soll es für Klasse Choice aber hier sein Bewenden haben.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....;
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Nun ein sicherlich nicht untypisches Beispiel für die Verwendung von Klasse Label. Ein Label ist einfach ein Rechteck mit einem Text darin. Mit einem Label-Objekt kann nur auf Programmebene interagiert werden, der Nutzer des GUIs kann es sich nur ansehen.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....;
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Mit diesem Konstruktor wird der Text, der auf dem Label dargestellt wird, initial gesetzt.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....;
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Mit Methode setAlignment kann man auswählen, ob der Text horizontal zentriert so wie hier oder statt dessen linksbündig oder rechtsbündig im Rechteck platziert werden soll. Für links- und rechtsbündige Platzierung ersetze man CENTER durch LEFT beziehungsweise RIGHT.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....;
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Der Hintergrund soll hier eine Bedeutung bekommen: grün dafür, dass alles ok ist, rot für Alarm.

***Nebenbemerkung*: Da ein nicht unerheblicher Teil der Bevölkerung rot-grün-blind ist, sollte man sich nicht allein auf die Farbe als Medium zur Vermittlung der Botschaft verlassen. Das machen wir hier auch nicht, sondern wie wir gleich sehen werden, wird immer auch der Text passend ausgewechselt.**

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....;
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Zum Einfügen des Labels im Fenster nur zwei Worte: wie gehabt.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....;
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Unser Beispiel beinhaltet wieder wie eben bei Checkbox eine imaginäre Maschine, die wieder über die ebenfalls imaginäre Klasse SomeKindOfMachine mit dem Programm verknüpft ist. Diesmal wollen wir sie aber nicht abschalten, sondern wir wollen dem Nutzer im Label mitteilen, ob alles ok mit der Maschine ist oder nicht.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Label label = new label ( “Everything is fine“ );
label.setAligment ( Label.CENTER );
label.setBackground ( Color.GREEN );
frame.add ( label );
SomeKindOfMachine machine = .....
SomeKindOfMachineListener listener
        = new MyMachineListener ( label );
machine.addSomeKindOfMachineListener ( listener );
```

Jetzt müssen wir erstmals auf Events nicht bei einem graphischen Objekt warten, sondern bei einer anderen Entität, konkret bei unserer imaginären Maschine. Zu diesem Zweck werden wir jetzt zum ersten Mal ein eigenes Listener-Interface definieren und nicht nur ein vorgegebenes verwenden.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public interface SomeKindOfMachineListener {
   void machineHasGoneDown();
   void machineIsUpAgain();
}
```

Dieses Interface ist recht einfach. Es hat zwei Methoden, die sogar parameterlos sind. Die für das Label allein wichtige Information, die von der Maschine an den Listener zu kommunizieren ist, ist binär und sozusagen in der Auswahl der Methode kodiert.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyMachineListener
                    implements SomeKindOfMachineListener {
   private Label label;
   public MyMachineListener ( Label label ) {
      this.label = label;
   }
   ..........
}
```

Natürlich müssen wir wie bisher auch hier noch eine Klasse vom Listener-Interface ableiten. Die auf dieser Folie zu sehenden Bestandteile – ein private-Attribut und ein Konstruktor, der den aktualen Parameterwert dem private-Attribut zuweist – beinhaltet nichts Neues und sollte selbsterklärend sein.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void machineHasGoneDown () {
   label.setBackground ( Color.RED );
   label.setText ( “Red alert!“ );
}

public void machineIsUpAgain () {
   label.setBackgound ( Color.GREEN );
   label.setText ( “Everything is fine again \u1F60C“ );
}
```

jetzt müssen noch die beiden Methoden des neuen Listener-Interface in der das Interface implementierenden Klasse definiert werden.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void machineHasGoneDown () {
   label.setBackground ( Color.RED );
   label.setText ( “Red alert!“ );
}

public void machineIsUpAgain () {
   label.setBackgound ( Color.GREEN );
   label.setText ( “Everything is fine again \u1F60C“ );
}
```

Wir hatten schon geklärt, dass der Hintergrund grün bei Normalbetrieb und rot im Falle eines Alarms sein soll.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void machineHasGoneDown () {
   label.setBackground ( Color.RED );
   label.setText ( “Red alert!“ );
}

public void machineIsUpAgain () {
   label.setBackgound ( Color.GREEN );
   label.setText ( “Everything is fine again \u1F60C“ );
}
```

Und mit Methode setText kann der momentan in einem Label angezeigte Text durch den Parameter überschrieben werden.

Label

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public void machineHasGoneDown () {
   label.setBackground ( Color.RED );
   label.setText ( “Red alert!“ );
}

public void machineIsUpAgain () {
   label.setBackgound ( Color.GREEN );
   label.setText ( “Everything is fine again \u1F60C“ );
}
```

Erinnerung **von Kapitel 01b, Abschnitt „Allgemein: Primitive Datentypen“: Einen Wert vom primitiven Datentyp char kann man immer durch seine Unicode-Nummer im Hexadezimalsystem angeben, wobei der Backslash und klein-u für Unicode voranzustellen sind. Wenn Sie die hexadezimalen Ziffern als Suchbegriff in Ihre favorisierte Suchmaschine eingeben, sollten Sie etliche Treffer bekommen, die Ihnen das mit dieser Unicode-Nummer verbundene Zeichen zeigen. Damit ist dieses Beispiel für Labels komplett.**

List

TECHNISCHE UNIVERSITÄT DARMSTADT

```
List list = new List ( 3, true );
list.add ( “Carpenter“ );
list.add ( “Kubrick“ );
list.add ( “Lucas“ );
list.add ( “Scott“ );
list.add ( “Spielberg“ );
.....
int[ ] selectedIndexes = list.getSelectedIndexes();
```

Als nächstes Klasse List. Beachten Sie, dass diese Klasse in Package java.awt zu finden ist, das Interface List aus dem Kapitel zu Generics und Collections hingegen in java.util. Abgesehen von der Namensgleichheit haben das Interface in java.util und die Klasse in java.awt nichts miteinander zu tun. Die Namensgleichheit ist aber dennoch kein Zufall, denn die repräsentierten Konzepte haben viel Ähnlichkeit miteinander, und das Gemeinsame lässt sich abstrakt gut in den Begriff „Liste“ fassen.

In diesem Kapitel ist ausschließlich von Klasse List in Package java.awt die Rede.

List

TECHNISCHE UNIVERSITÄT DARMSTADT

```
List list = new List ( 3, true );
list.add ( “Carpenter“ );
list.add ( “Kubrick“ );
list.add ( “Lucas“ );
list.add ( “Scott“ );
list.add ( “Spielberg“ );
.....
int[ ] selectedIndexes = list.getSelectedIndexes();
```

Ein Objekt der Klasse List ist ein Menü, das ein paar Unterschiede zu einem Objekt der eben betrachteten Klasse Choice aufweist. Die zwei für uns wesentlichen Unterschiede schlagen sich in den Parametern dieses Konstruktors nieder.

List

TECHNISCHE UNIVERSITÄT DARMSTADT

```
List list = new List ( 3, true );
list.add ( “Carpenter“ );
list.add ( “Kubrick“ );
list.add ( “Lucas“ );
list.add ( “Scott“ );
list.add ( “Spielberg“ );
.....
int[ ] selectedIndexes = list.getSelectedIndexes();
```

Der erste Parameter gibt an, wie viele Menüpunkte zugleich angezeigt werden. Das sind immer Menüpunkte mit unmittelbar aufeinanderfolgenden Positionen. Im konkreten Beispiel werden also entweder Carpenter, Kubrick und Lucas, Kubrick, Lucas und Scott oder Lucas, Scott und Spielberg angezeigt. Wenn die Anzahl der Menüpunkte so wie hier größer als die Anzahl der gleichzeitig anzuzeigenden Menüpunkte ist, bietet Klasse List eine Scrollbar, um den angezeigten Ausschnitt aus der Menüleiste zu ändern.

List

TECHNISCHE UNIVERSITÄT DARMSTADT

```
List list = new List ( 3, true );
list.add ( “Carpenter“ );
list.add ( “Kubrick“ );
list.add ( “Lucas“ );
list.add ( “Scott“ );
list.add ( “Spielberg“ );
.....
int[ ] selectedIndexes = list.getSelectedIndexes();
```

Der zweite Unterschied ist, dass man bei der Einrichtung eines Objektes von Klasse List – oder auch später, mit Methode setMultipleMode – angeben kann, ob immer nur ein einziger Menüpunkt ausgewählt werden kann oder mehrere zugleich. Konkret wird mit true angegeben, dass *mehrere* Menüpunkte ausgewählt werden können.

List

TECHNISCHE UNIVERSITÄT DARMSTADT

```
List list = new List ( 3, true );
list.add ( “Carpenter“ );
list.add ( “Kubrick“ );
list.add ( “Lucas“ );
list.add ( “Scott“ );
list.add ( “Spielberg“ );
.....
int[ ] selectedIndexes = list.getSelectedIndexes();
```

Dazu korrespondierend gibt es diese Methode, mit der jederzeit abgefragt werden kann, welche Menüpunkte gerade ausgewählt sind. Da das mehrere sein können, ist die Rückgabe ein Array. Die int-Werte sind die Positionen der ausgewählten Menüpunkte, und das Array ist gerade so groß wie die Anzahl der ausgewählten Menüpunkte.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                                ( Scrollbar.VERTICAL, initialValue,
                                  visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Jetzt zu Scrollbars als eigenständigen graphischen Komponenten. Scrollbars werden häufig automatisch hinzugefügt, so wie soeben bei Klasse List. Wir können aber auch selbst Scrollbars einfügen, die dann als Schieberegler fungieren.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                              ( Scrollbar.VERTICAL, initialValue,
                                visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Mit Angabe VERTICAL beziehungsweise HORIZONTAL wird angegeben, ob die Scrollbar von oben nach unten oder von links nach rechts geht.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                              ( Scrollbar.VERTICAL, initialValue,
                                visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Die letzten beiden Parameter dieses Konstruktors geben den Wertebereich an, über den gescrollt werden kann. Dieser Wertebereich beeinflusst *nicht*, wie groß die Scrollbar auf dem Bildschirm sein wird. Die Größe der Scrollbar auf dem Bildschirm wird ausschließlich auf Basis der miteinander konkurrierenden Platzansprüche der einzelnen Komponenten des Fensters erstellt, das sehen wir uns gleich im Abschnitt zum Layout-Manager genauer an.

Wir können den Wertebereich also problemlos so definieren, dass er nahtlos zur Anwendungslogik passt.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                                ( Scrollbar.VERTICAL, initialValue,
                                  visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Die Anwendungslogik ist in diesem kleinen Beispiel, dass der Nutzer den Rotanteil einer Farbe in dem uns dafür schon bekannten Bereich von 0 bis 255 über die Scrollbar festlegen können soll.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                              ( Scrollbar.VERTICAL, initialValue,
                                visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Am Anfang, bei Einrichtung der Scrollbar, soll der von der Scrollbar gezeigte Ausschnitt genau mittig sind.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                              ( Scrollbar.VERTICAL, initialValue,
                                visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Mit dem dritten Parameter wird angegeben, wie groß der Balken zum Scrollen sein soll.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                                ( Scrollbar.VERTICAL, initialValue,
                                  visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Auch hier wieder sieht das Einfügen völlig identisch aus.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                                ( Scrollbar.VERTICAL, initialValue,
                                  visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Hier sehen wir erstmals, wozu das Ganze gut sein soll: Die Hintergrundfarbe des Fensters – genauer: deren Rotanteil – soll über die Scrollbar gesteuert werden. Grün- und Blauanteile sollen 0 sein, die Hintergrundfarbe des Fensters variiert also über die verschieden dunklen reinen Rottöne.

Damit die Logik von Fensterfarbe und Scrollbar von Anfang an zusammenpassen, wird der Rotanteil im Fenster zu Beginn auf den Wert gesetzt, der zum initialen Wert der Scrollbar passt.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
final int maxColorValue = 255;
final int initialValue = 128;
final int visibleRange = 16;
Scrollbar scrollbar = new Scrollbar
                                ( Scrollbar.VERTICAL, initialValue,
                                  visibleRange, 0, maxColorValue );
frame.add ( scrollBar );
frame.setBackground ( new Color ( initialValue, 0, 0 ) );
scrollbar.addAdjustmentListener
                    ( new MyAdjustmentListener ( frame ) );
```

Die Events, die eine Scrollbar liefert, sind natürlich anders strukturiert als bei anderen graphischen Objektarten. Deshalb gibt es ein eigenes Listener-Interface, das wir dann auf der nächsten Folie implementieren.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyAdjustmentListener implements AdjustmentListener {
   private Frame frame;
   public MyAdjustmentListener ( Frame frame ) {
      this.frame = frame;
   }
   public void adjustmentValueChanged ( AdjustmentEvent event ) {
      frame.setBackground ( event.getValue(), 0, 0 );
   }
}
```

Unsere Implementation von AdjustmentListener passt sogar auf eine einzelne Folie, was sicherlich damit zu tun hat, dass AdjustmentListener ein funktionales Interface ist.

Das private-Attribut und der Konstruktor bieten nichts Neues.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyAdjustmentListener implements AdjustmentListener {
   private Frame frame;
   public MyAdjustmentListener ( Frame frame ) {
      this.frame = frame;
   }
   public void adjustmentValueChanged ( AdjustmentEvent event ) {
      frame.setBackground ( event.getValue(), 0, 0 );
   }
}
```

Diese Methode von Interface AdjustmentListener ist zu implementieren.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyAdjustmentListener implements AdjustmentListener {
   private Frame frame;
   public MyAdjustmentListener ( Frame frame ) {
      this.frame = frame;
   }
   public void adjustmentValueChanged ( AdjustmentEvent event ) {
      frame.setBackground ( event.getValue(), 0, 0 );
   }
}
```

Wie gesagt, der Effekt einer Interaktion des Nutzers mit der Scrollbar soll sein, dass der Rotanteil der Hintergrundfarbe des Fensters sich ändert.

Scrollbar

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyAdjustmentListener implements AdjustmentListener {
   private Frame frame;
   public MyAdjustmentListener ( Frame frame ) {
      this.frame = frame;
   }
   public void adjustmentValueChanged ( AdjustmentEvent event ) {
      frame.setBackground ( event.getValue(), 0, 0 );
   }
}
```

Methode getValue von Klasse AdjustmentEvent liefert den neuen Wert im Bereich 0 bis 255, an dem der Balken in der Scrollbar steht, nachdem der Nutzer mit dem Scrollen fertig ist. Da wir den Wertebereich für Farben eins-zu-eins als Wertebereich für die Scrollbar übernommen haben, bedarf es hier keiner Umrechnung von einem Wertebereich in einen anderen.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
TextField passwordField = new TextField ( 256 );
passwordField.setEchoChar ( ´*´ );
MyPasswordHandler handler = .....;
passwordField.addKeyListener
        ( new MyPasswordListener ( handler, passwordField ) );
frame.add ( passwordField );
```

Als nächstes und letztes wäre jetzt eigentlich Klasse TextComponent dran. Aber TextComponent ist eigentlich nur eine gemeinsame Abstraktion von zwei Klassen, die zwei leicht verschiedene, immer wieder auftretende Fälle abdecken. Diese beiden Klassen heißen TextField und TextArea. Sie sind direkt von TextComponent abgeleitet. Diese beiden Klassen betrachten wir hier, zuerst die einfachere, TextField.

Ein Objekt der Klasse TextField repräsentiert eine Zeile, in die der Nutzer etwas hineinschreiben kann. Typische Anwendungsfälle sind Einloggen mit Textfeldern für Benutzername und Passwort oder etwa einzelne Felder in Online-Formularen. Wir betrachten hier einen Anwendungsfall, der Ihnen sicherlich wohl vertraut ist: das Eingeben eines Passworts. Genauer: Sie haben Ihren Benutzernamen schon in einer vorherigen Maske eingegeben und geben jetzt Ihr Passwort mit einer zweiten Maske ein, und Ihre Eingabe wird durch Betätigen der Enter-Taste beendet.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
TextField passwordField = new TextField ( 256 );
passwordField.setEchoChar ( ´*´ );
MyPasswordHandler handler = .....;
passwordField.addKeyListener
        ( new MyPasswordListener ( handler, passwordField ) );
frame.add ( passwordField );
```

Mit diesem Konstruktor von Klasse TextField lässt sich die Zeichenzahl in der Zeile spezifizieren.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
TextField passwordField = new TextField ( 256 );
passwordField.setEchoChar ( ´*´ );
MyPasswordHandler handler = .....;
passwordField.addKeyListener
        ( new MyPasswordListener ( handler, passwordField ) );
frame.add ( passwordField );
```

Das Passwort-Beispiel bietet die Gelegenheit, ein spezielles Feature einzuführen: Die Zeichen, die man eingibt, werden nicht angezeigt, sondern durch ein einheitliches Zeichen ersetzt. Sehr häufig wird der Stern dafür genommen, das machen wir auch.

Vor dem Aufruf von setEchoChar wäre jede Nutzereingabe eins-zu-eins auf dem Bildschirm angezeigt worden. Will man das später wieder so haben, ruft man setEchoChar mit dem Wert 0 auf, zum Beispiel in der Form (char)0 oder ´\u0000´.

***Erinnerung*: Zeichen sind als Zahlenwerte kodiert, und die 0 ist für kein Zeichen reserviert, sondern dient ähnlich wie null bei Referenztypen zur Anzeige, dass es eigentlich kein Zeichen sein soll.**

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
TextField passwordField = new TextField ( 256 );
passwordField.setEchoChar ( ´*´ );
MyPasswordHandler handler = .....;
passwordField.addKeyListener
        ( new MyPasswordListener ( handler, passwordField ) );
frame.add ( passwordField );
```

Der Einfachheit halber gehen wir davon aus, dass wir uns schon eine Klasse gebaut haben, die mit dem Passwort aus unserem TextField umzugehen weiß.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
TextField passwordField = new TextField ( 256 );
passwordField.setEchoChar ( ´*´ );
MyPasswordHandler handler = .....;
passwordField.addKeyListener
        ( new MyPasswordListener ( handler, passwordField ) );
frame.add ( passwordField );
```

Da es hier um Tastatureingaben geht, ist es vielleicht nicht überraschend, dass hier der KeyListener zum Zuge kommt. Den müssen wir geeignet implementieren. Unsere Implementation benötigt einen Verweis auf den Passwort-Handler sowie auf das Passwort-Feld.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyPasswordListener extends KeyAdapter {
   private TextField passwordField;
   private PasswordHandler handler;
   public MyPasswordListener
                ( MyPasswordHandler handler, TextField passwordField ) {
      this.passwordField = passwordField;
      this.handler = handler;
   }
   .........
}
```

Hier jetzt die Implementation unserer speziellen KeyListener-Klasse. Die private-Attribute und ihre Initialisierung durch die Attribute des Konstruktors, das ist alles wie mehrfach gesehen.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyPasswordListener extends KeyAdapter {
    private TextField passwordField;
    private PasswordHandler handler;
    public MyPasswordListener
                ( MyPasswordHandler handler, TextField passwordField ) {
        this.passwordField = passwordField;
        this.handler = handler;
    }
    .........
}
```

Wir implementieren wieder nicht das Interface KeyListener direkt, sondern leiten von KeyAdapter ab, da uns nur der Fall interessiert, dass die Taste auf der Tastaur ganz normal kurz angeklickt wird.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyPasswordListener extends KeyAdapter {
   .........
   public void keyTyped ( KeyEvent event ) {
      if ( event.getKeyChar() == KeyEvent.VK_ENTER )
         handler.handle ( passwordField.getText() );
   }
}
```

Nun die Methode des Listeners für den Fall eines normalen kurzen Drucks auf die Tastatur. Aus der Behandlung von KeyListener weiter vorne in diesem Kapitel wissen wir schon, dass diese Methode keyTyped heißt und einen Parameter vom formalen Typ KeyEvent hat.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyPasswordListener extends KeyAdapter {
   .........
   public void keyTyped ( KeyEvent event ) {
      if ( event.getKeyChar() == KeyEvent.VK_ENTER )
         handler.handle ( passwordField.getText() );
   }
}
```

Dort hatten wir auch schon gesehen, dass KeyEvent eine Methode getKeyChar hat, deren Rückgabe die vom Nutzer angetippte Taste kodiert.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyPasswordListener extends KeyAdapter {
   .........
   public void keyTyped ( KeyEvent event ) {
      if ( event.getKeyChar() == KeyEvent.VK_ENTER )
         handler.handle ( passwordField.getText() );
   }
}
```

Ebenfalls dort hatten wir gesehen, dass Klasse KeyEvent für jede Taste eine diese Taste identifizierende Klassenkonstante hat, auch für Funktionstasten. Bei der allgemeinen Besprechung von KeyEvent hatten wir die Backspace-Taste als Beispiel für Funktionstasten gesehen, hier brauchen wir jetzt die Enter-Taste.

TextComponent / TextField

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyPasswordListener extends KeyAdapter {
   .........
   public void keyTyped ( KeyEvent event ) {
      if ( event.getKeyChar() == KeyEvent.VK_ENTER )
         handler.handle ( passwordField.getText() );
   }
}
```

Wir hatten eingangs gesagt, dass wir den Fall betrachten, in dem eine Eingabe im TextField durch Enter abgeschlossen ist. Sobald das Enter-Zeichen auftritt, ist also die Eingabe des Passworts als beendet anzusehen, und der Text im TextField kann mit Methode getText ausgelesen und weiterverarbeitet werden.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
String textForEmptyArea = “<type here>“;

final int scrollbars = TextArea.SCROLLBARS_BOTH;

TextArea editArea
    = new TextArea ( textForEmptyArea, 5, 40, scrollbars );

editArea.addFocusListener
            ( new EmptyAreaListener ( editArea, textForEmptyArea ) );

frame.add ( editArea );
```

Zum Abschluss dieses Abschnitts kommen wir nun zu Klasse TextArea, also der zweiten Klasse in java.awt, die von TextComponent direkt abgeleitet ist. Ein Objekt dieser Klasse repräsentiert nicht eine einzelne edierbare Zeile wie TextField, sondern einen Eingabebereich, der prinzipiell aus beliebig vielen Zeilen derselben Länge bestehen kann. Typischer Anwendungsfall ist ein Freitextfeld in einem Online-Formular.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
String textForEmptyArea = "<type here>";
final int scrollbars = TextArea.SCROLLBARS_BOTH;
TextArea editArea
    = new TextArea ( textForEmptyArea, 5, 40, scrollbars );
editArea.addFocusListener
            ( new EmptyAreaListener ( editArea, textForEmptyArea ) );
frame.add ( editArea );
```

Dieser Text soll immer dann im edierbaren Bereich stehen, wenn erstens dieser Bereich leer ist und zweitens der Mausfokus nicht auf diesem TextArea-Objekt ist. Sie kennen das sicherlich aus eigener Erfahrung mit Online-Formularen. Der erste Parameter dieses Konstruktors setzt diesen Text initial in den edierbaren Bereich.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
String textForEmptyArea = "<type here>";
final int scrollbars = TextArea.SCROLLBARS_BOTH;
TextArea editArea
    = new TextArea ( textForEmptyArea, 5, 40, scrollbars );
editArea.addFocusListener
            ( new EmptyAreaListener ( editArea, textForEmptyArea ) );
frame.add ( editArea );
```

Aber wenn später der edierbare Bereich wieder leer wird, dann soll der initiale Text wieder angezeigt werden, solange der Mausfokus woanders ist. Dafür muss dann unsere Listener-Klasse sorgen, die wir gleich definieren.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
String textForEmptyArea = "<type here>";
final int scrollbars = TextArea.SCROLLBARS_BOTH;
TextArea editArea
    = new TextArea ( textForEmptyArea, 5, 40, scrollbars );
editArea.addFocusListener
            ( new EmptyAreaListener ( editArea, textForEmptyArea ) );
frame.add ( editArea );
```

Klasse TextArea bietet die Möglichkeit anzugeben, ob eine horizontale oder eine vertikale Scrollbar oder sogar beide eingefügt werden sollen. Wir entscheiden uns für beide. Für die anderen drei Fälle ersetze man BOTH durch VERTICAL_ONLY, HORIZONTAL_ONLY oder NONE.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
String textForEmptyArea = "<type here>";

final int scrollbars = TextArea.SCROLLBARS_BOTH;

TextArea editArea
    = new TextArea ( textForEmptyArea, 5, 40, scrollbars );

editArea.addFocusListener
            ( new EmptyAreaListener ( editArea, textForEmptyArea ) );

frame.add ( editArea );
```

Der zweite und der dritte Parameter dieses Konstruktors von Klasse TextArea geben die initiale Anzahl der Zeilen und die Länge der Zeilen an.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class EmptyAreaListener implements FocusListener {
   TextArea editArea;
   String textForEmptyArea;
   public EmptyAreaListener
                        ( TextArea textArea, String textForEmptyArea ) {
      this.editArea = editArea;
      this.textForEmptyArea = textForEmptyArea;
   }
   .........
}
```

Nun noch die FocusListener-Klasse, dann ist diesen Beispiel und damit dieser Abschnitt fertig. Die private-Attribute und der Konstruktor sind auch hier wieder wie gehabt.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class EmptyAreaListener implements FocusListener {
   TextArea editArea;
   String textForEmptyArea;
   public EmptyAreaListener
                        ( TextArea textArea, String textForEmptyArea ) {
      this.editArea = editArea;
      this.textForEmptyArea = textForEmptyArea;
   }
   .........
}
```

Interface FocusListener ist nicht zu verwechseln mit dem Interface WindowFocusListener, das wir schon vorher gesehen hatten. Sinn und Zweck sind bei beiden identisch, nur bei dem einen für ganze Fenster, bei dem anderen für einzelne Komponenten von Fenstern. Die Event-Klassen sind auch unterschiedlich, angepasst an die jeweiligen Erfordernisse. Und die Namen der Methoden sind leicht anders: *mit* Präfix Window beim WindowFocusListener, *ohne* dieses Präfix beim FocusListener.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public focusGained ( FocusEvent event ) {
   if ( textArea.getText().equals ( textForEmptyArea ) )
      textArea.setText ( "" );
}

public focusLost ( FocusEvent event ) {
   if ( textArea.getText().equals ( "" ) )
      textArea.setText ( textForEmptyArea );
}
```

Interface FocusListener hat zwei Methoden, focusGained und focusLost. Wir erinnern uns: Solange der Mausfokus anderswo ist, soll der Inhalt der String-Variable textForEmptyArea in einem eigentlich leeren edierbaren Bereich stehen.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public focusGained ( FocusEvent event ) {
   if ( textArea.getText().equals ( textForEmptyArea ) )
      textArea.setText ( "" );
}

public focusLost ( FocusEvent event ) {
   if ( textArea.getText().equals ( "" ) )
      textArea.setText ( textForEmptyArea );
}
```

In dem Moment, in dem der Mausfokus wieder in die TextArea kommt, ist also der Fall interessant, dass textForEmptyArea der Inhalt des edierbaren Bereiches ist. In diesem Fall soll dieser Platzhaltertext verschwinden, bevor der Nutzer mit dem Edieren beginnt.

TextComponent / TextArea

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public focusGained ( FocusEvent event ) {
   if ( textArea.getText().equals ( textForEmptyArea ) )
      textArea.setText ( "" );
}

public focusLost ( FocusEvent event ) {
   if ( textArea.getText().equals ( "" ) )
      textArea.setText ( textForEmptyArea );
}
```

Umgekehrt, wenn der Mausfokus anderswohin geht: Falls jetzt der edierbare Bereich leer ist, soll textForEmptyArea hier erscheinen. Damit sind dieses Beispiel und dieser Abschnitt beendet.

TECHNISCHE UNIVERSITÄT DARMSTADT

Hierarchie von graphischen Komponenten und LayoutManager

Bisher hatten wir alle Arten von graphischen Komponenten nebeneinander her betrachtet. Das ändern wir jetzt.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Von Component direkt abgeleitet:

- Button
- Canvas
- Checkbox
- Choice
- Label
- List
- Scrollbar
- TextComponent

Ebenfalls von Component direkt abgeleitet:

- Container

Von Container direkt abgeleitet:

- Window

Von Window direkt abgeleitet:

- Frame

Zunächst einmal sehen wir uns die Vererbungshierarchie auf den Klassen aus Package java.awt an, die wir bisher so gesehen haben.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Von Component direkt abgeleitet:

- Button
- Canvas
- Checkbox
- Choice
- Label
- List
- Scrollbar
- TextComponent

Ebenfalls von Component direkt abgeleitet:

- Container

Von Container direkt abgeleitet:

- Window

Von Window direkt abgeleitet:

- Frame

Alle Klassen für graphische Komponentenarten, die wir besprochen hatten, sind direkt von einer Klasse namens Component abgeleitet.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Von Component direkt abgeleitet:

- Button
- Canvas
- Checkbox
- Choice
- Label
- List
- Scrollbar
- TextComponent

Ebenfalls von Component direkt abgeleitet:

- Container

Von Container direkt abgeleitet:

- Window

Von Window direkt abgeleitet:

- Frame

Zu einer weiteren Klasse namens Container, die direkt von Component abgeleitet ist, kommen wir gleich.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Von Component direkt abgeleitet:

- Button
- Canvas
- Checkbox
- Choice
- Label
- List
- Scrollbar
- TextComponent

Ebenfalls von Component direkt abgeleitet:

- Container

Von Container direkt abgeleitet:

- Window

Von Window direkt abgeleitet:

- Frame

Von dieser ominösen Klasse Container sind weitere Klassen direkt abgeleitet, unter anderem eine Klasse namens Window, die wir zu Beginn des Kapitels nur kurz erwähnt haben, ohne näher darauf einzugehen. Diese Klasse repräsentiert ein Fenster *ohne* Rahmen.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Von Component direkt abgeleitet:

- Button
- Canvas
- Checkbox
- Choice
- Label
- List
- Scrollbar
- TextComponent

Ebenfalls von Component direkt abgeleitet:

- Container

Von Container direkt abgeleitet:

- Window

Von Window direkt abgeleitet:

- Frame

Der Unterschied zwischen Window und Frame ist dann, dass ein Fenster der Klasse Frame – wie der Name schon sagt – einen Rahmen hat.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Component (alle in früheren Abschnitten aufgetretenen):

- **public void addFocusListener (FocusListener listener)**
- **public void addKeyListener (KeyListener listener)**
- **public void addMouseListener (MouseListener listener)**
- **public void addMouseMotionListener (MouseMotionListener listener)**
- **public void addMouseWheelListener (MouseWheelListener listener)**
- **public FontMetrics getFontMetrics (Font font)**
- **public void paint (Graphics graphics)**
- **public void setBackground (Color color)**
- **public void setFont (Font font)**
- **public void setVisible (boolean visible)**

Etliche der Methoden, die wir bisher auf diverse Arten von graphischen Komponenten angewendet hatten, sind in Wirklichkeit schon in der Basisklasse Component definiert und in den Klassen für die einzelnen Komponentenarten beziehungsweise für Fenster jeweils entweder vererbt oder überschrieben.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Component (alle in früheren Abschnitten aufgetretenen):

```
public void addFocusListener ( FocusListener listener )
public void addKeyListener ( KeyListener listener )
public void addMouseListener ( MouseListener listener )
public void addMouseMotionListener ( MouseMotionListener listener )
public void addMouseWheelListener ( MouseWheelListener listener )
public FontMetrics getFontMetrics ( Font font )
public void paint ( Graphics graphics )
public void setBackground ( Color color )
public void setFont ( Font font )
public void setVisible ( boolean visible )
```

Diese Methoden sind offensichtlich für alle Arten von graphischen Komponenten sinnvoll und daher zu Recht schon in der Basisklasse Component definiert.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Component (alle in früheren Abschnitten aufgetretenen):

- public void addFocusListener (FocusListener listener)
- public void addKeyListener (KeyListener listener)
- public void addMouseListener (MouseListener listener)
- public void addMouseMotionListener (MouseMotionListener listener)
- public void addMouseWheelListener (MouseWheelListener listener)
- public FontMetrics getFontMetrics (Font font)
- public void paint (Graphics graphics)
- public void setBackground (Color color)
- public void setFont (Font font)
- public void setVisible (boolean visible)

Die Methode paint ist letztendlich bei allen Arten von GUI-Komponenten dafür verantwortlich, wie eine graphische Komponente dargestellt wird. Deshalb ist sie ebenfalls schon in Component definiert und nicht erst in Canvas, wo wir sie kennen gelernt hatten. Es gehört zur Logik von Canvas, dass man paint in einer von Canvas abgeleiteten Klasse so überschreibt, dass ein selbstentworfenes Bild herauskommt; bei den anderen Komponentenarten wird man paint fast immer unverändert lassen.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Component (alle in früheren Abschnitten aufgetretenen):

- public void addFocusListener (FocusListener listener)
- public void addKeyListener (KeyListener listener)
- public void addMouseListener (MouseListener listener)
- public void addMouseMotionListener (MouseMotionListener listener)
- public void addMouseWheelListener (MouseWheelListener listener)
- public FontMetrics getFontMetrics (Font font)
- public void paint (Graphics graphics)
- public void setBackground (Color color)
- public void setFont (Font font)
- public void setVisible (boolean visible)

Nicht nur ganze Fenster, sondern jede einzelne Komponente für sich kann sichtbar oder unsichtbar gemacht werden. Der Hauptzweck ist derselbe wie bei Fenstern: Es kann einige Zeit dauern, bis das graphische Bild einer Komponente vollständig aufgebaut ist, und in dieser Zeit sollte die Komponente nicht sichtbar sein.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Component comp1 = .....;
Component comp2 = .....;
Component comp3 = .....;
Container container1 = new Container ();
container1.add ( comp1 );
container1.add ( comp2 );
Container container2 = new Container ();
container2.add ( comp3 );
container2.add ( container1 );
```

Kommen wir jetzt zu Klasse Container. Wie der Name schon sagt, hat diese direkt von Component abgeleitete Klasse die Funktion, mehrere Komponenten zu einer zusammenzufassen. Man könnte auch sagen, ein Container ist eine Liste von graphischen Komponenten. Daher ist die Methode add, die wir regelmäßig auf Frame-Objekte angewandt hatten, in Klasse Container erstmals definiert.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Component comp1 = .....;
Component comp2 = .....;
Component comp3 = .....;
Container container1 = new Container ();
container1.add ( comp1 );
container1.add ( comp2 );
Container container2 = new Container ();
container2.add ( comp3 );
container2.add ( container1 );
```

Diese acht Anweisungen spiegeln unser bisheriges Verständnis gut wider. Sie sind völlig analog zu unseren bisherigen Anwendungen der Methode add mit der indirekt von Container abgeleiteten Klasse Frame.

Hierarchie

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Component comp1 = .....;
Component comp2 = .....;
Component comp3 = .....;
Container container1 = new Container ();
container1.add ( comp1 );
container1.add ( comp2 );
Container container2 = new Container ();
container2.add ( comp3 );
container2.add ( container1 );
```

In dieser letzten Zeile zeigt sich die ganze Stärke des Konzepts: Dadurch, dass Container von Component abgeleitet ist, können auch Container zu anderen Containern mit Methode add hinzugefügt werden. Wir bekommen eine Hierarchie von Komponenten: Ein Container kann gewöhnliche Komponenten und Container enthalten, diese Container wiederum gewöhnliche Komponenten und Container, und so weiter. Die Container können dann natürlich auch Fenster mit und ohne Rahmen sein, das heißt, auch Fenster können Teil von anderen Fenstern sein.

***Nebenbemerkung*: Das kann man sich so ähnlich vorstellen wie bei Dateistrukturen: Ein Verzeichnis kann Dateien und Verzeichnisse enthalten, diese Verzeichnisse können wiederum Dateien und Verzeichnisse enthalten, und so weiter.**

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

- **public void paint (Graphics graphics)**
- **public void add (Component comp)**
- **public void add (Component comp, Object constraints)**
- **public void setLayout (LayoutManager manager)**
- **public void validate ()**

Wir beschäftigen uns kurz mit ein paar ausgewählten Methoden von Klasse Container und leiten dabei zum zweiten Thema dieses Abschnitt über, Interface LayoutManager.

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

- **public void paint (Graphics graphics)**
- **public void add (Component comp)**
- **public void add (Component comp, Object constraints)**
- **public void setLayout (LayoutManager manager)**
- **public void validate ()**

Methode paint ist zwar schon in der direkten Basisklasse Component von Klasse Container definiert. Aber es ist sicherlich einleuchtend, dass paint für einen Container etwas anderes machen sollte als für einzelne graphische Komponenten. Daher ist Methode paint für Container überschrieben und ruft im Wesentlichen Methode paint für jede einzelne momentan im Container gespeicherte graphische Komponente auf, natürlich auch für alle momentan gespeicherten Container, die dann ihrerseits wieder Methode paint für alle in ihnen gespeicherten Komponenten aufrufen, und so weiter.

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

- **public void paint (Graphics graphics)**
- **public void add (Component comp)**
- **public void add (Component comp, Object constraints)**
- **public void setLayout (LayoutManager manager)**
- **public void validate ()**

***Diese* Methode add haben wir jetzt oft genug gesehen.**

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

- **public void paint (Graphics graphics)**
- **public void add (Component comp)**
- **public void add (Component comp, Object constraints)**
- **public void setLayout (LayoutManager manager)**
- **public void validate ()**

Aber Klasse Container hat noch eine zweite Methode desselben Namens add, die uns gleich beschäftigen wird. Wir halten hier erst einmal fest, dass *diese* Methode add noch weitere Informationen bekommt, die das Hinzufügen des ersten Parameters offenbar irgendwie steuern.

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

- **public void paint (Graphics graphics)**
- **public void add (Component comp)**
- **public void add (Component comp, Object constraints)**
- **public void setLayout (LayoutManager manager)**
- **public void validate ()**

Jedes Container-Objekt hat zu jedem Zeitpunkt genau einen LayoutManager, der – wie der Name schon sagt, die Steuerung übernimmt, wie die einzelnen Komponenten des Containers platziert werden, also wohin jede einzelne Komponente platziert wird und wie groß sie dargestellt wird.

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

```
public void paint ( Graphics graphics )
public void add ( Component comp )
public void add ( Component comp, Object constraints )
public void setLayout ( LayoutManager manager )
public void validate ()
```

Und dieser zweite Parameter der zweiten Methode add wird an den LayoutManager weitergereicht, die dieser als einen Satz von Vorgaben versteht, die seine Freiheit, die Komponente zu platzieren, einschränkt, daher der Name constraints. Da niemand vorhersehen kann, was für Informationen irgendwelche LayoutManager-Klassen benötigen, die im Laufe der Zeit irgendwo auf der Welt definiert werden, hat man sich für größtmögliche Gestaltungsfreiheit entschieden, indem man Klasse Object zum formalen Typ des Parameters constraints gemacht hat.

Hierarchie / LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden von Container (Auswahl):

- **public void paint (Graphics graphics)**
- **public void add (Component comp)**
- **public void add (Component comp, Object constraints)**
- **public void setLayout (LayoutManager manager)**
- **public void validate ()**

Verschiedenste Methoden von Klasse Container beziehungsweise Component ändern das Layout, wenn sie auf den Container als Ganzes oder auf einzelne Komponenten im Container angewendet werden. Ein Beispiel dafür ist die Änderung der Schriftart mit der Methode setFont von Klasse Component. Da sich dadurch die Größe aller Texte und somit auch die Größe der Komponenten potentiell ändern, passt das bisherige Layout aller Komponenten im Fenster potentiell nicht mehr und muss aktualisiert werden durch Aufruf von validate.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( comp1, BorderLayout.NORTH );
frame.add ( comp2, BorderLayout.NORTH );
frame.add ( comp3, BorderLayout.WEST );
frame.add ( comp4, BorderLayout.EAST );
frame.add ( comp5, BorderLayout.SOUTH );
frame.add ( comp6, BorderLayout.CENTER );
frame.setVisible ( true );
```

Wir wenden jetzt beispielhaft die zweite add-Methode an.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( comp1, BorderLayout.NORTH );
frame.add ( comp2, BorderLayout.NORTH );
frame.add ( comp3, BorderLayout.WEST );
frame.add ( comp4, BorderLayout.EAST );
frame.add ( comp5, BorderLayout.SOUTH );
frame.add ( comp6, BorderLayout.CENTER );
frame.setVisible ( true );
```

Wir hatten schon gesagt, dass jeder Container zu jedem Zeitpunkt genau einen Layoutmanager hat. Wenn ein Objekt von Container oder von einer Klasse, die von Container abgeleitet ist, eingerichtet wird, wird intern ein LayoutManager eingerichtet. Von welcher LayoutManager-Klasse der ist, unterscheidet sich für die verschiedenen von Klasse Container abgeleiteten Klassen. Für die Klassen Window und Frame ist die Klasse BorderLayout voreingestellt.

LayoutManager

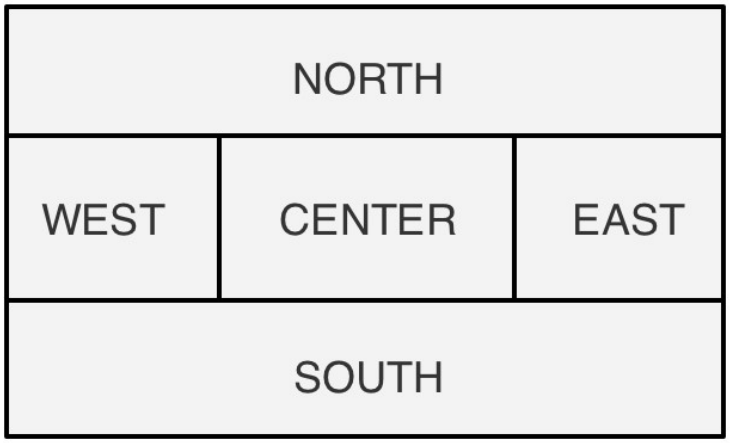
TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( comp1, BorderLayout.NORTH );
frame.add ( comp2, BorderLayout.NORTH );
frame.add ( comp3, BorderLayout.WEST );
frame.add ( comp4, BorderLayout.EAST );
frame.add ( comp5, BorderLayout.SOUTH );
frame.add ( comp6, BorderLayout.CENTER );
frame.setVisible ( true );
```

Die möglichen Informationen, die der LayoutManager namens BorderLayout verarbeiten kann, sind als Klassenkonstanten von BorderLayout vordefiniert. Alle diese Klassenkonstanten sind vom Typ String, passen also unter den formalen Typ Object des zweiten Parameters. Was diese fünf Konstanten besagen, sehen wir uns auf der nächsten Folie an.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT



Die LayoutManager-Klasse BorderLayout unterteilt die Fläche in fünf Bereiche wie hier angedeutet. Die Höhe und Breite jedes dieser Bereiche hängt davon ab, wie groß die Gesamtfläche ist und wie viele und wie große Komponenten in diesen Bereich eingefügt wurden. Aber die Größen der fünf Bereiche werden auch untereinander austariert, so dass letztendlich das Layout jedes Bereichs auch von den Inhalten der anderen vier Bereiche abhängt. Das wird alles automatisch durch Klasse BorderLayout geleistet.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( comp1 );
frame.add ( comp2, BorderLayout.CENTER );
frame.setVisible ( true );
```

Wie passt nun die erste Methode add hier hinein? Nun, wird keiner der fünf Bereiche angegeben, dann wird implizit der Bereich CENTER gewählt. In unseren vorangehenden Beispielen wurden also alle Komponenten nicht einfach in das Fenster eingefügt, wie wir es bisher formuliert haben, sondern genauer in den Bereich CENTER von BorderLayout. Da die anderen vier Bereiche leer waren, wurde sie von BorderLayout auf Größe 0 geschrumpft – WEST und EAST auf *Breite* 0 und NORTH und South auf *Höhe* 0. Daher ist dann die Einteilung in fünf Bereiche nicht auf dem Bildschirm sichtbar.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere LayoutManager-Klassen (Auswahl):

- **BoxLayout**
- **GridLayout**
- **FlowLayout**
- **CardLayout**

Wahrscheinlich ist fast immer BorderLayout die richtige Wahl für ein Fenster, aber nicht unbedingt für einzelne Container innerhalb eines Fensters. Wir sprechen nur ganz kurz ein paar andere häufig genutzte LayoutManager-Klassen an.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere LayoutManager-Klassen (Auswahl):

- **BoxLayout**
- **GridLayout**
- **FlowLayout**
- **CardLayout**

Damit werden die einzelnen Komponenten einfach in einer Reihe nacheinander ausgelegt. Man kann im Konstruktor angeben, ob die Komponenten horizontal oder vertikal ausgelegt werden.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere LayoutManager-Klassen (Auswahl):

- **BoxLayout**
- **GridLayout**
- **FlowLayout**
- **CardLayout**

Hier werden die Komponenten matrixartig neben- und übereinander platziert, wie auf einer Telefontastatur, daher grid für Gitter. Die Anzahl Zeilen und Spalten dieser Gitterstruktur werden im Konstruktor angegeben.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere LayoutManager-Klassen (Auswahl):

- **BoxLayout**
- **GridLayout**
- **FlowLayout**
- **CardLayout**

Während BorderLayout, BoxLayout und GridLayout sich die Freiheit herausnehmen, die Größe jeder einzelnen zu platzierenden Komponente nach der Gesamtsituation selbst zu definieren, wählt FlowLayout für jede Komponente die bestmögliche Größe, die mit Methode getPreferredSize von Klasse Component abgefragt werden kann.

Die Komponenten werden in einer Zeile nebeneinander ausgelegt, und wann immer der verfügbare Platz nicht für die nächste Komponente ausreicht, wird eine neue Zeile aufgemacht. Die Platzierung kann man als analog zum zeilenweisen Schreiben eines Textes bei fester Spaltenbreite ansehen.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere LayoutManager-Klassen (Auswahl):

- **BoxLayout**
- **GridLayout**
- **FlowLayout**
- **CardLayout**

Diese LayoutManager-Klasse weicht von den anderen ab. Sie zeigt nicht alle Komponenten zugleich, sondern nacheinander. Mit Methoden first und last kann man sich die erste beziehungsweise letzte Komponente anzeigen lassen, mit next und previous die jeweils nächste beziehungsweise vorhergehende.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( ..... );
frame.add ( ..... );
frame.add ( ..... );
frame.setVisible ( true );
frame.add ( ..... );
frame.validate ();
frame.setFont ( ..... );
frame.validate ();
```

Da das Thema sehr wichtig ist, kommen wir noch einmal auf Methode validate von Klasse Component zu sprechen.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( ..... );
frame.add ( ..... );
frame.add ( ..... );
frame.setVisible ( true );
frame.add ( ..... );
frame.validate ();
frame.setFont ( ..... );
frame.validate ();
```

Aber das Layout nach den drei ersten add ist noch nicht validiert, zudem invalidieren beide Aufrufe das Layout.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( ..... );
frame.add ( ..... );
frame.add ( ..... );
frame.setVisible ( true );
frame.add ( ..... );
frame.validate ();
frame.setFont ( ..... );
frame.validate ();
```

Da das Fenster sichtbar ist, sollte Methode validate aufgerufen werden, um das Layout neu zu berechnen.

LayoutManager

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Frame frame = new Frame ( “Hello World“ );
frame.add ( ..... );
frame.add ( ..... );
frame.add ( ..... );
frame.setVisible ( true );
frame.add ( ..... );
frame.validate ();
frame.setFont ( ..... );
frame.validate ();
```

Nach Anwendungen von Methoden, die die Anzahl oder Größe der einzelnen Komponenten beeinflussen könnten, muss *immer* Methode validate aufgerufen werden, um das Layout wieder zu richten. Das hatten wir in den bisherigen Beispielen sträflich ausgelassen, um die Darstellung einfach zu halten.

Damit ist der Abschnitt zur Hierarchie und zum Interface LayoutManager beendet.

TECHNISCHE UNIVERSITÄT DARMSTADT

Java Swing

Oracle Docs: die Klassen in javax.swing

Das Package java.awt ist weiterhin die Grundlage in Java für alles, was mit GUIs zu tun hat. Aber obendrauf wurde eine zweite GUI-Bibliothek definiert, deren Klassen und Interfaces dasselbe leisten wie die in java.awt, aber noch einiges mehr an Funktionalität bieten. Im Regelfall nimmt man die Klassen und Interfaces aus Java Swing, aber dahinter steht – teilweise sichtbar – doch weiterhin das Package java.awt.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

Zunächst einmal ein erster grober Überblick.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- **JButton**
- **JCheckBox**
- **JLabel**
- **JList<T>**
- **JScrollBar**
- **JTextComponent**
  - **JTextArea**
  - **JTextField**

Dies ist das zentrale Package für Java Swing. Links sehen Sie zwei der konkreten Anknüpfungspunkte zwischen AWT und Swing:

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

Klasse JFrame in Swing ist von Klasse Frame in AWT abgeleitet, kann also alles, was Frame kann, aber auch noch ein bisschen mehr. Auf dieses Mehr kommen wir später zu sprechen.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

JComponent ist von Container abgeleitet, hat also neben den Funktionalitäten von Component auch die von Container.

***Nebenbemerkung*: Dass JComponent von Container, nicht von Component abgeleitet ist, ist sicherlich verwunderlich. Es hat auch keinen konzeptuellen Grund, sondern ist einfach ein Workaround. Eigentlich wäre eine Klasse JContainer sinnvoll, die einerseits von JComponent aus Swing, andererseits von Container aus AWT erbt. Aber Mehrfachvererbung gibt es in Java nun einmal nicht. Also hat man kurzerhand die Klasse JContainer fallen gelassen und die Funktionalität von Container mittels Ableitung zusätzlich in JComponent gepackt und so dieselben Ziele zwar etwas unschön, aber eben ohne Mehrfachvererbung zu erreichen.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

Die Klassen für die verschiedenen Arten von GUI-Komponenten sind hingegen wieder völlig analog zu den entsprechenden Klassen in AWT gebildet und von der übergeordneten Komponentenklasse abgeleitet. Auch diese Klassen finden sich in Package javax.swing.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

Analog zu AWT ist JTextComponent von JComponent und sind JTextArea und JTextField von JTextComponent abgeleitet.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

In AWT sind die beiden Klassen Button und CheckBox direkt von Klasse Component abgeleitet. Die beiden Klassen JButton und JCheckBox sind nicht direkt von JComponent abgeleitet, sondern indirekt: JButton über eine Zwischenklasse namens AbstractButton, JCheckBox ebenfalls über AbstractButton, aber dann sogar noch über eine zweite Klasse namens JToggleButton. Von diesen beiden Zwischenklassen sind noch weitere Klassen für andere Arten von GUI-Komponenten abgeleitet.

***Nebenbemerkung*: Der Grund für die Einführung dieser Zwischenklassen ist, dass man noch weitere Arten von GUI-Komponenten identifiziert hat, die man gerne als Java-Klassen realisieren möchte und die so eng mit JButton beziehungsweise JCheckBox verwandt sind, dass es sich gelohnt hat, die Gemeinsamkeiten in die Zwischenklassen AbstractButton und JToggleButton herauszufaktorisieren.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

In javax.swing:

JFrame extends java.awt.Frame

JComponent extends java.awt.Container

Von JComponent abgeleitet:

- JButton
- JCheckBox
- JLabel
- JList<T>
- JScrollBar
- JTextComponent
  - JTextArea
  - JTextField

Noch ein Detail: Im Gegensatz zu List in AWT ist JList in Swing nun generisch. Damit kann man eine Liste von beliebigem Referenztyp darstellen. Wenn man nichts an der vorgegebenen Konfiguration ändert, stellt Methode paint von JList einfach für alle Listenelemente die Rückgabe der Methode toString, die ja von Klasse Object an alle Referenztypen vererbt wird, in der momentan voreingestellten Schriftart und Schriftgröße auf dem Bildschirm dargestellt.

***Nebenbemerkung*: Häufig ist die Rückgabe von toString keine adäquate Darstellung der Listenelemente. Wenn Sie mehr darüber erfahren möchten, wie eine andere Darstellungsweise realisiert werden kann, nutzen Sie die Dokumentation des Interface ListCellRenderer im Package javax.swing als Einstiegspunkt.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet über Component hinaus: Tool Tips

```
JButton button = new JButton ( “Buy!“ );
button.setToolTipText ( “Sends your purchase order“ );
```

Wir schauen uns jetzt ein bisschen genauer an, was Klasse JComponent in Swing gegenüber Klasse Component in AWT mehr zu bieten hat, aber auch wieder nur ausgewählte Aspekte. Ein erstes einfaches Beispiel sind Tool Tips.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet über Component hinaus: Tool Tips

```
JButton button = new JButton ( “Buy!“ );
button.setToolTipText ( “Sends your purchase order“ );
```

Als konkretes Beispiel nehmen wir JButton her, wir hätten aber auch jede andere von JComponent direkt oder indirekt abgeleitete Klasse hernehmen können.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet über Component hinaus: Tool Tips

```
JButton button = new JButton ( “Buy!“ );
button.setToolTipText ( “Sends your purchase order“ );
```

Mit diesem Aufruf wird ein Mechanismus eingerichtet, mit dem Sie sicherlich vertraut sind: Wenn Sie die Maus auf den Button ziehen und dort kurz pausieren, dann wird in einem separaten kleinen Popup-Fenster der Text angezeigt, der als Parameter der Methode setToolTipText übergeben wird. Wird null anstelle eines String-Objektes übergeben, dann wird der Tool Tip für diesen Button ausgeschaltet. Vor dem ersten Aufruf von setToolTipText ist er tatsächlich ausgeschaltet.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Komponenten innerhalb eines Fensters möchte man häufig gerne durch rechteckige Ränder vom Rest des Fensters abgrenzen, und man möchte dann auch gerne den Stil auswählen, in dem der Rand angezeigt wird.

***Achtung*: Ränder von Komponenten sind nicht zu verwechseln mit den einheitlich stilisierten Rändern um Fenster vom Typ Frame oder JFrame. Letztere werden vom Window Manager verwaltet, wie wir besprochen hatten.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Klasse JComponent hat eine Methode setBorder. Der Parameter spezifiziert, wie der Rand dieser Komponente aussehen soll.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Aus Gründen, die hier zu weit führen würden, wird allgemein empfohlen, die Methoden von Klasse BorderFactory zu verwenden, solange man mit der Auswahl von Randarten zufrieden ist, die durch die Methoden von BorderFactory erzeugt werden können.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Dies sind ein paar ausgewählte Beispiele für Klassenmethoden von BorderFactory, die Rückgabetyp Border haben.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Mit dieser Methode wird ganz primitiv ein rechteckiger Rand erzeugt, der aus vier Linien gleicher Farbe und Dicke besteht, je eine in jeder Himmelsrichtung. Die beiden Parameter geben die Farbe und die Dicke der Randlinien an.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Sie kennen das, wenn Komponenten einen gewissen 3D-Effekt dadurch erzeugen, dass es durch die Randgestaltung so aussieht, als wäre diese Komponente ein klein wenig tiefer oder höher als ihre Umgebung. Das nennt man im Englischen bevel border. Bei der hier zweimal aufgerufenen Methode können Sie angeben, ob die Komponente etwas tiefer oder etwas höher erscheinen soll.

***Nebenbemerkung*: Im Prinzip können an diesem 3D-Effekt bis zu vier Farben beteiligt sein, daher hat BorderFactory eine weitere Klassenmethode desselben Namens, der noch vier weitere Parameter hat, alle vom formalen Typ Color.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Im Unterschied zu einer bevel border erscheint bei einer etched border nur der Rand selbst etwas tiefer oder höher.

***Nebenbemerkung*: Auch hier kann man die Farben mit anderen Klassenmethoden von BorderFactory mit demselben Namen auch die Farben festlegen.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Randdarstellungen

setBorder (Border border)

BorderFactory.createLineBorder (Color.RED, thickness)

BorderFactory.createBevelBorder (BevelBorder.LOWERED)

BorderFactory.createBevelBorder (BevelBorder.RAISED)

BorderFactory.createEtchedBorder (EtchedBorder.LOWERED)

BorderFactory.createEtchedBorder (EtchedBorder.RAISED)

BorderFactory.createEmptyBorder ()

Schlussendlich erlaubt *diese* Methode, die Randgestaltung wieder auf den Ausgangszustand zurückzusetzen.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

```
UIManager.setLookAndFeel ( String name )
        throws UnsupportedLookAndFeelException,
                ClassNotFoundException,
                InstantiationException,
                IllegalAccessException

UIManager.getSystemLookAndFeelClassName ()
```

Gerne möchte man die Gesamterscheinung der GUI kontrollieren, das Look&Feel. In der Regel will man sich entweder an den Standard auf der verwendeten Plattform anpassen, oder man will plattformübergreifend einen eigenen Stil entwickeln. Auf dieser Folie sehen Sie Code, mit dem das GUI sich auf jeder möglichen Plattform an den jeweiligen Standard dieser Plattform anpassen lässt.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

```
UIManager.setLookAndFeel ( String name )
        throws UnsupportedLookAndFeelException,
               ClassNotFoundException,
               InstantiationException,
               IllegalAccessException

UIManager.getSystemLookAndFeelClassName ()
```

Dazu ruft man diese Klassenmethode auf. Klasse UIManager findet sich ebenfalls in Package javax.swing.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

```
UIManager.setLookAndFeel ( String name )
    throws UnsupportedLookAndFeelException,
           ClassNotFoundException,
           InstantiationException,
           IllegalAccessException

UIManager.getSystemLookAndFeelClassName ()
```

Allerdings müssen diese vier Exception-Klassen gefangen oder weitergereicht werden, obwohl wir uns in diesem Beispiel eigentlich sicher sein können, dass keine Exception geworfen wird.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

```
UIManager.setLookAndFeel ( String name )
        throws UnsupportedLookAndFeelException,
                ClassNotFoundException,
                InstantiationException,
                IllegalAccessException

UIManager.getSystemLookAndFeelClassName ()
```

Die unten gezeigte Klassenmethode der Klasse UIManager liefert einen String zurück, der als Parameter von setLookAndFeel genau den Standard auf der momentanen Plattform zum Look&Feel dieser GUI macht.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

UIManager.setLookAndFeel (LookAndFeel lookAndFeel)
throws UnsupportedLookAndFeelException

UIManager.setLookAndFeel (new MetalLookAndFeel ())

Eine zweite Klassenmethode desselben Namens setLookAndFeel kommt zum Einsatz, wenn wir einen plattformübergreifenden Java-Standard für das Look&Feel wählen möchten.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

```
UIManager.setLookAndFeel ( LookAndFeel lookAndFeel )
    throws UnsupportedLookAndFeelException

UIManager.setLookAndFeel ( new MetalLookAndFeel () )
```

Package javax.swing enthält auch eine Klasse LookAndFeel, deren Name wohl selbsterklärend ist. Das ist der formale Parametertyp dieser Methode setLookAndFeel.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

UIManager.setLookAndFeel (LookAndFeel lookAndFeel)
throws UnsupportedLookAndFeelException

UIManager.setLookAndFeel (new MetalLookAndFeel ())

Der plattformübergreifende Java-Standard heißt Metal und wird durch Wahl der indirekt von Klasse LookAndFeel abgeleiteten Klasse MetalLookAndFeel als aktualen Parametertyp eingerichtet.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Look and Feel

UIManager.setLookAndFeel (LookAndFeel lookAndFeel)
throws UnsupportedLookAndFeelException

UIManager.setLookAndFeel (new MetalLookAndFeel ())

Bei der ersten Methode setLookAndFeel war der Parameter ein String, hier ist er ein Objekt einer genau zu diesem Zweck maßgeschneiderten Klasse. In diesem Fall können viele Ausnahmefälle gar nicht erst auftreten, so dass hier nur noch diese eine Exception geworfen werden kann, nämlich wenn das gewählte Look&Feel auf dieser Plattform nicht unterstützt wird. Beim Java-Standard Metal kann das natürlich nicht passieren, aber gefangen oder weitergereicht werden muss diese Exception-Klasse dennoch.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Unser Beispiel Passworteingabe für das Interface KeyListener hatte nur auf eine bestimmte Taste reagiert, nämlich Enter. Situationen dieser Art, nämlich dass bestimmte Aktionen an bestimmte Tasten oder Tastenkombinationen gebunden werden sollen, sind so häufig, dass die dafür notwendige Funktionalität in Swing so integriert worden ist, dass die ganzen technischen Details mit Listener und so weiter, um die wir uns bei AWT selbst kümmern mussten, dahinter versteckt sind.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Wir sehen uns eine der gebotenen Möglichkeiten an. Begonnen wird damit, die Tastenkombination als String zu kodieren. Die Kodierungsregeln finden Sie in der Dokumentation der Klasse KeyStroke in Package javax.swing.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Diese als String kodierte Tastenkombination wird dann durch die dafür vorgesehene Methode in ein entsprechendes Objekt eben dieser Klasse KeyStroke umgewandelt.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = "alt shift X";
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Die Klasse JComponent – und damit auch jede von JComponent abgeleitete Klasse – enthält zwei interne Komponenten namens inputMap und actionMap, in denen man Paare bestehend aus einem Schlüsselwert und einer zugehörigen Information speichern kann, wobei die Schlüsselwerte in einer solchen Map paarweise verschieden sein müssen.

***Erinnerung*: Das Konzept Map haben wir schon im gleichnamigen Abschnitt in Kapitel 07 gesehen.**

Die Klassen InputMap und ActionMap implementieren zwar nicht das dort vorgestellte Interface Map, sind aber konzeptuell sehr ähnlich und haben sehr ähnliche Aufgaben.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Beide Klassen haben eine Methode put, um ein neues Paar einzufügen. Ist der Schlüsselwert schon in der Map, dann wird die zugehörige Information mit dem zweiten Parameter überschrieben.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Der zweite Parameter bei der InputMap und der erste Parameter bei der ActionMap sind wohl meistens Strings, können aber Objekte von beliebigen Klassen sein. Sie müssen nur wertgleich sein.

***Erinnerung*: In Kapitel 03b hatten wir den Unterschied zwischen Objektidentität und Wertgleichheit im gleichnamigen Abschnitt herausgearbeitet.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Und so, indirekt über keyStrokeString, ist eine Aktion mit einer Tastaturkombination verbunden.

***Nebenbemerkung*: Beim Design der Klassen zur Unterstützung von Key Bindings hätte man anstelle dieser beiden Maps auch eine einzige Map einführen können, die direkt keyStroke mit action verknüpft, ohne Umweg über keyStrokeString. Man hat sich dagegen entschieden, da es in komplexeren Situationen als hier nicht mehr unbedingt eine Eins-zu-Eins-Beziehung zwischen einer InputMap und einer ActionMap gibt. Mit dieser Zerlegung in InputMap und ActionMap ist es beispielsweise möglich, dass eine InputMap mehrere Objekte von ActionMap bedient, die mit derselben Tastaturkombination dann unterschiedliche Aktionen verknüpfen. Umgekehrt kann es so auch mehrere InputMap-Objekte für eine einzelne ActionMap geben.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Für Aktionen gibt es im Package javax.swing das Interface Action, das das uns vertraute Interface ActionListener aus dem Package java.awt erweitert. Die Methode actionPerformed, die schon in ActionListener vorhanden war, führt die Aktion aus.

Für die verschiedensten Standardsituationen, was man so alles durch eine Tastenkombination vielleicht an Aktionen auslösen möchte, sind spezifische Klassen in der Java-Standardbibliothek vorhanden, die Action implementieren. Wir greifen nur ein einfaches Beispiel heraus, dass ohne weitere Konfigurierungen sofort verwendbar ist: Die Methode actionPerformed der Klasse UnderlineAction sorgt dafür, dass Texte von jetzt an unterstrichen werden beziehungsweise von jetzt an *nicht mehr* unterstrichen sind.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet: Key Bindings

```
String keyStrokeString = “alt shift X“;
KeyStroke keyStroke = KeyStroke.getKeyStroke ( keyStrokeString );
textArea.getInputMap().put ( keyStroke, keyStrokeString );
StyledEditorKit.UnderlineAction action
            = new StyledEditorKit.UnderlineAction ();
textArea.getActionMap().put ( keyStrokeString, action );
```

Die Klasse UnderlineAction ist in die Klasse StyledEditorKit eingebettet. Diese Klasse enthält einige nützliche Funktionalität rund um das Edieren von Texten, speziell zu Stilmitteln, wozu ja auch Unterstreichungen gehören.

***Erinnerung*: Kapitel 09, Abschnitt „Verschachtelte Klassen (nested classes)“.**

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet:

- **Drag&Drop**
- **Assistive Technologies**

JFrame bietet

- **Separierung von Hauptmenü und Rest des Fensters**

Zwei ausgewählte weitere Neuerungen bei Klasse JComponent gegenüber Klasse Component sprechen wir nur noch kursorisch an, dazu eine der Neuerungen von Klasse JFrame gegenüber Klasse Frame.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet:

- **Drag&Drop**
- **Assistive Technologies**

JFrame bietet

- **Separierung von Hauptmenü und Rest des Fensters**

Für Drag&Drop müssen Sie nichts tun, das bekommen Ihre Swing-Applikationen geschenkt. Sie könnten den Mechanismus nach eigenen Vorstellungen verändern, dazu sind Möglichkeiten vorgesehen, das wird von Oracle aber eher nicht empfohlen.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet:

- **Drag&Drop**
- **Assistive Technologies**

JFrame bietet

- **Separierung von Hauptmenü und Rest des Fensters**

JComponent bietet diverse Unterstützungsmöglichkeiten, damit auch Personen mit verschiedensten Handicaps mit Ihrer GUI arbeiten können.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

JComponent bietet:

- **Drag&Drop**
- **Assistive Technologies**

JFrame bietet

- **Separierung von Hauptmenü und Rest des Fensters**

Dieser Aspekt von JFrame ist nicht nur, aber insbesonders nützlich auf Plattformen, auf denen das Menü zu einem Fenster nicht oben im Fenster erscheint, sondern beispielsweise am oberen Rand des Bildschirms. Durch diese Separierung ist es umstandslos möglich, dem Menü eine völlig andere Position zu geben als dem eigentlichen Fenster.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- **JFormattedTextField**
- **JPasswordField**
- **JRadioButton**
- **JToolBar**
- **JSlider**
- **Popup**
- **JTable**

Noch ein paar weitere ausgewählte Klassen seien kurz angesprochen.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- **JFormattedTextField**
- **JPasswordField**
- **JRadioButton**
- **JToolBar**
- **JSlider**
- **Popup**
- **JTable**

Diese direkt von JTextField abgeleitete Klasse erlaubt es, den einzugebenden Text vorzuformatieren, zum Beispiel als Datumsangabem, wodurch dann auch tatsächlich nur Datumseingaben akzeptiert werden. Die Formatierungsregeln sind an eine Klasse Formatter in Package java.util delegiert.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- JFormattedTextField
- JPasswordField
- JRadioButton
- JToolBar
- JSlider
- Popup
- JTable

Der Name dieser Klasse ist wohl selbsterklärend.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- JFormattedTextField
- JPasswordField
- JRadioButton
- JToolBar
- JSlider
- Popup
- JTable

Eine solche Komponente ist typischerweise dargestellt als ein kleiner, anklickbarer Bereich, häufig ein Kreis oder Quadrat, und einem erklärenden Text in unmittelbarer Nähe. Objekte von Klasse RadioButton werden in der Regel nicht als eigenständige GUI-Komponenten verwendet, sondern im Rahmen eines Objekts der Klasse ButtonGroup. Nur eines der RadioButton-Objekte in einer ButtonGroup kann angeklickt sein; klickt man ein anderes an, ist das vorhergehende nicht mehr angeklickt.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- JFormattedTextField
- JPasswordField
- JRadioButton
- JToolBar
- JSlider
- Popup
- JTable

Eine vereinfachte Möglichkeit, Standard-Menüs zu arrangieren.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- JFormattedTextField
- JPasswordField
- JRadioButton
- JToolBar
- JSlider
- Popup
- JTable

Eine Klasse für Schieberegler – sicherlich sinnvoller als unser Versuch weiter vorne in diesem Kapitel, eine Scrollbar als Slider zu verwenden.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- **JFormattedTextField**
- **JPasswordField**
- **JRadioButton**
- **JToolBar**
- **JSlider**
- **Popup**
- **JTable**

Popup-Fenster werden ebenfalls unterstützt.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

Weitere nützliche GUI-Komponentenklassen (Auswahl):

- JFormattedTextField
- JPasswordField
- JRadioButton
- JToolBar
- JSlider
- Popup
- JTable

Ein Objekt der Klasse JTable repräsentiert eine gewöhnliche Tabelle. Wir schauen uns diese Klasse auf der nächsten Folie beispielhaft noch etwas genauer an.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Genauer gesagt, sehen wir auf dieser Folie die wichtigsten Ausschnitte aus *einem* typischen Anwendungsfall.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Der erste Parameter des Konstruktors ist die Matrix der Tabelleneinträge. Wie der hier von uns gewählte Name der Variablen schon suggeriert, werden mit dem *zweiten* Parameter die Namen der einzelnen Spalten eingelesen.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Daher ist der erste Parameter ein Array von Arrays und der zweite Parameter ein einfaches Array.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Beides ist maximal flexibel gehalten dadurch, dass die formalen Parameter jeweils Object als Komponententyp haben.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Klasse JScrollPane kapselt ein Objekt von Klasse Component aus dem Packaga java.awt beziehungsweise ein Objekt einer direkt oder indirekt von Component abgeleiteten Klasse ein. Hier ist das Klasse JTable, die direkt von JComponent und daher indirekt über Container und JComponent von Component abgeleitet ist. Und zwar zeigt JScrollPane nur einen Ausschnitt aus der im Konstruktor übergebenen GUI-Komponente und bietet eine horizontale und eine vertikale ScrollBar an, mit denen der Ausschnitt verschoben werden kann.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Diese Methode ist eine Besonderheit von Klasse JTable: Es kann ja sein, dass die Tabelle in der präferierten Größe der Zellen vertikal kleiner als der Ausschnitt ist. Mit aktualem Parameter true wird dafür gesorgt, dass die Tabelle in diesem Fall soweit vertikal gestreckt wird, dass sie die gesamte Höhe ausfüllt.

Java Swing

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Object[ ][ ] entries = .....;
Object[ ] columnNames = .....;
JTable table = new JTable ( entries, columnNames );
JScrollPane scrollPane = new JScrollPane ( table );
table.setFillsViewportHeight ( true );
```

```
int[ ] selectedRows = table.getSelectedRows ();
int[ ] selectedColumns = table.getSelectedColumns ();
```

Im Prinzip kann der Nutzer mit GUI-Komponenten vom Typ JTable so umgehen, wie Sie das von Tabellenprogrammen typischerweise gewohnt sind. Und auf Programmebene kann man wie auch bei anderen Arten von GUI-Komponenten auf Nutzerinteraktionen reagieren.

Diese beiden Zeilen sind nur zwei einfache Beispiele: Die Indizes derjenigen Zeilen beziehungsweise Spalten, die momentan angeklickt und damit ausgewählt sind, werden hier zurückgeliefert, und das Programm kann mit diesen Zeilen und Spalten dann irgendetwas machen.

Damit ist dieser Abschnitt und das ganze Kapitel beendet.