Grundlagen der Informatik

Bruno Lurz
Helmut Herold
Martin Lurz
Jürgen Wohlrab

Pearson

EXTRAS ONLINE

Zugangscode

Falls Sie beim Kauf Ihres eBooks keinen Zugangscode erhalten haben, kontaktieren Sie uns bitte über die folgende Seite und halten Sie Ihre Rechnung/Bestellbestätigung bereit:
https://www.pearson.de/ebook-zugangscode

```
public class Schnittstelle { // ... Startklasse
  public static void main (String args[]) {
     Punkt    p1 = new Punkt(50, 100);
     Punkt    p2 = new Punkt(150, 101);
     Punkt    p3 = new Punkt(50, 100);
     Viereck  v1 = new Viereck(100, 100);
     Viereck  v2 = new Viereck(20, 120);
     Viereck  v3 = new Viereck(80, 125);
      if (p1.istKleiner(p2)) System.out.println("p1 ist kleiner als p2");
      if (!p2.istKleiner(p1)) System.out.println("p2 ist groesser als p1");
      if (p3.istGleich(p1)) System.out.println("p1 ist gleich p3");

      if (v2.istKleiner(v1)) System.out.println("v2 ist kleiner als v1");
      if (!v1.istKleiner(v2)) System.out.println("v1 ist groesser als v2");
      if (v3.istGleich(v1)) System.out.println("v1 ist gleich v3");
  }
}
```

Nachfolgend ist die Ausgabe zu Listing 7.16 gezeigt:

```
p1 ist kleiner als p2
p2 ist groesser als p1
p1 ist gleich p3
v2 ist kleiner als v1
v1 ist groesser als v2
v1 ist gleich v3
```

Abbildung 7.68 zeigt die Darstellung von Schnittstellen in UML.

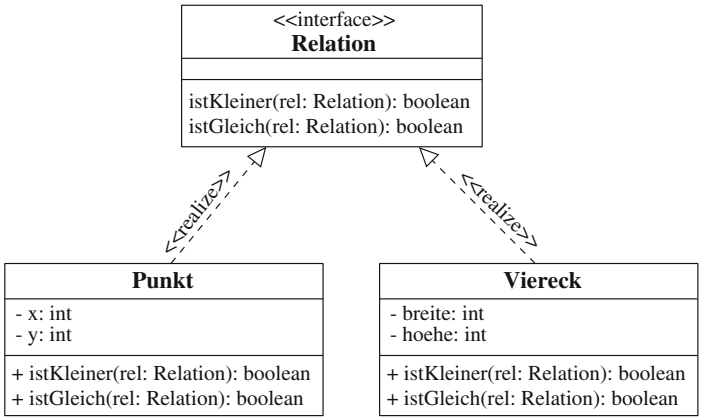

Abbildung 7.68: Schnittstellen in UML

7.6.6 GUI-Programmierung in Java

Unter GUI-Programmierung (*GUI = Graphical User Interface*) versteht man die Programmierung von Anwendungen mit grafischen Benutzeroberflächen.

Java stellt dazu in der Klassenbibliothek die so genannten AWT-Klassen *(Abstract Windowing Toolkit)* und die *JFC*- bzw. *Swing*-Klassen zur Verfügung. Das Aussehen der Objekte unter dem AWT hängt vom Betriebssystem ab, während die Swing-Objekte auf allen Betriebssystemen gleich aussehen. Die Namen der AWT- und der Swing-Klassen sind weitgehend identisch, nur dass bei den Swing-Klassen dem Klassennamen noch der Buchstabe `J` vorangestellt ist. Wegen der beschränkten Möglichkeiten des AWT wurden ursprünglich die so genannten *Java Foundation Classes (JFC)* entwickelt, die Swing enthalten. Später hat sich der Name *Swing* für *JFC* eingebürgert.

Ein erstes einfaches Fenster

Ein normales Fenster (*Window*), das verkleinert, vergrößert und verschoben werden kann, wird durch die Klasse `Frame` bzw. `JFrame` zur Verfügung gestellt. Abbildung 7.69 zeigt, wie man ein einfaches Fenster programmieren kann, in dem der Text „`Erstes Fenster`“ steht und dessen Titelleiste die Beschriftung „`AWT-Fenster`“ bzw. „`Swing-Fenster`“ besitzt.

```
/* FensterAWT.java */
import java.awt.*;

class FensterA extends Frame {
    FensterA(String title) {
        super(title); // Titelleiste
        setSize(200, 100); // Groesse
        setVisible(true); // nun sichtbar
        add(new Label("Erstes Fenster"));
    }
}
public class FensterAWT {
  public static void main(String arg[]) {
      FensterA meinFenster
          = new FensterA("AWT-Fenster");
  }
}
```

```
/* FensterSwing.java */
import javax.swing.*;
import java.awt.*;
class FensterS extends JFrame {
    FensterS(String title) {
        super(title); // Titelleiste
        setSize(200, 100); // Groesse
        setVisible(true); // nun sichtbar
        add(new Label("Erstes Fenster"));
    }
}
public class FensterSwing {
  public static void main(String arg[]) {
      FensterS meinFenster
          = new FensterS("Swing-Fenster");
  }
}
```

Abbildung 7.69: Ein erstes einfaches Fenster in Java

Die durch die Listings in Abbildung 7.69 erzeugten Fenster lassen sich jedoch nur dadurch beenden, dass man in der Konsole, in der man dieses Programm gestartet hat, dieses gewaltsam mit der Eingabe von *Strg-C* beendet. In den nachfolgenden Beispielprogrammen wird nun nur noch die neuere `Swing`-Variante verwendet.

Ereignisse (Events)

GUI-Programme werden durch *Ereignisse* gesteuert, auf die das jeweilige Programm entsprechend reagiert. Das betrifft vor allem auch sämtliche Benutzereingaben, wie Tastendrucke oder Mausklicks. *Ereignisse* oder *Events* treten ein, wenn sich der Zustand eines Objekts ändert, weil z. B. ein Button gedrückt wurde, der Inhalt eines Texteingabefeldes geändert, ein Listenelement ausgewählt oder eine Eingabe über die Tastatur erfolgte. Dabei gibt es *Ereignisquellen* (Buttons, Menüs usw.) und *Ereignis-*

empfänger. Die Methoden zum Empfangen und Verarbeiten von Ereignissen sind im Paket `java.awt.event` definiert. Für jede Ereignisklasse sind bestimmte Schnittstellen (*EventListener*) definiert, die die jeweiligen Methoden zum Empfang der gewünschten Ereignisse enthalten. Beispiele hierfür sind:

- `ActionListener`: Ereignisse wie z. B. Anklicken von Buttons oder Auswahl eines Menüeintrags usw.,
- `TextListener`: Text eines Texteingabefeldes wurde geändert,
- `KeyListener`: Taste auf der Tastatur wurde gedrückt,
- `MouseListener` und `MouseMotionListener`: Maus-Ereignisse wie Klicken oder Bewegen,
- `WindowListener`: Ereignisse im Zusammenhang mit Fenstern, wie z. B. Öffnen, Schließen, Aktivieren, Deaktivieren usw. von Fenstern,
- `ItemListener`: Ereignisse wie z. B. Anklicken eines Eintrages in einer Liste,
- `FocusListener`: Tastatur-Fokus hat sich geändert.

Als erstes Beispiel soll dies an einem Button demonstriert werden. Der Ablauf der Ereignis-Behandlung für einen Button ist:

- Das Betätigen eines Buttons erzeugt automatisch ein `ActionEvent`-Objekt.
- Zuständig für den Empfang dieses Events ist ein `ActionListener`-Objekt.
- Empfängt das `ActionListener`-Objekt ein `ActionEvent`-Objekt, so wird automatisch die Methode `actionPerformed(ActionEvent e)` des Listener-Objekts ausgeführt.

Die Vorgehensweise bei der Programmierung ist dabei folgende:

- Es muss ein spezielles `ActionListener`-Objekt erzeugt und beim Button mit der Methode `addActionListener(ActionListener al)` registriert werden, damit es die Events empfangen kann.
- `ActionListener` ist ein Interface. Um ein Objekt zu erzeugen, muss also zunächst eine Klasse zur Verfügung gestellt werden, die dieses Interface implementiert.
- In der vom Interface vorgeschriebenen Methode `actionPerformed()`, die zu implementieren ist, kann festgelegt werden, was bei Eintreten des Ereignisses ausgeführt werden soll.

Listing 7.17 demonstriert die Ereignisbehandlung für einen Button, den es oben im Fenster anzeigt, wobei jeder Mausklick auf diesen Button eine zufällige Hintergrundfarbe für das Fenster nach sich zieht, wie es z. B. in Abbildung 7.70 gezeigt ist. Dieses Beispiel zeigt auch noch die Möglichkeit der gezielten Anordnung von Komponenten in Fenstern mit Hilfe von so genannten *Layout-Manager*-Objekten und das Schließen des Fensters über das allgemeine *Schließen-Icon* in der Titelleiste.

Listing 7.17: `Farbwechsel.java`: Zufällige Hintergrundfarbe bei Button-Klick

```
import java.awt.*;
import java.awt.event.*; // fuer Ereignisse
import javax.swing.*;
public class Farbwechsel extends JFrame {
  private Container c; // Container fuer Button
  private JButton button;
   public Farbwechsel(String titel) {
      super(titel);
      c = getContentPane(); // Container erzeugen
      // .....  1. Button erzeugen
      button = new JButton("Hintergrundfarbe wechseln");
      c.add(button, BorderLayout.NORTH); // Button oben anordnen
      // .....  2. Listener fuer Events dem Button zuteilen
      ButtonListener horcher = new ButtonListener();
      button.addActionListener(horcher);
   }
   // Innere Listener-Klasse
   class ButtonListener implements ActionListener {
      public void actionPerformed(ActionEvent e) {
         float rot   = (float) Math.random(); // Zufallszahl aus Intervall [0,1)
         float gruen = (float) Math.random(); // Zufallszahl aus Intervall [0,1)
         float blau  = (float) Math.random(); // Zufallszahl aus Intervall [0,1)
         Color rgb = new Color(rot, gruen, blau);
         c.setBackground(rgb);
      }
   }
   public static void main(String[] args) {
      Farbwechsel f = new Farbwechsel("Fenster mit Button");
      f.setSize(300, 200);
      f.setVisible(true);
      // Programm nun mit Mausklick auf Schliessen-Button beendbar
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

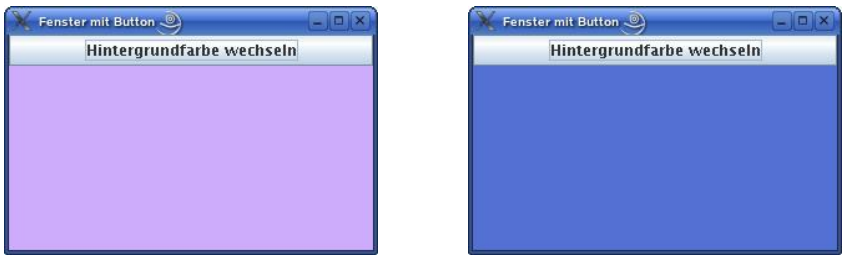

Abbildung 7.70: Bei jedem Button-Klick neue Hintergrundfarbe

Im nächsten Beispiel wird nun die Behandlung von Maus-Ereignissen demonstriert. Der Ablauf der Ereignis-Behandlung für eine Mausaktion ist:

- Eine Aktion mit der Maus (z. B. Klick oder Bewegung) erzeugt ein `MouseEvent`-Objekt.

- Zuständig für den Empfang dieses Events ist ein `Mouse`**Listener**-Objekt.
- Empfängt das `Mouse`**Listener**-Objekt ein `Mouse`**Event**-Objekt, so wird abhängig von der entsprechenden Mausaktion automatisch eine der folgenden Methoden des Listener-Objekts ausgeführt:

  `mouseEntered(Mouse`**Event e)** – Mauszeiger in die Komponente bewegt

  `mouseExited(Mouse`**Event e)** – Mauszeiger aus der Komponente bewegt

  `mousePressed(Mouse`**Event e)** – Maustaste gedrückt

  `mouseReleased(Mouse`**Event e)** – Maustaste losgelassen

  `mouseClicked(Mouse`**Event e)** – Maustaste geklickt

Die Vorgehensweise bei der Programmierung ist dabei folgende:

- Es muss ein spezielles `MouseListener`-Objekt erzeugt und bei der Komponente mit Hilfe der Methode `addMouseListener(MouseListener ml)` als Ereignis-Empfänger registriert werden.
- `MouseListener` ist ein Interface. Um ein Objekt zu erzeugen, muss also zunächst eine Klasse zur Verfügung gestellt werden, die dieses Interface implementiert.
- In der vom Interface vorgeschriebenen Methode `mouseX...()`, die zu implementieren ist, kann festgelegt werden, was bei Eintreten des speziellen Maus-Ereignisses ausgeführt werden soll.

Listing 7.18 demonstriert die Ereignisbehandlung für Maus-Events, wobei es an jeder Stelle im Fenster, an der der Benutzer die Maus klickt, einen blau gefüllten Kreis zeichnet, wie es z. B. links in Abbildung 7.71 zu sehen ist. Verlässt der Benutzer mit der Maus das Fenster, so werden alle diese Kreise durch schwarze Linien miteinander verbunden, wie es z. B. rechts in Abbildung 7.71 gezeigt ist. Bewegt er die Maus wieder in das Fenster, wird der ganze Fensterinhalt gelöscht, und er kann sich erneut mit Mausklicks Punkte malen lassen, die dann beim Verlassen des Fensters alle wieder miteinander verbunden werden.

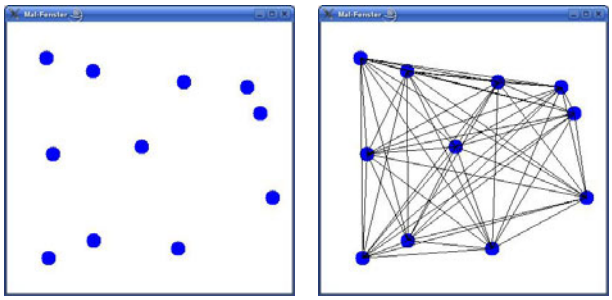

Abbildung 7.71: Punkte bei Mausklicks und Verbinden dieser beim Verlassen des Fensters

Listing 7.18: `Mausevent.java`: Punkte an Mausklicks und Verbinden der Punkte beim Verlassen des Fensters

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

public class Mausevent extends JFrame {
    Container c;
    private int[] x, y;
    private int pktZahl;

    public Mausevent(String titel) {
        super(titel);
        x = new int[10000];
        y = new int[10000];
        pktZahl = 0;
        c = getContentPane(); // Container erzeugen
        // Listener fuer Maus-Events dem Fenster zuteilen
        addMouseListener(new Horch());
    }
    class Horch implements MouseListener { // Innere Horch-Klasse
        public void mouseClicked(MouseEvent e) {
            Graphics g = getGraphics();
            g.setColor(Color.blue);
            g.fillOval(e.getX()-10, e.getY()-10, 20, 20);
            x[pktZahl] = e.getX();
            y[pktZahl] = e.getY();
            if (pktZahl < 10000) pktZahl++;
        }
        public void mouseEntered(MouseEvent e) {
            pktZahl = 0;
            c.setBackground(Color.blue); // um blaue Punkte zu uebermalen
            c.setBackground(Color.white); // nun ganzen Hintergrund weiss
        }
        public void mouseExited(MouseEvent e) {
            Graphics g = getGraphics();
            for (int i=0; i < pktZahl-1; i++)
                for (int j=i+1; j < pktZahl; j++)
                    g.drawLine(x[i], y[i], x[j], y[j]);
        }
        public void mousePressed(MouseEvent e) { }
        public void mouseReleased(MouseEvent e) { }
    }
    public static void main(String[] args) {
        Mausevent f = new Mausevent("Mal-Fenster");
        f.setSize(400, 400);
        f.setVisible(true);
        // Programm nun mit Mausklick auf Schliessen-Button beendbar
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

In Listing 7.18 sind einige neue Konstrukte hinzugekommen, die nachfolgend kurz erläutert werden.

Die Klasse `Graphics` und ihre Methoden

In der Klasse `Graphics` sind alle wichtigen Grafik-Methoden zum Zeichnen zusammengefasst. Um nun in einem Fenster zu zeichnen, benötigt man ein Objekt der Klasse `Graphics`, das man sich mit der Methode `getGraphics()` liefern lassen kann, wie z. B. bei:

```
public void mouseClicked(MouseEvent e) {
    Graphics g = getGraphics();
    g.setColor(Color.blue);
    g.fillOval(e.getX()−10, e.getY()−10, 20, 20);
}
```

Einige wichtige Methoden von `getGraphics()` sind z. B.:

- `setColor(Color farbe)` – setzt die Malfarbe.
- `drawLine(int x1, int y1, int x2, int y2)`
  zeichnet eine Linie von Punkt `(x1,y1)` nach Punkt `(x2,y2)`.
- `drawRect(int x, int y, int b, int h)`
  zeichnet ein Rechteck, dessen linke obere Ecke der Punkt `(x,y)` ist, mit einer Breite von `b` und einer Höhe von `h`. Möchte man ein gefülltes Rechteck, muss man `fillRect()` verwenden.
- `drawOval(int x, int y, int b, int h)`
  zeichnet eine Ellipse bzw. einen Kreis in einem Rechteck, dessen linke obere Ecke der Punkt `(x,y)` ist, und das eine Breite von `b` und eine Höhe von `h` hat. Möchte man eine gefüllte Ellipse bzw. einen gefüllten Kreis, muss man `fillOval()` verwenden.
- `drawString(String str, int x, int y)`
  gibt den String `str` aus, wobei `x` und `y` die Anfangsposition und die Basislinie festlegen, an der der String im Fenster auszugeben ist.
- `setFont(Font font)`
  setzt den Zeichensatz (Font) für die Ausgabe von Strings.

Inzwischen steht eine weitere Klasse `Graphics2D` für Grafik-Operationen zur Verfügung, die mehr Funktionalität anbietet, wie z. B. auch das Setzen der Linienstärke.

Informationen zu Mausaktionen über `MouseEvent`

Maus-Events liefern über das Argument vom Typ `MouseEvent` Informationen zur stattgefundenen Mausaktion, wie z. B. über `e.getX()` und `e.getY()` die Position, an der der Mausklick stattfand.

Adapter-Klassen mit leeren Methoden-Implementierungen

In Listing 7.18 mussten alle von der Interface-Klasse `MouseListener` angebotenen Methoden implementiert werden, selbst wenn sie nicht gebraucht wurden, wie z. B. die beiden folgenden leeren Methoden:

```
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
```

Um nun nicht immer alle Ereignisse der jeweiligen Ereignis-Interfaceklasse vollständig implementieren zu müssen, bietet Java so genannte *Adapter*-Klassen zu den Ereignis-Klassen, die alle im jeweiligen Interface festgelegten Ereignisse mit leeren Methoden implementieren. Bei Verwendung dieser Adapter-Klassen muss man dann nicht alle Ereignisse implementieren, sondern kann sich auf die beschränken, die für den jeweiligen Anwendungsfall von Interesse sind. Hätte man z. B. in Listing 7.18 statt der folgenden Zeile

```
class Horch implements MouseListener {
```

Folgendes angegeben

```
class Horch extends MouseAdapter {
```

so hätte man die (leere) Implementierung der beiden Methoden `mousePressed()` und `mouseReleased()` weglassen können. Diese Vererbung kann man genau dann nutzen, wenn die Klasse nur von einer einzigen Klasse erben soll. Sobald jedoch Methoden mehrerer übergeordneter Klassen implementiert werden sollen, kann nur von einer Klasse geerbt werden. Für die weiteren Fälle müssen dann die Interfaces implementiert werden, da Java keine Mehrfachvererbung wie z. B. C++ zulässt:

```
class Horch extends MouseAdapter implements MouseMotionListener {
```

▶ Übung: Programm zu den Adapter-Klassen

Erstellen Sie ein Programm `Mausevent2.java`, das das Gleiche leistet wie das Programm in Listing 7.18, nur dass es stattdessen die Adapter-Klasse `MouseAdapter` verwendet.

Neben der Interface-Klasse `MouseListener` existiert noch die Interface-Klasse `MouseMotionListener`, die die folgenden beiden Methoden anbietet:

- `mouseMoved(MouseEvent e)` – wird ausgeführt, wenn Maus bewegt wurde,
- `mouseDragged(MouseEvent e)` – wird ausgeführt, wenn Maus mit gedrückter Maustaste bewegt wurde.

Listing 7.19, das das Malen in einem Fenster beim Bewegen der Maus mit gedrückter Maustaste ermöglicht, ist dazu ein Demonstrationsbeispiel.

Listing 7.19: `Mausmal.java`: Malen mit gedrückter Maustaste

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Mausmal extends JFrame {
    Container c;
    private int x, y;
    public Mausmal(String titel) {
        super(titel);
        x = y = 0;
        c = getContentPane(); // Container erzeugen
        // Listener fuer Maus-Events dem Fenster zuteilen
        addMouseListener(new Horch());
        addMouseMotionListener(new Horch());
    }
    class Horch extends MouseAdapter implements MouseMotionListener {
        public void mouseReleased(MouseEvent e) {
            x = y = 0;
        }
        public void mouseDragged(MouseEvent e) {
            Graphics g = getGraphics();
            g.setColor(Color.blue);
            if (x != 0 && y != 0)
                g.drawLine(x, y, e.getX(), e.getY());
            x = e.getX();
            y = e.getY();
        }
        public void mouseMoved(MouseEvent e) { } // hier leere Methode notwendig
    }
    public static void main(String[] args) {
        Mausmal f = new Mausmal("Mal-Fenster");
        f.setSize(400, 400);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Abschließendes Beispiel mit weiteren GUI-Komponenten

Listing 7.20 zeigt weitere GUI-Komponenten in Java, wie z. B. Panels für Unterfenster in einem Hauptfenster, Menüs und ein Dialogfenster. Dieses Programm bietet oben eine Menüleiste an, wie es in Abbildung 7.72 gezeigt ist. Über diese Menüeinträge kann der Benutzer Folgendes veranlassen:

- `Datei/Quit` – Programm beenden.
- `Bearbeiten/Clear` – Inhalt des rechten Panels löschen.
- `Bearbeiten/Neue Farbe` – Neue zufällige Farbe im rechten Panel.
- `Hilfe/Info` – Einblenden eines Dialogfensters (siehe rechts in Abbildung 7.72).

Klickt der Benutzer auf den Button „`Text`“ im rechten Panel, wird ihm im linken Panel auf weißem Hintergrund angezeigt, wie oft er diesen Button schon angeklickt hat. Bei

einem Mausklick in das rechte Panel wird an dieser Stelle die Position ausgegeben, an der dieser Klick stattfand.

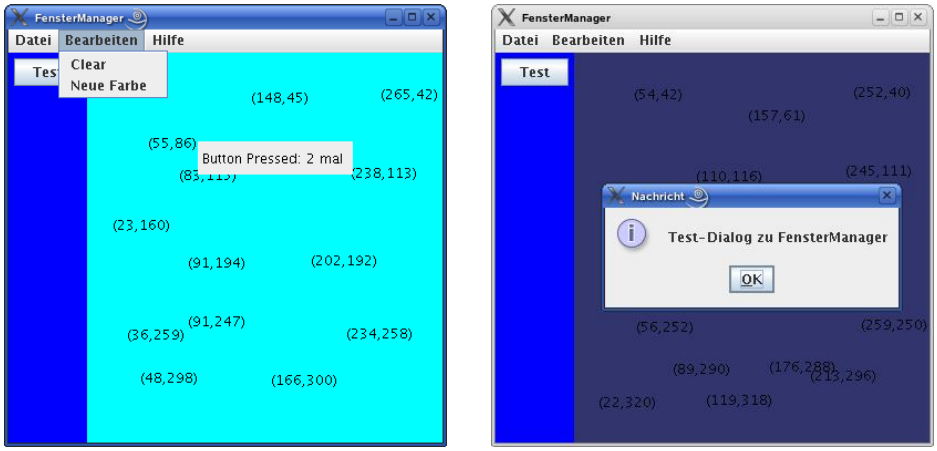

Abbildung 7.72: Panels, Menüs und Dialogfenster

Listing 7.20: `FensterManager.java`: Einige weitere GUI-Komponenten

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

class TeilFenster extends JFrame
{
  public TeilFenster(FensterManager fm, MeinPanel meinPanel)
  {
     super("FensterManager");
     setSize( 400, 400 );
     addWindowListener( fm );

     JPanel hauptPanel = new JPanel(); // Layout: West, North, East, South und Center
     hauptPanel.setLayout( new BorderLayout() );

     JPanel linksPanel = new JPanel();
     linksPanel.setBackground( Color.blue );

     JButton bt = new JButton("Test") ;
     bt.addActionListener( fm );
     bt.setSize( 40, 20 );
     linksPanel.add( bt );

        // ... linksPanel und meinPanel in hauptPanel einordnen
     hauptPanel.add( "West", linksPanel );
     hauptPanel.add( "Center", meinPanel );
       // ... hauptPanel im Fenster einfuegen
     getContentPane().add( hauptPanel );
```

```
        //          ..........................................          Menues
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar( menuBar );

        JMenu mDatei = new JMenu( "Datei" );
        JMenu mBearb = new JMenu( "Bearbeiten" );
        JMenu mHilfe = new JMenu( "Hilfe" );
        menuBar.add( mDatei );
        menuBar.add( mBearb );
        menuBar.add( mHilfe );

        JMenuItem quit = new JMenuItem( "Quit");
        quit.addActionListener( fm );
        mDatei.add( quit );

        JMenuItem clear = new JMenuItem( "Clear");
        clear.addActionListener( fm );
        mBearb.add( clear );

        JMenuItem farbe = new JMenuItem( "Neue Farbe");
        farbe.addActionListener( fm );
        mBearb.add( farbe );

        JMenuItem info = new JMenuItem( "Info..." );
        info.addActionListener( fm );
        mHilfe.add( info );

        setVisible(true);
    }
}
class MeinPanel extends JPanel {
    public MeinPanel( FensterManager fm ) {
        addMouseListener( fm );
        setBackground( Color.cyan );
    }
}
public class FensterManager extends MouseAdapter
                            implements ActionListener, WindowListener {
    static int n = 0;
    static MeinPanel meinPanel;

    public static void main(String argv[]) {
        FensterManager fm = new FensterManager();
        meinPanel         = new MeinPanel( fm );
        TeilFenster fenster = new TeilFenster( fm, meinPanel );
    }

     //          ..........................................          Einzige implementierte Maus–Methode
    public void mousePressed(MouseEvent ev) {
        int x = ev.getX(),
            y = ev.getY();
        Graphics g = meinPanel.getGraphics();
        g.drawString("(" + x + "," + y + ")", x, y);
     }
```

Copyright