

# Einführung in die Informatik - Teil 4b -

## Zeichnen mit der Maus im JBuilder

### Inhalt:

1. Arbeiten mit der Maus
2. Aufgaben

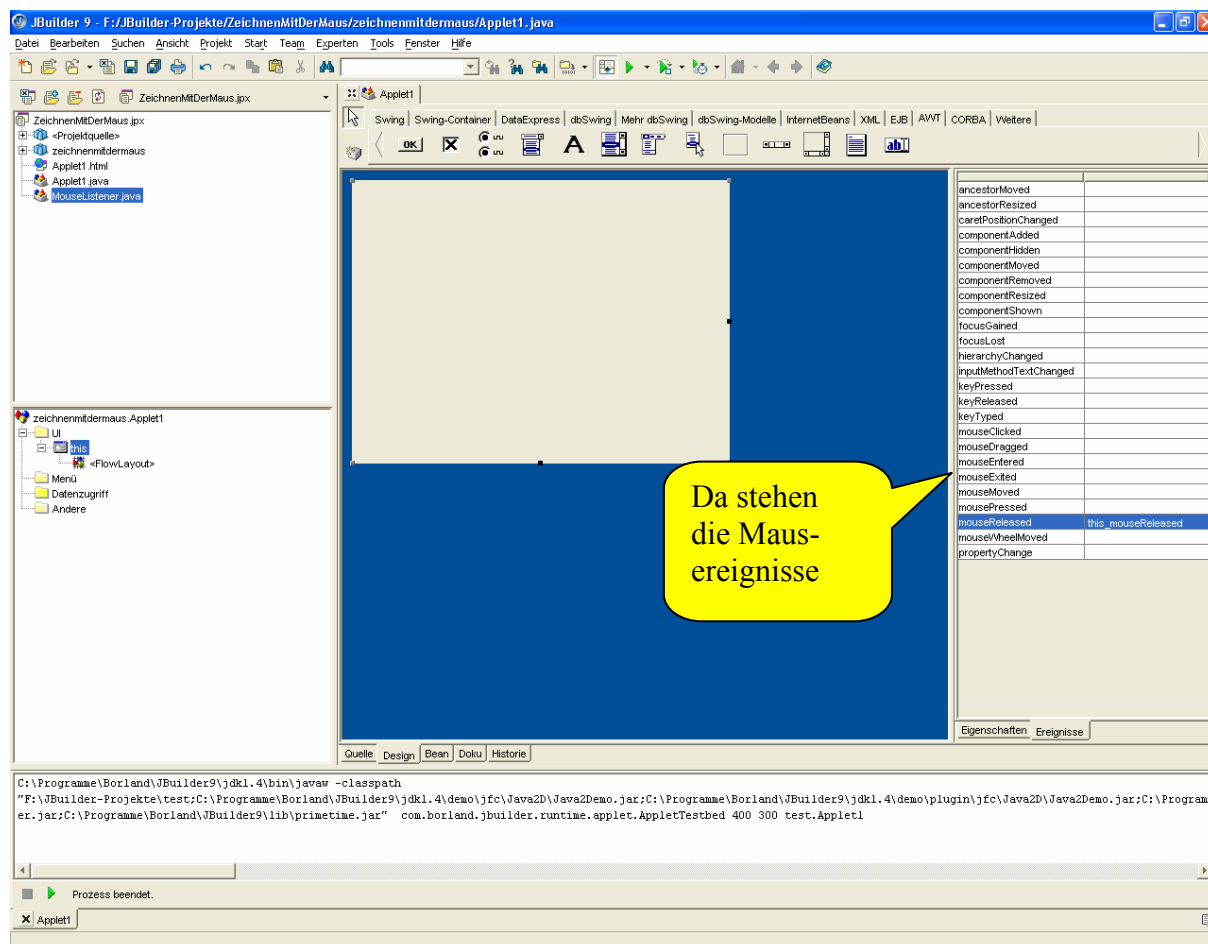
### Literaturhinweise:

- Küchlin/Weber: Einführung in die Informatik, Objektorientiert mit Java, Springer 1998
- Krüger, Guido: Handbuch der Java-Programmierung, <http://www.javabuch.de> oder Addison Wesley 2002

## 1. Arbeiten mit der Maus

**Mausereignisse** werden durch Benutzeraktionen ausgelöst, die sich auf die Computermaus beziehen. Leider werden in den unterschiedlichen Java-Versionen unterschiedliche Modelle zur Behandlung von Mausereignissen benutzt. U.a. führt das zu der unschönen Situation, dass es verschiedene Methoden zur Behandlung desselben Ereignisses gibt, die – wenn man versehentlich eine „falsche“ Bezeichnung wählt - anstandslos übersetzt werden, ohne zum gewünschten Ergebnis zu führen. (Bsp: es gibt die Methoden *mouseDrag()* und *mouseDragged()*) Wir werden hier das vom JBuilder unterstützte Modell benutzen, weil es so schön einfach ist.

Ereignisverarbeitung kann im JBuilder im *Design-Modus* automatisch eingefügt werden. Wir müssen dazu die Ereignis-Registerkarte rechts im Bild anklicken.



Bei der Maus kann man die Maustaste drücken und wieder loslassen oder die Maus bewegen. Klicken wir in das weiße Feld neben den genannten Ereignissen, dann werden entsprechende Ereignisbehandlungsmethoden in den Applet-Quelltext eingefügt. Da der JBuilder einen Mouse-Adapterklasse einfügt und zur Nutzung bereitstellt, ist der Umfang des Code erheblich. Wichtig für uns sind nur die **rot** hervorgehobenen Zeilen.

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

```
public class Applet1 extends Applet
{
    public Applet1() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception {
        this.addMouseListener(new Applet1_this_mouseMotionAdapter(this));
        this.addMouseListener(new Applet1_this_mouseAdapter(this));
    }

    void this_mousePressed(MouseEvent e)
    {
    }

    void this_mouseDragged(MouseEvent e)
    {
    }
}
```

```
class Applet1_this_mouseAdapter extends java.awt.event.MouseAdapter {
    Applet1 adaptee;

    Applet1_this_mouseAdapter(Applet1 adaptee) {
        this.adaptee = adaptee;
    }
    public void mousePressed(MouseEvent e) {
        adaptee.this_mousePressed(e);
    }
}
```

```
class Applet1_this_mouseMotionAdapter extends
java.awt.event.MouseMotionAdapter {
    Applet1 adaptee;

    Applet1_this_mouseMotionAdapter(Applet1 adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseDragged(MouseEvent e) {
        adaptee.this_mouseDragged(e);
    }
}
```

In diese fügen wir den erforderlichen Quellcode ein.

Wir wollen die Maus zum Zeichnen benutzen. Dazu merken wir uns jeweils die Koordinaten des Punktes, an dem die Maustaste gedrückt wurde und benutzen diese als ersten Anfangspunkt einer Linie, die gezeichnet wird, wenn die Maus bei gedrückter Maustaste bewegt wird. Nachdem eine Linie gezeichnet wurde, merken wir uns deren Endpunkt als neuen Anfangspunkt.

Die Koordinaten des Punktes, an dem die Maus gedrückt wurde, erhalten wir vom Mausereignis *MouseEvent* *e*. Die x-Koordinate mit

*e.getX()*, die y-Koordinate mit *e.getY()*. Diese beiden Werte speichern wir in den **globalen Variablen** *xanf* und *yanf*, die im gesamten Applet zur Verfügung stehen. Wir vereinbaren diese deshalb ganz am Anfang des Applets (s.u.) als ganze Zahlen.

Beim Drücken der Maus, also bei der Behandlung des *mousePressed*-Ereignisses, erhalten diese ihre Werte.

```
void this_mousePressed(MouseEvent e)
{
    xanf=e.getX();
    yanf=e.getY();
}
```

Wird die Maus bei gedrückter Maustaste bewegt, dann wird das *mouseDragged*-Ereignis ausgelöst. In diesem Fall zeichnen wir eine Linie vom alten Punkt zum neuen und merken uns den neuen. Zuvor holen wir uns mit *getGraphics()* den aktuellen Grafikkontext.

```
void this_mouseDragged(MouseEvent e)
{
    Graphics g = getGraphics();
    g.drawLine(xanf, yanf, e.getX(), e.getY());
    xanf = e.getX();
    yanf = e.getY();
}
```

Zuletzt schalten wir das Auffrischen der Grafik ab, bei der sonst jedes Mal die Zeichenfläche automatisch gelöscht wird.

```
public void update(Graphics g)
{
    paint(g);
}
```



Damit erhalten wir das vollständige Programm:

```
package zeichnenmitdermaus;
```

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

```
public class Applet1 extends Applet  
{
```

```
    int xanf,yanf;
```

Koordinaten des jeweils ersten  
Punktes

```
    public Applet1() {  
        try {  
            jbInit();  
        }  
        catch(Exception e) {  
            e.printStackTrace();  
        }  
    }
```

```
    private void jbInit() throws Exception {  
        this.addMouseListener(new Applet1_this_mouseMotionAdapter(this));  
        this.addMouseListener(new Applet1_this_mouseAdapter(this));  
    }
```

```
    public void update(Graphics g)  
    {  
        paint(g);  
    }
```

Löschen des Grafikbereichs verhindern.

```
    void this_mousePressed(MouseEvent e)  
    {  
        xanf=e.getX();  
        yanf=e.getY();  
    }
```

Beim jeweils ersten Tastendruck  
Koordinaten merken

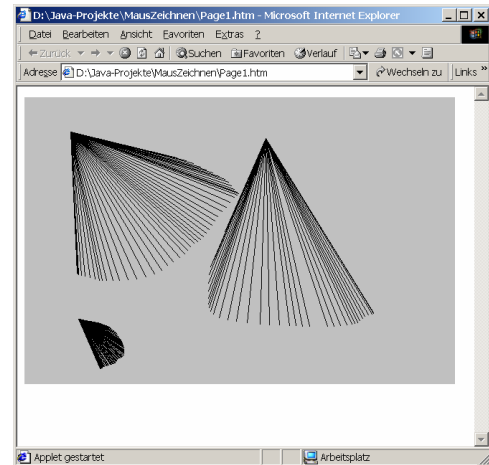
```
    void this_mouseDragged(MouseEvent e)  
    {  
        Graphics g = getGraphics();  
        g.drawLine(xanf, yanf, e.getX(), e.getY());  
        xanf = e.getX();  
        yanf = e.getY();  
    }  
}
```

Bei Mausbewegungen Linie zeichnen  
und neuen „Anfangspunkt“ merken

```
class Applet1_this_mouseAdapter extends java.awt.event.MouseAdapter {  
    //... wie oben
```

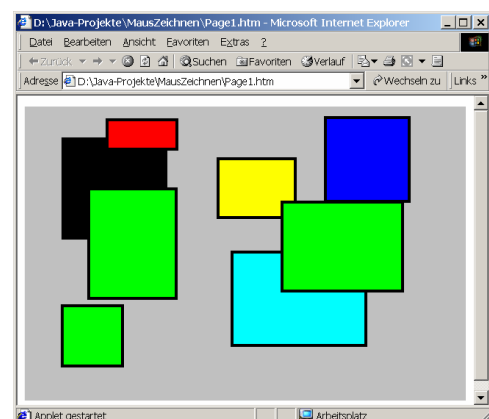
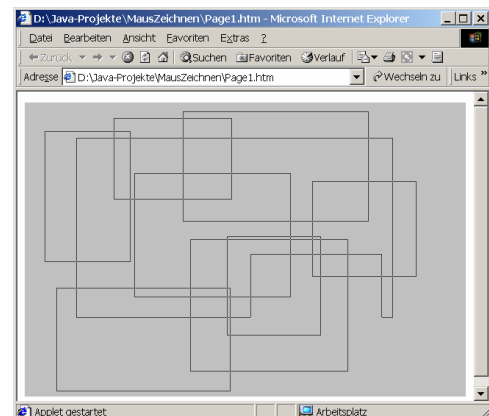
## 2. Aufgaben

1. a: Verändern Sie die Anfangspunktkoordinaten beim Zeichnen nicht. Erzeugen Sie dann so etwas wie die abgebildeten „Fächer“.
  - b: Lassen Sie die Linienfarben beim Zeichnen zufällig verändern.
  - c: Lassen Sie die Linienfarben zufällig, aber „schön“ verändern, so dass sich „geschmackvolle“ Fächer ergeben.
2. Sparen Sie eine „Leiste“ am oberen Bildrand vom Zeichnen aus. Färben Sie die Leiste ein.



3. a: Führen Sie farbige Flächen („Buttons“) auf der oberen Leiste ein, die aus Mausklicks reagieren, z.B. ihre Farbe ändern oder sichtbar „gedrückt“ werden.
- b: Beschriften Sie die Buttons. Verändern Sie bei Mausklicks auf die Buttons die Zeichenfarbe.

4. a: Lassen Sie mit Hilfe der Maus Rechtecke am Bildschirm zwischen Anfangspunkt und aktuellem Mauspunkt zeichnen.
- b: Zeichnen Sie die Rechtecke im invertierenden Modus. Löschen Sie die gerade gezeichneten Rechtecke nicht zu schnell, sonst sehen Sie gar nichts am Bildschirm, weil die Grafikkarte nicht schnell genug ist!
- c: Füllen Sie die so gezeichneten Rechtecke farbiger aus, nachdem die Maustaste losgelassen wurde.
- d: Lassen Sie die Zeichenfarbe mit Hilfe der Buttons wählen.
- e: Schreiben Sie jetzt das Programm „Mondrian“ so um, dass „mondrianische“ Bilder mit der Maus gezeichnet werden können.



5. Benutzen Sie Buttons, um unterschiedliche Formen am Bildschirm erzeugen zu können. Man kann z.B. Kreise „aufziehen“, Polygone zeichnen, Dreiecke „setzen“, ...
6. Überlegen Sie sich die auftretenden Probleme, wenn man gezeichnete Rechtecke wieder löschen möchte. Ähnliche Aufgaben erwarten Sie, wenn Sie die Rechtecke nachträglich