

Einführung in die Informatik - Teil 6b -

Java-Anwendungen mit dem JBuilder

Inhalt:

1. Anwendungen versus Applets
2. Die Arbeit mit IDEs
3. J++-Anwendungen entwickeln
4. Erzeugen einer neuen leeren Anwendung
5. Der Quelltext
6. Das Einfügen von Steuerelementen
7. Zeichnen
5. 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. Anwendungen versus Applets

Anwendungen sind Java-Programme, die durch den Java-Interpreter unter einem Betriebssystem ablaufen. Sie werden also nicht direkt ausgeführt, sondern „interpretiert“. Der Zweck der Übung ist es, Java-Programme maschinen- und **betriebsystemunabhängig** zu halten, also dafür zu sorgen, dass jedes Java-Programm auf jedem Rechner lauffähig ist, wenn nur ein Java-Interpreter existiert. Solche Programme laufen meist als **Kommandozeilenversionen**, weil es gar nicht so einfach ist, sie innerhalb eines Fensters der grafischen Oberfläche, also z. B. unter Windows oder KDE, laufen zu lassen. Der Einführungsunterricht mit Anwendungen unter Java arbeitet deshalb meist mit einer DOS-Box, ähnelt also sehr der Arbeitsweise vor 30 Jahren und verfügt so über weit weniger Komfort als etwa ein Uralt-Pascal-System. Applets haben dieses Problem nicht, weil sie innerhalb der HTML-Seite in einem Browser laufen, der fast immer ein eigenes Fenster hat.

Einen Ausweg bieten integrierte grafische Entwicklungsumgebungen (IDEs) wie z. B. ©Forte für Java von SUN, das es für verschiedene Betriebssysteme gibt, ©J++ von Microsoft © oder eben der JBuilder von Borland. Diese leiten eine Anwendung von einer **Form** (einem Formular) ab, das schon über alle notwendigen Eigenschaften verfügt, um als eigenständiges Programm in einem Fenster zu laufen. Neue Anwendungen erben diese Eigenschaften. Die Programmierer ergänzen dann nur noch den Code, der spezifisch für das neue Programm ist. Forte und JBuilder benutzen zu diesem Zweck z. B. die Formulare des **AWT** (Abstract Windowing Toolkit) oder der **Swing**-Bibliothek. Damit bleiben die erstellten Programme maschinenunabhängig. J++ benutzt statt dessen die **Windows Foundation Classes MFC**, die es nur für Microsoft Windows gibt, erzeugt also Windows-spezifischen Code, der in eine ausführbare Datei gepackt wird – ein normales EXE-File. Forte und JBuilder erfordern leider weit leistungsfähigere Rechner als J++, so dass die Arbeit auf älteren Schulrechnern etwas unerfreulich wird. (Vielleicht ändert sich das ja bald.)

Als Konsequenz aus dieser Situation benutzen wir für die Arbeit mit Applets reines Java, das unter allen Browsern und Betriebssystemen lauffähig sein sollte. Die Schülerinnen und Schüler können so ihre Arbeitsergebnisse bei Bedarf im Internet veröffentlichen. Für Anwendungen aber nutzen wir unter J++ die Windows-Erweiterungen, um auch auf weniger leistungsfähigen Schulrechnern effizienten Code zu erzeugen, unter JBuilder wieder reines Java. Die Unterschiede beim Programmieren sind minimal, weil der Code weitgehend von der IDE automatisch erzeugt wird. Die eigene Programmierarbeit verbleibt damit in jedem Fall im reinen Java. Und auch die Arbeit mit den Systemen ähnelt sich so stark, dass ein Wechsel zu einem anderen System unproblematisch bleibt.

Die Entwicklung von Anwendungen entspricht der Arbeit mit Delphi-, VisualBasic oder C++-Systemen, die ebenfalls ohne den Umweg über Interpreter ausführbare Dateien erzeugen.

2. Die Arbeit mit IDEs

IDEs sind Werkzeuge zur Entwicklung von Programmen. Die Grundidee der Systeme besteht daraus, den „Programmierern“ fertige Programmstrukturen und darin lauffähige Objektklassen für die Standardaufgaben eines Programms bereitzustellen. Die Programmierer erzeugen – auch hier von der IDE unterstützt – Instanzen der vorgegebenen Objektklassen, deren Datenfelder sie mit konkreten Werten füllen. Das geschieht weitgehend interaktiv am Bildschirm, indem die Instanzen entweder mit der Maus bearbeitet werden, um z. B. die Größe und Position zu bestimmen, oder durch das Setzen bestimmter Eigenschaften (*Properties*) wie Farbe oder Schriftart in einem *Eigenschaften*-Fenster.

Objekte agieren (meist) nicht selbst, sondern reagieren auf Ereignisse (*Events*), die z. B. durch einen Mausklick oder einen Tastendruck auf der Tastatur ausgelöst werden. Das Betriebssystem empfängt solche Ereignisse und reicht sie an die Elemente auf dem Bildschirm weiter. Ein typisches *Mausereignis* ist ein Doppelklick auf das Kreuz-Symbol in der oberen rechten Ecke eines Fensters, das z. B. eine Meldung „<Doppelklick an der Position (400,120)>“ erzeugt, die dann der Reihe nach an die Fenster des Desktops durchgereicht wird. Stellt das angeklickte Fenster fest, dass sein Kreuzchen getroffen wurde, dann reagiert es darauf, indem es sich selbst schließt.

Die Objekte müssen somit

- dem Systems bekannt gemacht werden, um die Botschaften des Systems zu empfangen, und
- über Methoden verfügen, um auf die Standardereignisse des Systems zu reagieren.

Beides wird erreicht, indem alle Anwendungen von einer Mutterklasse *Form* abgeleitet werden, die über solche Eigenschaften verfügt und sie an die Tochterklassen vererbt. Die Reaktion auf die Standardereignisse beruht typischerweise darin, nichts zu tun.

Weil die Elemente z. B. eines Windows-Programms ebenso wie die Standardereignisse festgelegt sind, kann das System fertige Programmschablonen erstellen, die die vom Programmierer am Entwicklungsbildschirm zusammengestellte Oberfläche aus Fenstern, Beschriftungen, Knöpfen, Ein- und Ausgabefeldern, ... erzeugen, ohne dass eine einzige Zeile Quelltext selbst geschrieben werden muss. Damit ist aber nur festgelegt, wie die Oberfläche aussieht, aber nicht, was sie tut. Bis auf wenige Standardreaktionen (z. B. Fenster verkleinern oder schließen) reagiert das Programm gar nicht!

Das „Programmieren“ unter IDEs besteht deshalb weitgehend daraus, die anfangs leeren Ereignisbehandlungsmethoden (*Event-Handler*) auszufüllen, indem Quelltext eingegeben wird, der beschreibt, wie das gerade bearbeitete Objekt auf einzelne Ereignisse zu reagieren hat:

- Was passiert, wenn ein bestimmter Knopf angeklickt wird? (*Das Programm wird beendet.*)
- Was passiert, wenn ein anderer Knopf angeklickt wird? (*Eine Datei wird gespeichert.*)
- Was passiert, wenn eine Taste gedrückt wird? (*Je nach gedrückter Taste wird anders reagiert.*)
- ...

3. JBuilder-Anwendungen entwickeln

Das Schreiben von Javaprogrammen geschieht im Allgemeinen in drei Phasen:

1. Zusammenstellung der Programmoberfläche in der Entwicklungsumgebung.

Dazu werden Instanzen der Klassen erzeugt, die über dem Designerfenster am oberen Bildschirmrand angeboten werden, z. B. Fenster, Knöpfe, Textfelder, ... Zuerst sollte dazu die gewünschte Bibliothek ausgewählt werden (meist AWT). Die Eigenschaften dieser Objekte werden mit Hilfe des Eigenschaftsfensters geeignet gesetzt. Der JBuilder erzeugt automatisch den dazu gehörende Java-Code.

2. Ausfüllen der benötigten Ereignisbehandlungsmethoden mit Quelltext.

Nach Auswahl des gewünschten Ereignisses im Eigenschaftsfenster (z. B. des „mouse-Clicked-Ereignisses“) erzeugt der JBuilder ggf. eine passende Adapterklasse sowie eine leere Java-Methode und springt an die richtige Stelle im Programm. Der „wohlüberlegte“ Text wird hier eingegeben.

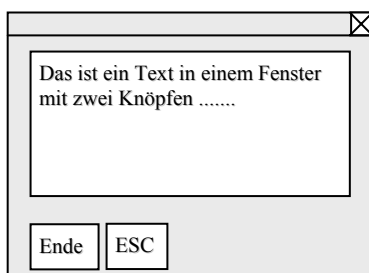
3. Übersetzen und Testen des erzeugten Programms

Der JBuilder übersetzt das Programm in ausführbaren Code und erzeugt eine ohne weitere Zusätze ausführbare Datei, die sofort gestartet wird. Das so erzeugte Programm wird normalerweise fehlerhaft sein, zumindest aber unvollständig. Deshalb sucht man die Fehler und ergänzt ggf. die Oberfläche bzw. ändert den Programmcode.

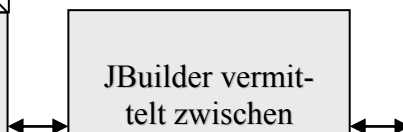
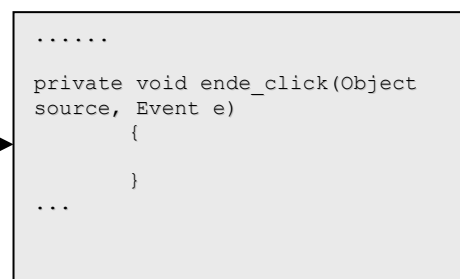
Insgesamt wird mit einem JBuilder-System auf zwei verschiedenen Ebenen gearbeitet:

- In der visuellen Entwicklungsumgebung werden Objekte erzeugt und mit Eigenschaften ausgestattet, die die spätere Programmoberfläche bilden.
- Parallel dazu wird vom JBuilder automatisch Programm-Quelltext erzeugt, der zur Laufzeit genau die in der Entwicklungsumgebung definierte Oberfläche nachbildet. Dieser wird ggf. per Hand verändert oder ergänzt.

Visuelle Entwicklungsumgebung



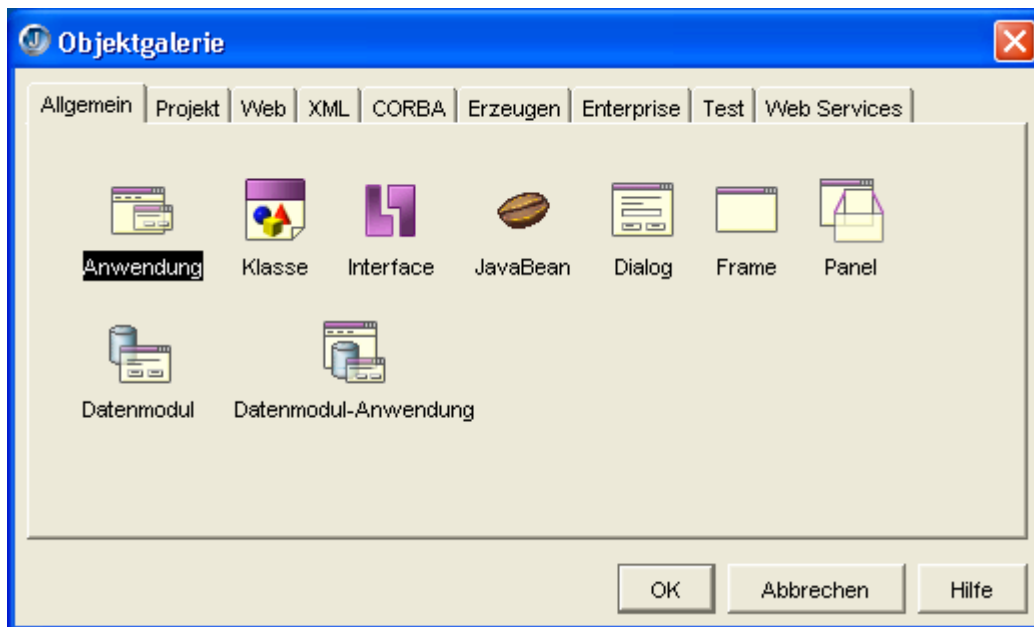
Java-Quelltext



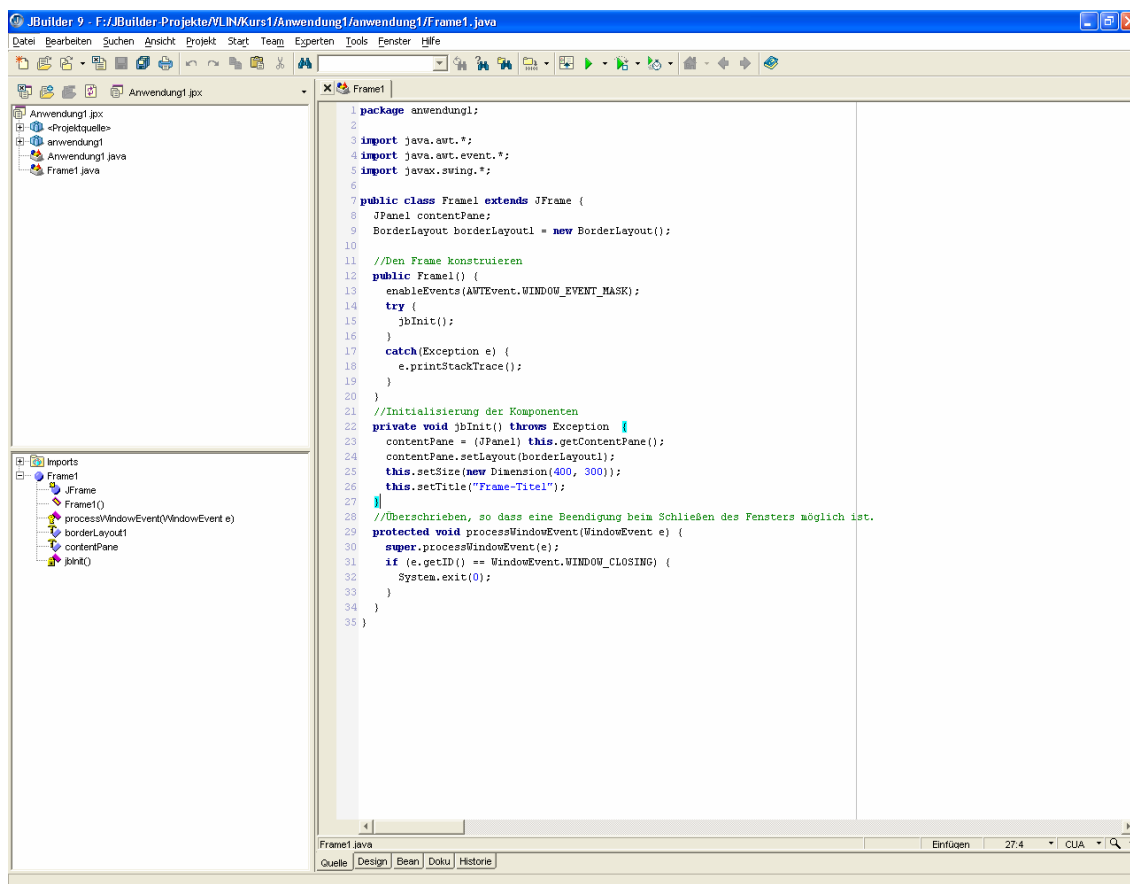
4. Erzeugen einer neuen leeren Anwendung

Mit dem JBuilder erzeugen wir Anwendungen ähnlich wie Applets. Dazu starten wir den JBuilder, wählen den Menüpunkt **Datei → Neues Projekt**, benennen das Projekt und wählen das gewünschte Verzeichnis. In diesem erzeugen wir nun eine neue Anwendung über

Datei → Neu...



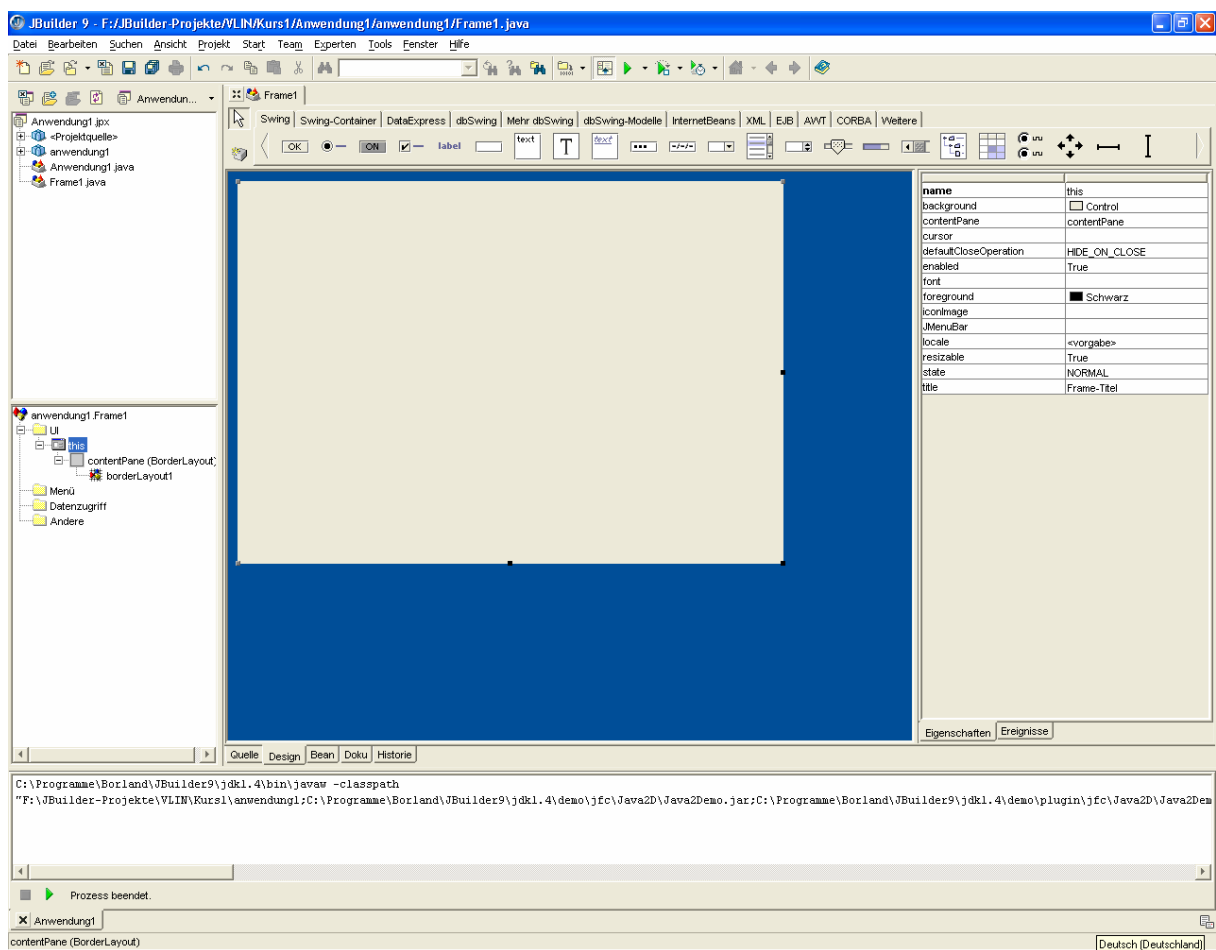
Nach der Beantwortung der auch bei Applets üblichen Fragen erhält man den Quelltext einer leeren Anwendung, die nur ein Fenster bereitstellt.



Dieses lassen wir mal laufen:



Nach dem Umschalten in den Designer-Modus der IDE erhalten wir ein Bild, das sich kaum von der Applet-Entwicklungsumgebung unterscheidet.



5. Der Quelltext

Der Quelltext unserer Java-Anwendung enthält Java-Code und zusätzlich eine Reihe von Anmerkungen und Erklärungen. Diese können wir teilweise löschen. Die automatisch erzeugten Anweisungen zur Erzeugung des Formulars mit seinen Komponenten müssen aber bleiben.

```
package anwendung1;

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

public class Frame1 extends JFrame {
    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();

    //Den Frame konstruieren
    public Frame1() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    //Initialisierung der Komponenten
    private void jbInit() throws Exception {
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(borderLayout1);
        this.setSize(new Dimension(578, 405));
        this.setTitle("Frame-Titel");
    }

    //Überschrieben, so dass eine Beendigung beim Schließen des Fensters
    //möglich ist.
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
        }
    }
}
```

benötigte Pakete

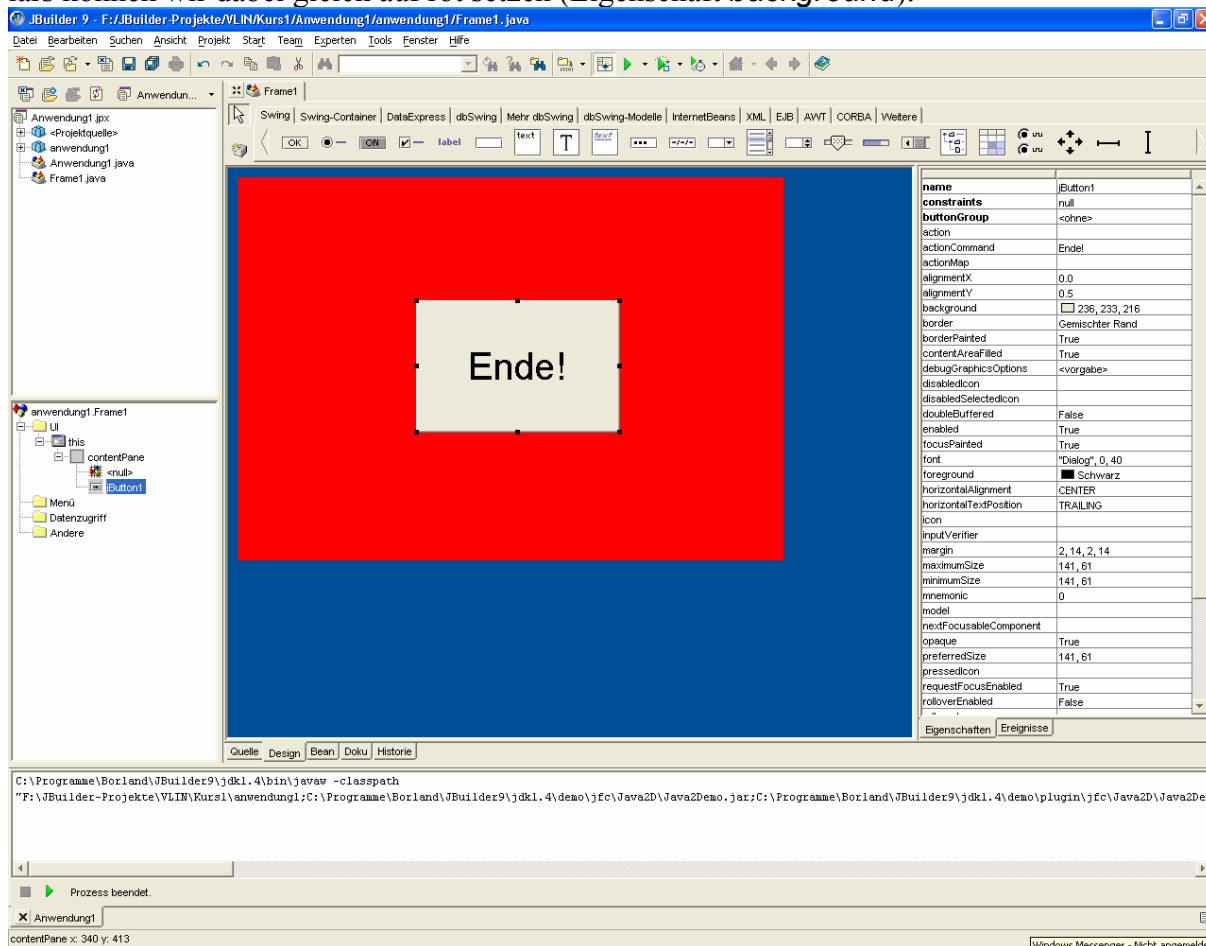
Die Klasse *Frame1* von *JFrame* ableiten

Hier werden die Komponenten des Fensters eingerichtet

Bei Programmläufen wird das Formular erzeugt und die Kontrolle an dieses übergeben. Im Normalfall wird danach auf Benutzeraktionen gewartet, die entsprechende Reaktionen auslösen.

6. Das Einfügen von Steuerelementen

Wir wollen nur einen einzelnen Button einfügen, mit dessen Hilfe wir das Programm wieder beenden. Dazu setzen wir die *Layout*-Eigenschaft des Fensters auf *null*, wählen dann z. B. im *SWING*-Werkzeugfenster den Button aus und klicken auf unser Formular. Dort erscheint ein Knopf mit dem Namen *jbutton1*. Im Eigenschaftsfenster ändern wir seine Aufschrift in *Ende!* und wählen eine größere Schrift aus – über die Eigenschaft *font*. Die Farbe des Formulars können wir dabei gleich auf rot setzen (Eigenschaft *background*).



Doppelklicken wir jetzt auf den Knopf, dann wird eine leere Ereignisbehandlungsmethode erzeugt, die auf das Klicken reagiert. In diese fügen wir den Code für das Beenden von Programmen ein: *System.exit(0)*.

```
void jButton1_actionPerformed(ActionEvent e) {  
    System.exit(0);  
}
```

Wollen wir den Knopf auf andere Ereignisse reagieren lassen, dann müssen wir diese im Eigenschaftsfenster aufwählen und auf das leere Feld rechts daneben doppelklicken.

7. Zeichnen

Alle Komponenten (abgeleitet von *Component*) verfügen über eine *Graphics*-Komponente. Fenster-Formulare also auch. Diese müssen wir uns allerdings erstmal holen:

Graphics g = getGraphics();

Wollen wir auf Knopfdruck eine Linie zeichnen, dann geht das so:

```
void button1_actionPerformed(ActionEvent e)
{
    Graphics g = getGraphics();
    g.drawLine(0,0,200,200);
}
```

