

## 1. Anwendungen versus Applets

Anwendungen sind Java-Programme, die eigentlich durch den Java-Interpreter unter einem Betriebssystem ablaufen sollten. 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 dafür 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, ausführen 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, ©JBuilder von Borland oder eben ©J++ von Microsoft. 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 maschinen-unabhängig, müssen aber interpretiert werden. 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, das sehr schnell ablaufen kann. Forte und JBuilder erfordern also für die selbe Problemlösung meist 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.
- nutzen wir für Anwendungen aber die Windows-Erweiterungen von J++, um auch auf weniger leistungsfähigen Schulrechnern effizienten Code zu erzeugen. Die Unterschiede beim Programmieren können minimal gehalten werden, weil der MFC-spezifische Code weitgehend von der IDE automatisch erzeugt wird. Die eigene Programmierarbeit verbleibt damit im reinen Java. Und auch die Arbeit mit den Systemen ähnelt sich so stark, dass ein Wechsel zu einem anderen System unproblematisch bleibt.

Die MFC-Klassen haben zwar teilweise den gleichen Namen wie ihre AWT-Pendants, verfügen aber über eine andere Funktionalität. Besonders drastisch macht sich das bei der Grafik bemerkbar, die von einem ganz anderen Modell (mit **Pen** und **Brush**) ausgeht wie das AWT. Wir werden daher meist nur diejenigen Methoden verwenden, die in beiden Systemen zumindest ähnlich arbeiten.

Die Entwicklung von J++-Anwendungen entspricht der Arbeit mit Delphi-, VisualBasic oder C++-Systemen, die ebenfalls ohne den Umweg über Interpreter ausführbare Dateien erzeugen. (Hinweis: J++ kann auch ohne Windowsspezifische Erweiterungen Anwendungen erzeugen. Die IDE ist dann aber nur sehr eingeschränkt nutzbar!)

## 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 Elemente 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 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, ... erzeugt, 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 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. J++-Anwendungen entwickeln

Das Schreiben von J++-Javaprogrammen geschieht im allgemeinen in drei Phasen:

#### 1. Zusammenstellung der Programmoberfläche in der Entwicklungsumgebung.

Dazu werden Instanzen der Klassen erzeugt, die in dem *Werkzeugfenster* am linken Bildschirmrand angeboten werden, z. B. Fenster, Knöpfe, Textfelder, ... Die Eigenschaften dieser Objekte werden mit Hilfe des *Eigenschaftsfensters* geeignet gesetzt. J++ erzeugt automatisch den dazu gehörenden Java-Code.

#### 2. Ausfüllen der benötigten Ereignisbehandlungsmethoden mit Quelltext.

Nach Auswahl des gewünschten Ereignisses im Eigenschaftsfenster (z. B. des „Click-Ereignisses“) erzeugt J++ eine leere Java-Methode und springt an die richtige Stelle im Programm. Der „wohlüberlegte“ Programmcode wird hier eingegeben.

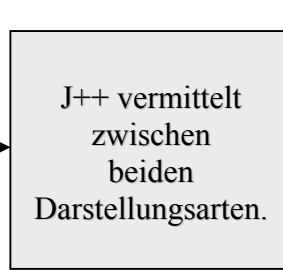
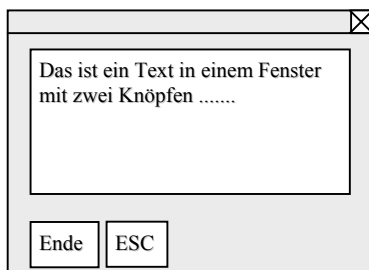
#### 3. Übersetzen und Testen des erzeugten Programms

J++ 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.

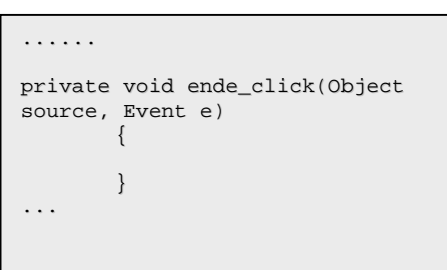
Insgesamt wird mit einem J++-System auf zwei verschiedenen Ebenen gearbeitet:

- In der visuellen Entwicklungsumgebung (dem *Designer*) werden Objekte erzeugt und mit Eigenschaften ausgestattet, die die spätere Programmoberfläche bilden. Man erreicht diese Ebene über den Menüpunkt **Ansicht → Designer** oder durch Klicken mit der rechten Maustaste auf die Formulardatei im *Projektextplorer* und Auswahl des *Ansichtsdesigners*.
- Parallel dazu wird von J++ automatisch Programm-Quelltext erzeugt, der zur Laufzeit genau die in der Entwicklungsumgebung definierte Oberfläche nachbildet. Dieser wird ggf. per Hand verändert oder ergänzt. Man erreicht diese Ebene über den Menüpunkt **Ansicht → Code** oder durch Klicken mit der rechten Maustaste auf die Formulardatei im *Projektextplorer* und Auswahl des *Code anzeigen*.

#### Visuelle Entwicklungsumgebung

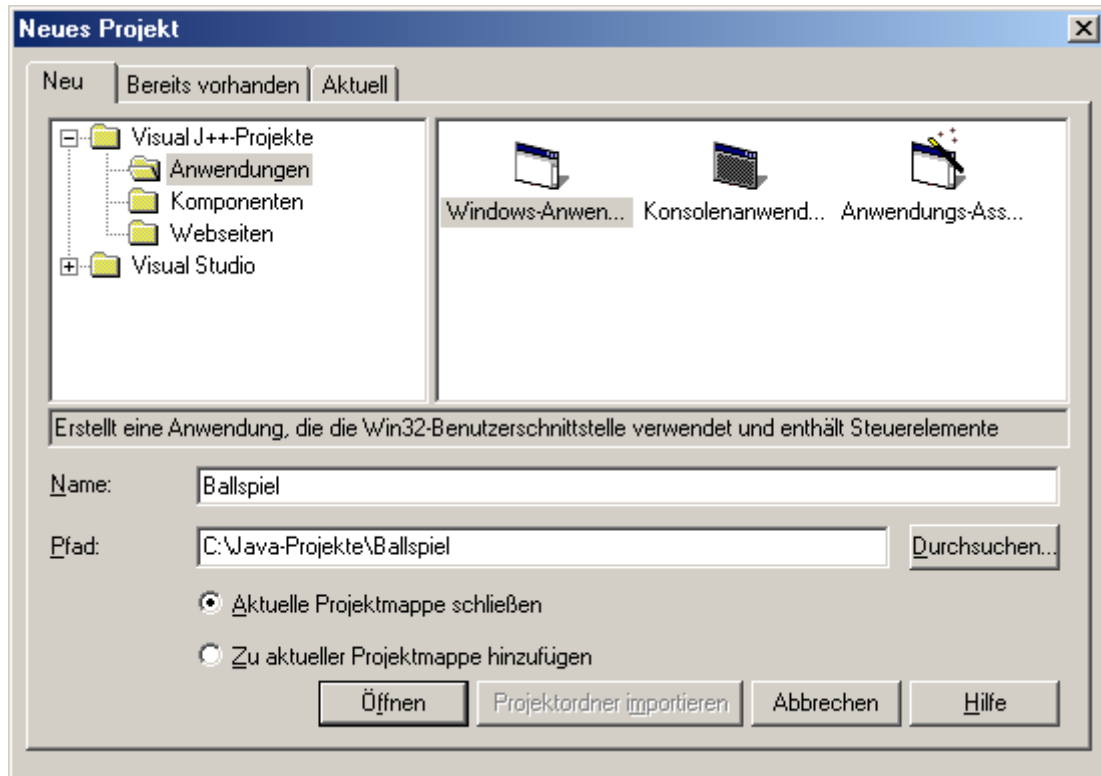


#### Java-Quelltext



## 4. Erzeugen einer neuen leeren Anwendung

Mit J++ erzeugen wir Anwendungen ähnlich wie Applets. Dazu starten wir J++, wählen den Menüpunkt **Datei** → **Neues Projekt** und erhalten das folgende Fenster.

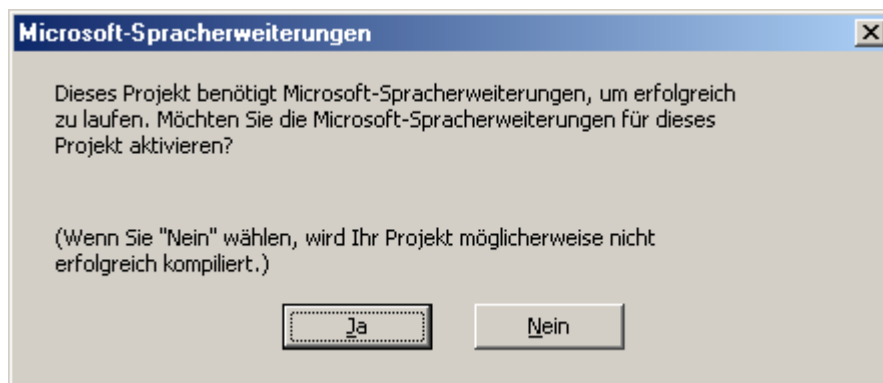


Darin

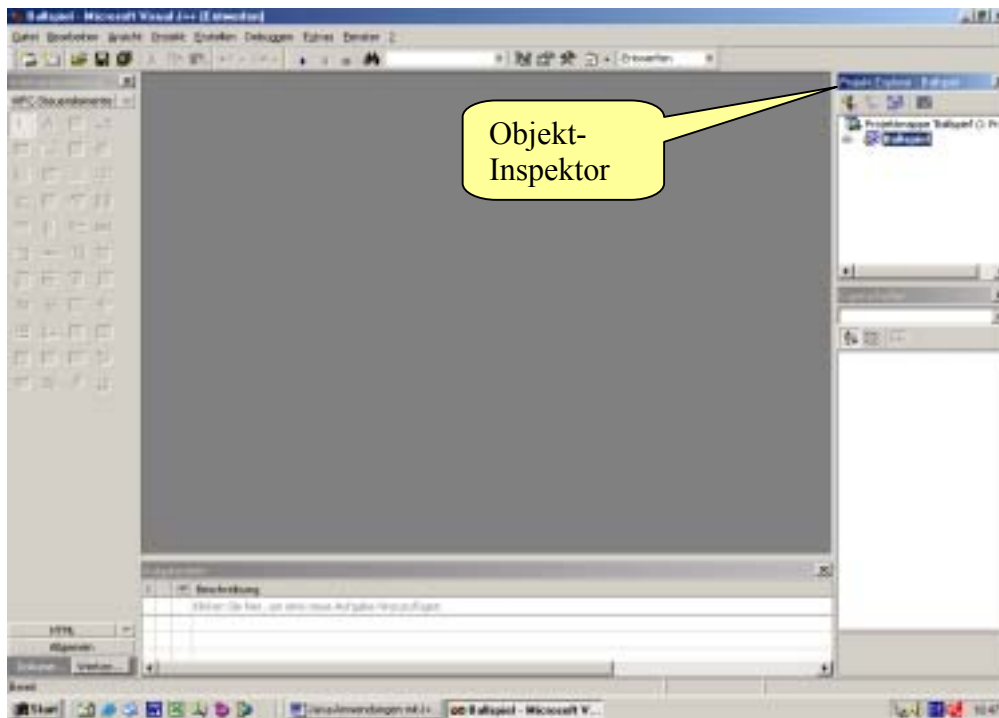
- wählen wir *Windows-Anwendung*
- vergeben einen aussagekräftigen Namen, z.B. *Ballspiel*
- und wählen als Pfad ein geeignetes Verzeichnis, z. B. *c:\Java-Projekte\Ballspiel*

Danach klicken wir den Button *Öffnen* an.

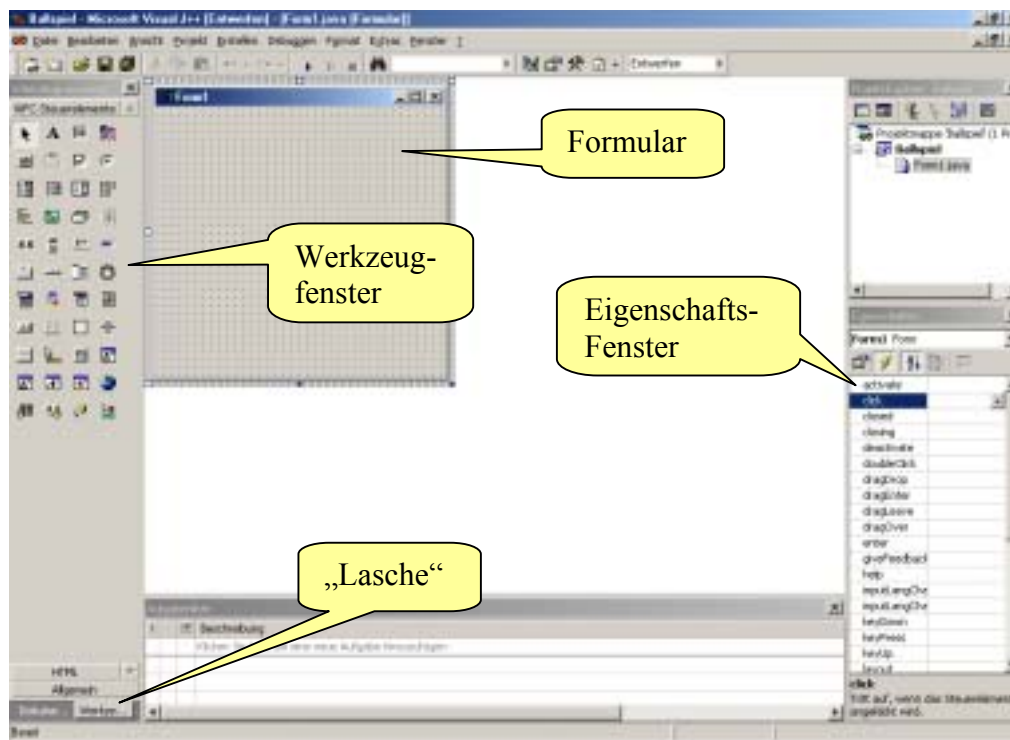
Wir erhalten ein Meldungsfenster, das uns über die Abweichung von der „reinen Lehre“ informiert.



Nach einem weiteren Informationsfenster mit einigen juristischen Hinweisen erhalten wir einen weitgehend leeren Bildschirm.



Wir erweitern im *Objekt-Explorer* am rechten Bildrand die Projektmappe durch Anklicken des +-Symbols und wählen `Form1.java` aus. Jetzt endlich steht uns ein leeres Formular und das *Werkzeugfenster* zur Verfügung. (Ggf. müssen wir das Werkzeugfenster über die „Lasche Werkzeuge“ ganz unten-links am Bildschirm „hervorholen“).



In diesem Zustand befindet sich J++ in der Designer-Ansicht, in der Steuerelemente des Werkzeugfensters erzeugt und „designed“ werden können. Bei Bedarf können wir auch über den Menüpunkt **Ansicht → Code** den Quelltext ansehen und bearbeiten (s.o.).

Das Projekt enthält jetzt eine lauffähige Anwendung, die wir direkt übersetzen und ausführen können. Wir klicken dazu den Pfeil (▶) in der oberen Befehlsleiste an. Nach kurzer Zeit erscheint das leere Anwendungsfenster und in unserem Verzeichnis befindet sich ein normales ausführbares Windows-Programm.

## 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 erzeugte Anweisung zur Erzeugung des Formulars mit seinen Komponenten müssen aber bleiben.

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
```

MFC-Klassen  
benutzen

```
/**
 * Diese Klasse kann eine variable Anzahl von Parameter auf der Befehls-
 * zeile entgegennehmen. Die Programmausführung startet mit der Methode
 * main(). Der Klassenkonstruktor wird nicht aufgerufen, bevor nicht ein
 * Objekt vom Typ 'Form1' in der Methode main() erstellt wird.
 */
```

```
public class Form1 extends Form
{
    public Form1()
    {
        // Erforderlich für die Unterstützung des
        // Visual J++-Formulardesigners
        initForm();
        // ZU ERLEDIGEN: Fügen Sie beliebigen Konstruktorcode hinter den
        // Aufruf von initForm hinzu
    }

    /**
     * Form1 überlädt dispose, damit es die Komponentenliste
     * bereinigen kann.
     */

    public void dispose()
    {
        super.dispose();
        components.dispose();
    }
}
```

Die Klasse *Form1* von *Form* ableiten

hier wird das Formular mit  
seinen Komponenten erzeugt

```
/**
 * HINWEIS: Der folgende Code ist für den Visual J++-
 * Formulardesigner erforderlich. Er kann mit dem Formulareditor
 * verändert werden. Ändern Sie ihn nicht mit dem Codeeditor.
 */

Container components = new Container();

private void initForm()
{
    this.setText("Form1");
    this.setAutoScaleBaseSize(new Point(5, 13));
    this.setClientSize(new Point(292, 273));
}
```

```
/**
 * Der Haupteinsprungpunkt für die Anwendung.
 *
 * Ein @param args-Array aus Parametern wird an die Anwendung
 * über die Befehlszeile übergeben.
 */

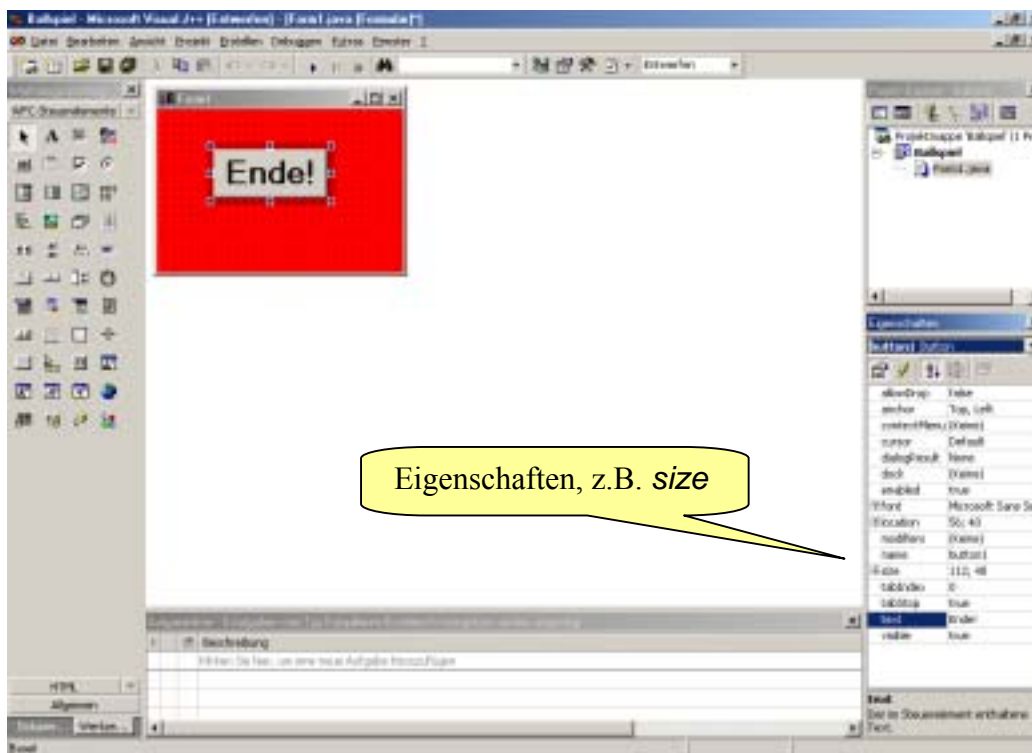
public static void main(String args[])
{
    Application.run(new Form1());
}
}
```

Methode *main* von Anwendungen

Anders als Applets müssen Anwendungen eine Methode *main* enthalten, die genau in der angegebenen Form definiert wird. Diese wird bei Programmläufen gestartet. In unserem Fall erzeugt sie das Formular und übergibt die Kontrolle an dieses. 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 wählen wir im Werkzeugfenster durch Anklicken den Button aus und klicken auf unser Formular. Dort erscheint ein Knopf mit dem Namen *button1*. Wir ändern durch „Ziehen“ an der unteren-rechten Ecke mehrmals seine Größe und beobachten die Änderungen im Eigenschaftsfenster (Eigenschaft *size*). Danach ändern wir die Werte dieser Eigenschaft im Eigenschaftsfenster und beobachten die Auswirkungen auf dem Formular. Danach platzieren wir den Button durch Verschieben in der Mitte des Fensters und beobachten die Änderungen der Eigenschaft *location*. Auch dort experimentieren wir mit der Direkteingabe von Werten. Im Eigenschaftsfenster ändern wir die Button-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 *backColor*).



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: *Application.exit()*.

```
private void button1_click(Object source, Event e)
{
    Application.exit();
}
```

Wollen wir den Knopf auf andere Ereignisse reagieren lassen, dann müssen wir diese im Eigenschaftsfenster aufwählen (über das Blitz-Symbol oben am Fensterrand).

## 7. Nutzung von Event-Handlern

Als Beispiel für programmierte Reaktionen auf andere als das Standardereignis wollen wir die Farbe des Fensters in Gelb ändern, wenn sich der Mauszeiger über dem Button befindet – ihn „betritt“ (Ereignis: *mouseEnter*). Beim „Verlassen“ (Ereignis: *mouseLeave*) des Knopfs wird die Fensterfarbe wieder auf Rot gesetzt.

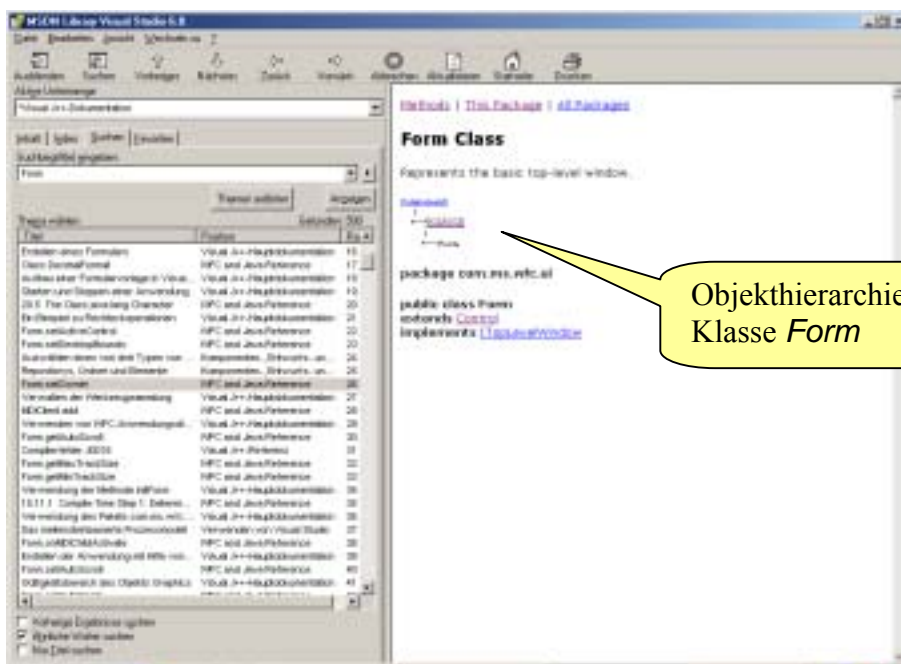
Die Farbe eines Fensters haben wir ja schon im Eigenschaftsfenster direkt gesetzt. Versuchen wir Ähnliches vom Programm her, z.B. durch die Anweisung

```
Form1.backColor = Color.YELLOW;
```

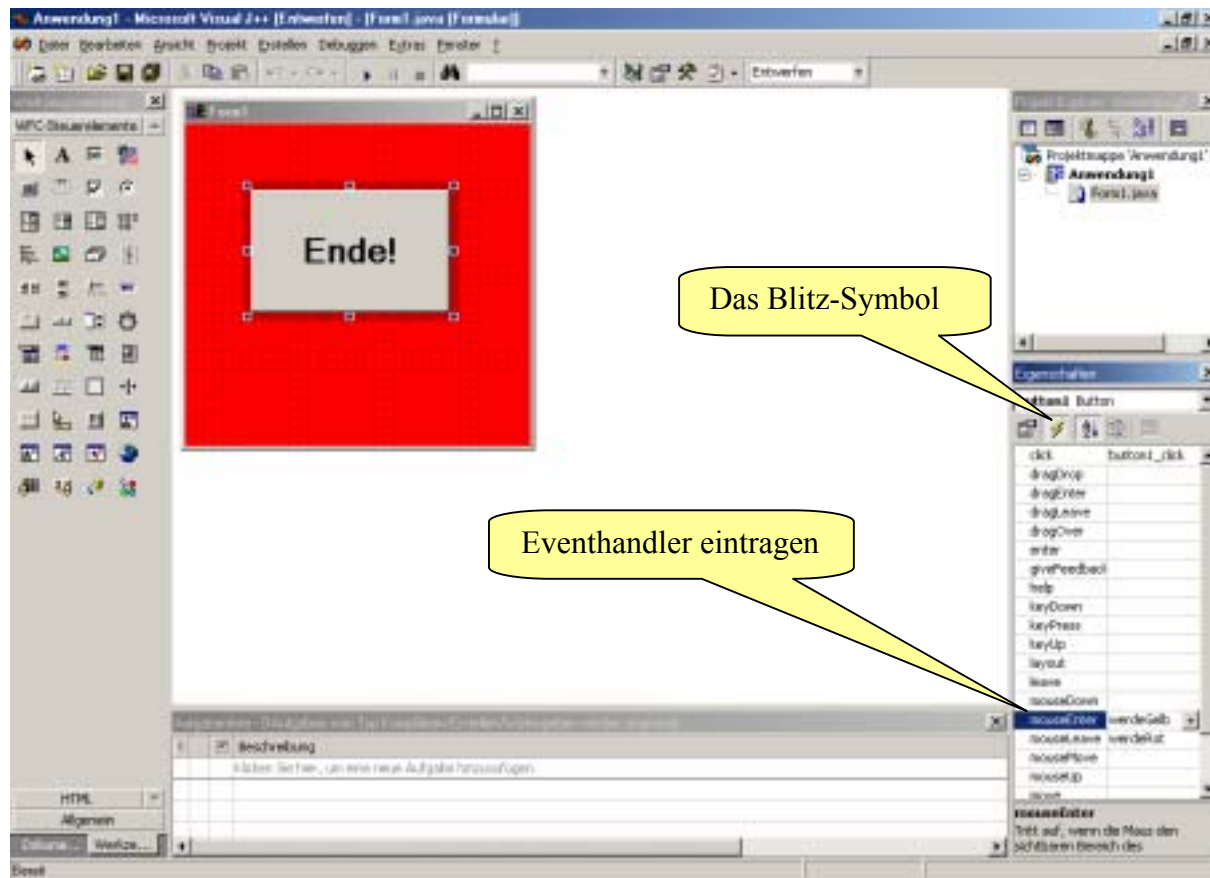
dann erhalten wir einen **FEHLER!** Wir können nämlich von außen nicht auf die „privaten“ Eigenschaften eines Objekts einwirken. Objekteigenschaften werden normalerweise über Methoden gelesen oder gesetzt, die mit *get...* oder *set...* beginnen, z.B. *getBackColor()* oder *setBackColor(...)*. Wo aber finden wir die?



In der J++-Hilfe finden wir bei der Klasse *Form* nichts Entsprechendes. In der Übersicht findet man aber die Objekthierarchie der Klasse *Form*, und da zeigt sich, dass diese von *Control* abgeleitet wurde. Unter den Methoden von *Control* finden wir alle benötigten Methoden, z. B. auch alles, was das Zeichnen, also die Grafik, betrifft.



Mit diesen Informationen können wir unsere Eventhandler schreiben. Dazu klicken wir im Designermodus den Knopf an und wählen über das Blitz-Symbol die Ereignisse aus. In die Felder von *mouseEnter* und *mouseLeave* tragen wir jeweils einen frei gewählten Namen ein – hier *werdeGelb* und *werdeRot* - und drücken die RETURN-Taste.



In den von J++ erzeugten Code tragen wir nur noch die richtigen Anweisungen ein.

```
private void werdeGelb(Object source, Event e)
{
    setBackgroundColor(Color.YELLOW);
}

private void werdeRot(Object source, Event e)
{
    setBackgroundColor(Color.RED);
}
```



Man beachte die Großschreibung bei den Farbnamen! (Gilt für die WFC-Graphics-Klasse.)

Der Parameter *source* gibt an, von welchem Objekt das Ereignis ausgelöst wurde, das die Methode aufgerufen hat. Wir können also durchaus dieselbe Methode unterschiedlichen Objekten zuordnen und erst nach dem Aufruf entscheiden, was zu tun ist. Dazu können wir anhand von *source* die benötigten Informationen ermitteln: Das Objekt enthält Klassennamen, den charakteristischen Feldnamen und dessen Feldwert des aufrufenden Objekts. Im oberen Bild wurde dies über *label1.setText(source.toString());* auf einer Labelkomponente dargestellt. Die Aufschrift des Buttons ist leicht zu ermitteln.