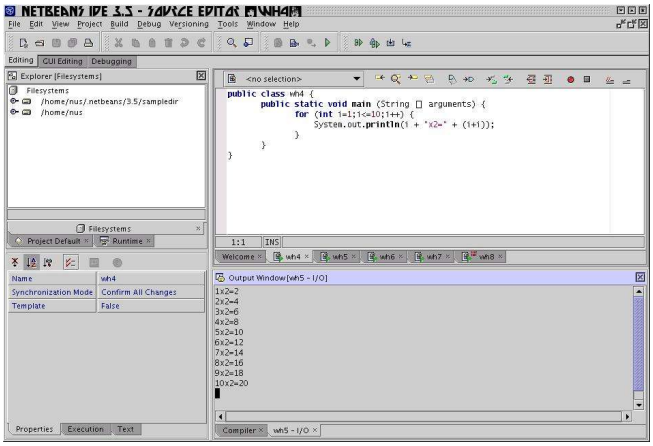
JAVA-Programme mit NetBeans entwickeln

NetBeans wird von der Firma SUN entwickelt und vertrieben; zumindest für nicht kommerzielle Zwecke kann diese IDE (*Integrated Developing Environment*) kostenlos verwendet werden.

Unter einer „IDE“ versteht man, wörtlich übersetzt, eine integrierte Entwicklungsumgebung. Dabei sollen ein Editor für den Quelltext, zahlreiche Tools zum Übersetzen, Testen und Ausführen eines Programmes, Hilfestellungen und die Unterstützung für so genannte Projekte zur Verfügung stehen. Im Allgemeinen können mehrere Dokumente gleichzeitig zur Bearbeitung offen stehen – wir sprechen dann von einer MDI (*Multi Document Interface*). So genannte „Wizards“ helfen bei Routine-Arbeiten, etwa beim Erstellen einer neuen Java-Klasse oder beim Entwerfen eines neuen Projekts.

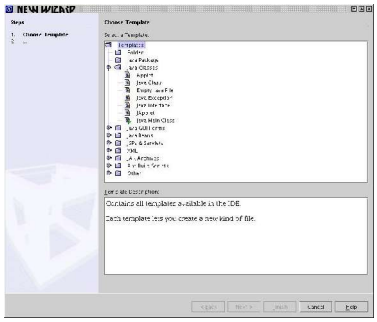
Nach dem Start erhalten wir eine für eine integrierte Entwicklungsumgebung typische Arbeitsfläche:



Im linken Bereich erkennen wir einen Dateimanager, in dem die verwendeten Arbeitsbereiche in einer Baumansicht dargestellt werden. Darunter können in der „Eigenschaften“-Ansicht (*properties*) verschiedene Informationen angezeigt werden. Im Quelltext-Fenster wird der gesamte Quelltext angezeigt und editiert. Das „syntax-highlighting“ erleichtert die Fehlersuche bei der Erstellung des Programmtextes. Alle Klassenobjekte können aus einer Auswahlliste gewählt werden: Ihre zugehörigen Eigenschaften werden dann in der „Eigenschaften“-Ansicht angezeigt. Mehrere Dateien können über die Karteireiter unterhalt des Editier-Bereiches ausgewählt werden. Das „Output“-Fenster unterhalb des Quelltextfensters enthält alle Meldungen, die vom Java-Programm bei der Laufzeit auf der Standardausgabe (Konsole) ausgegeben werden.

Eine neue Datei öffnen

Öffnet man eine neue Datei, so stellt der Wizard eine umfangreiche Auswahl verschiedener Vorlagen zur Verfügung:



Wir wählen beispielsweise eine „Applikation“ aus und legen als nächstes das Zielverzeichnis fest. Anschließend wählen wir den Klassennamen, wählen die notwendigen Packages und Felder. Der Wizard erzeugt schließlich ein umfangreiches Gerüst für den Quelltext, z.B:

```
/*
 * menger.java
 *
 * Created on 14. März 2004, 22:24
 */

package java.applet;

import java.awt.*;
import java.applet.*;

/**
 *
 * @author  nus
 */
public class menger extends java.awt.Component {

    /** Creates a new instance of menger */
    public menger() {
    }

    public void paint(Graphics g) {
    }

}
```

Die eingefügten Kommentarzeilen stellen eine Vorbereitung für eine standardisierte Dokumentation („JavaDoc“) dar.

Das Projekt erstellen, speichern, testen

Offensichtliche Fehler werden bereits beim Editieren angezeigt. Der Quelltext wird nach dem Speichern kompiliert (F9) – mit dem Befehl „Execute“ (F6) wird die Anwendung gestartet: Die Ausgabe wird entweder im Textausgabefenster angezeigt; oder es wird ein eigenes Fenster (Frame) geöffnet.

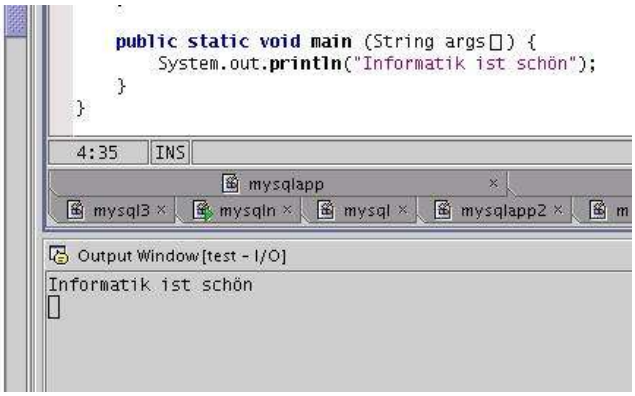
1. Beispiel: IDE handhaben

Wir erstellen mit Hilfe des Wizards für neue Dateien eine einfache JAVA-Klasse. Sie gibt nur einen Satz auf der Konsole aus:

Quelltext:

```
/*
 * test.java
 *
 * Created on 19. März 2004, 16:47
 */

/**
 *
 * @author  nus
 */
public class test {

    /** Creates a new instance of test */
    public test() {
    }

    public static void main (String args[]) {
        System.out.println("Informatik ist
schön");
    }
}
```

Das Compilieren (F9) und Ausführen (Execute, F6) liefert das folgende Ergebnis:



Das Ausgabefenster zeigt sowohl die Ergebnisse des Compilers (z.B. Fehlermeldungen) als auch die Konsolen-Ausgaben während der Laufzeit der Applikation an.

2. Beispiel: Rekursive Funktionen verwenden

Eine rekursiv definierte Funktion ruft sich selbst auf. Es ergibt sich von selbst, dass ein fortwährendes Aufrufen der Funktion durch eine entsprechende Abbruchbedingung verhindert werden muss.

Quelltext:

```
public class rekurs1 {

    /** Creates a new instance of rekurs1 */
    public rekurs1() {
    }

    public static void ausgabe(double x) {
        if (x > 2) {
            ausgabe(x/2);
            System.out.println(x);
        }
    }

    public static void main (String args[]) {
        ausgabe(20);
    }
}
```

Die Funktion `ausgabe()` ruft sich so lange selbst auf, bis der Wert der Variablen `x` die Zahl `2` unterschreitet. Wir erhalten:

```
2.5
5.0
10.0
20.0
```

Wo liegt der Unterschied?

```
    public static void ausgabe(double x) {
        if (x > 2) {
            System.out.println(x);
            ausgabe(x/2);
        }
    }
```

Diese Variante liefert:

```
20.0
10.0
5.0
2.5
```

Beachte, in welcher Reihenfolge die Funktion `ausgabe()` aufgerufen wird!

Bemerkung: Rekursive Funktionen erlauben im Allgemeinen sehr rasche und effiziente Algorithmen (z.B. Such- und Sortierverfahren). Wir werden rekursive Funktionen für das Erstellen so genannter **Fraktale** verwenden.

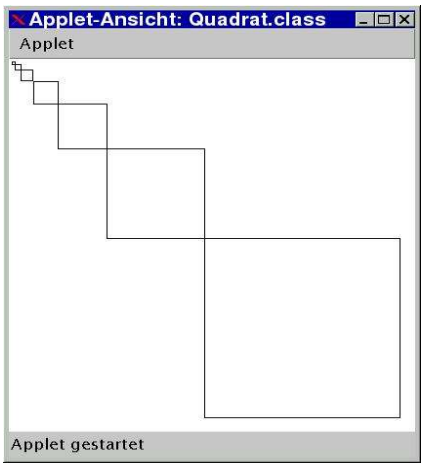
3. Beispiel: Ein einfaches Fraktal

Für die grafische Darstellung verwenden wir Grafikfunktionen von JAVA und geben in einem Applet aus. Wir erinnern uns, dass Applet-Klassen in einem HTML-Dokument eingebunden werden müssen. Dazu verwenden wir das <APPLET CODE="name.class" WIDTH="400" HEIGHT="400"></APPLET>-Element.

Quelltext:

```
...
import java.awt.*;

public class Quadrat extends javax.swing.JApplet {

    /** Creates a new instance of Quadrat */
    public Quadrat() {
    }

    public void paint (Graphics g) {
        quadrat(this, 200);
    }

    public void quadrat(Container ct, double x) {
        Graphics g = ct.getGraphics();
        if (x > 2) {
            quadrat(ct, x/2);
            g.drawRect((int) x, (int) x, (int) x,
(int) x);
        }
    }
}
```

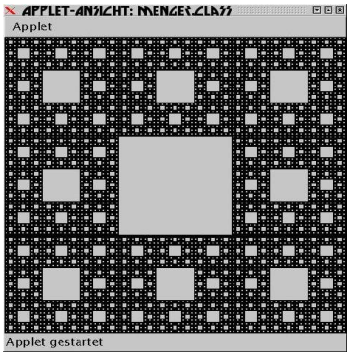
Die IDE erstellt beim Compilieren automatisch eine zugehörige HTML-Datei, sodass der Appletviewer beim Ausführen folgendes Ergebnis anzeigt:



Da die Grafik-Methode `drawRect()` ganzzahlige Argumente erwartet, werden die Dezimalzahlen mittels Type-Casting in Ganzzahlen umgewandelt.

4. Beispiel: Der Menger-Schwamm

Der Menger-Schwamm entspricht dem 3-dimensionalen Gegenstück des folgenden Fraktales:



Quelltext:

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

public class menger extends java.applet.Applet {

    public void init() {
    }

    public void paint (Graphics g) {
        meng(this, 240, 240, 240);
    }

    public void meng(Container ct, double c,
                double x, double y) {
        Graphics g = ct.getGraphics();
        if (c > 0.5) {
            meng(ct, c/3, x - 2*c/3, y - 2*c/3);
            meng(ct, c/3, x, y - 2*c/3);
            meng(ct, c/3, x + 2*c/3, y - 2*c/3);
            meng(ct, c/3, x - 2*c/3, y);
            meng(ct, c/3, x + 2*c/3, y);
            meng(ct, c/3, x - 2*c/3, y + 2*c/3);
            meng(ct, c/3, x, y + 2*c/3);
            meng(ct, c/3, x + 2*c/3, y + 2*c/3);
            g.drawRect((int)(x-c), (int) (y-c),
(int)(2*c), (int) (2*c));
        }
    }
}
```

Beachte auch hier die korrekte Verwendung einer HTML-Datei und das <applet>-Tag!

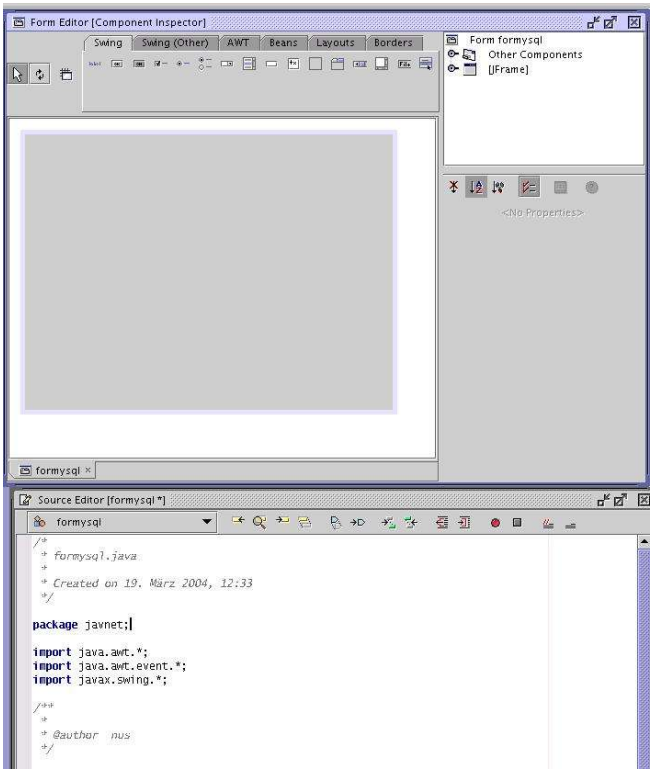
Bis jetzt haben wir keine Eingriffe durch den Benutzer zugelassen. Um Eingaben oder Auswahlen zur Laufzeit entgegen zu nehmen, sind so genannte Formulare notwendig. Wir erstellen sie im nächsten Abschnitt mit Hilfe des Formular-Generators.

Der Formulargenerator

Benutzereingaben sind während der Laufzeit über so genannte Formular-Objekte möglich (Texteingaben, Schaltflächen, Auswahlboxen, Menüs, Scrollbalken, Schiebefenster, ...). Die zum Teil komplizierte Syntax dieser Objekte und ihr Zusammenspiel kann mit Hilfe des „Form-Editors“ vereinfacht werden.

Allgemeine Vorgangsweise:

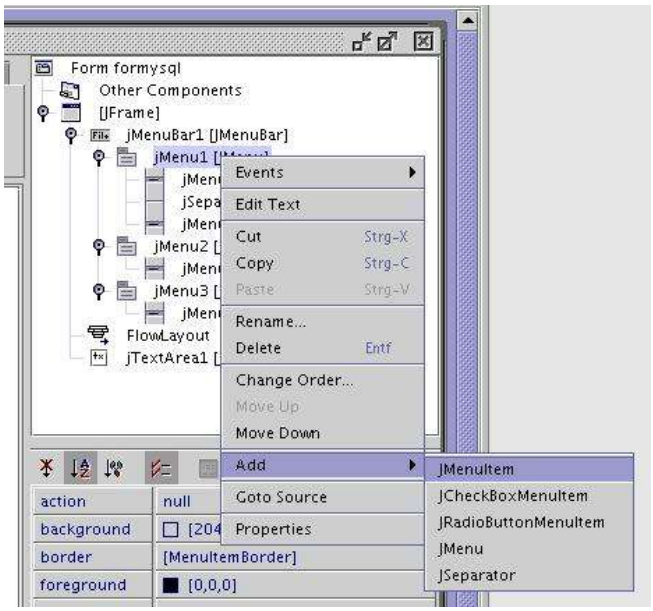
1. Wähle mit dem „New Wizard“ beispielsweise die JAVA-GUI-Form JFrame Form aus.
2. Lege das Arbeitsverzeichnis und den Klassennamen fest.
3. Füge die Interfaces ActionListener, ItemListener hinzu
4. Erzeuge zwei eigene Methoden (treiber_laden(), ausgabe ())
5. Schließe den Wizard ab und ergänze ggf. notwendige import-Zeilen (`import java.awt.*; import java.awt.event.*; import javax.swing.*;`)
6. Öffne den GUI-Editor:



7. Wähle die gewünschten Objekte (z.B. Menü, Menü-Einträge menuItems, TextArea, ...)
8. Wechlse in den Source-Editor ...

1. Beispiel: Menüleiste erzeugen

Zunächst ist aus dem Karteireiter das Swing-Objekt `JmenuBar` auszuwählen. Mit Hilfe der rechten Maustaste wählt man ein kontextbezogenes Menü, aus dem jeweils „passende“ Objekte hinzugefügt werden können („Add“). Zunächst müssen `JMenu`-Objekte für jede Menü-Leiste und schließlich `JMenuItems` hinzugefügt werden. In den entsprechenden Dialogen werden dabei die Eigenschaften dieser Elemente festgelegt.



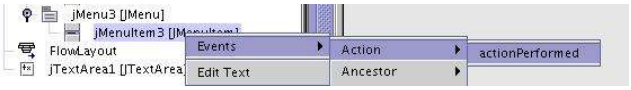
2. Beispiel: `JTextArea`-Objekt verwenden, Formular testen

Für die Ausgabe der weiteren Objekte müssen wir zunächst ein Layout auswählen. Anschließend fügen wir ein `JtextArea`-Objekt ein und legen beispielsweise Hintergrund-, Textfarbe und die Anzahl der Spalten und Reihen fest.

Schließlich kann das Formular hinsichtlich seines Layouts getestet werden:

3. Beispiel: Programmcode ergänzen

Mit Hilfe des Formular-Editors wurden alle Definitionen der sichtbaren Objekte erzeugt. Der Programmcode wird ausschließlich in den dafür vorgesehenen Methoden eingetragen. Um beispielsweise eine bestimmte Aktion an einen Menüeintrag zu binden muss dem JmenuItem-Objekt zunächst eine actionPerformed-Methode zugewiesen werden:



Dabei wird folgender Code erzeugt (fett gedruckt):

```
jMenu3.setText("Hilfe");
jMenuItem3.setText("\u00dcber");
jMenuItem3.addActionListener(new
            java.awt.event.ActionListener() {
    public void actionPerformed
            (java.awt.event.ActionEvent evt) {
      jMenuItem3ActionPerformed(evt);
    }
});
jMenu3.add(jMenuItem3);
jMenuBar1.add(jMenu3);
```

Der eigentliche Code wird innerhalb der Methode `jMenuItem3ActionPerformed(evt);` deklariert:

```
private void jMenuItem3ActionPerformed
            (java.awt.event.ActionEvent evt) {
  // Add your handling code here:
}
```

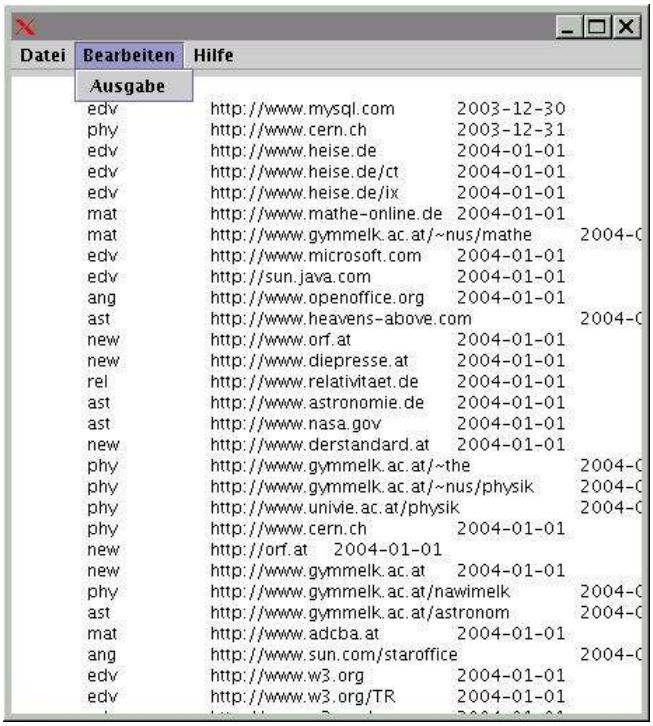
Über den Menüpunkt „Datei“ - „Ausgabe“ soll nun der Inhalt einer bestimmten MySQL-Tabelle ausgegeben werden. D

```
private void jMenuItem2ActionPerformed
     (java.awt.event.ActionEvent evt) {
  // Add your handling code here:
  treiber_laden();
  ausgabe();
}
```

Die beiden Methoden `treiber_laden()` und `ausgabe()` werden weiter unten deklariert; nach dem Laden des Datenbank-Treibers wird die Verbindung zum MySQL-Server hergestellt und ein entsprechendes SQL-Statement ausgeführt. Die Abfrageergebnisse werden zeilenweise über das `JTextArea-Objekt` ausgegeben.

```
void treiber_laden() {
  try {
    Class.forName("com.mysql.jdbc.Driver");
  }
  catch (Exception e) {
    System.out.println(e.getMessage());
  }
}

void ausgabe() {
  try {
    Connection con = DriverManager.getConnection
("jdbc:mysql://localhost/mdat_nus","info","info");
    Statement stmt = con.createStatement();
    ResultSet rslt = stmt.executeQuery("select *
             from weblinks");

    while (rslt.next()) {
      jTextArea1.append("\n" + rslt.getInt(1) +
"\t" + rslt.getString(2) + "\t" + rslt.getString
(3) + "\t" + rslt.getDate(5));
    }
    stmt.close();
    con.close();
  }
  catch (Exception e) {
    jTextArea1.append(e.getMessage());
  }
}
```

Damit erhalten wir eine JAVA-Anwendung, die über ein vertrautes Menü gesteuert werden kann:



Aufgaben:

1. Ergänze Programmteile zu den beiden Menüeinträgen „Datei“ - „Optionen“ und „Hilfe“ - „über“!
2. Formatiere die Ausgabe der MySQL_Tabelle innerhalb der Textausgabefläche!
3. Verwende den Formulargenerator, um schon erstellte Applikationen zu einem Programmpaket zusammenzustellen (z.B. Verschiedene Fraktale aus einem Menü auswählen, Erläuterung zu Rekursion, Hilfe, Optionen, etc.