Programmierung in Java

Programmierung grafischer Benutzeroberflächen mit Eclipse WindowBuilder Editor

Kapitel 1.3, 13

Prof. Dr. Markus Eikelberg

Rel. 7.30

Für diese Unterlagen gilt:

- © 1999, 2020 Prof. Dr. Markus Eikelberg
- LaTeX Version, Dokument `javakurs-kap-1.3_13.pdf`
- Druckdatum 14. März 2020
- Alle Urheber- und Leistungsschutzrechte vorbehalten.
- Keine unerlaubte Weitergabe dieser Unterlagen, keine unerlaubte Vervielfältigung, Vermietung, Sendung, Aufführung, Veröffentlichung oder Verleih ohne das schriftliche Einverständnis des Autors.
- Hinweis zum Urheberrecht: Soweit nicht anders gekennzeichnet ist der Urheber stets identisch mit dem Verfasser einer zitierten Quelle
- Das Urheberrecht für alle nicht als Zitat gekennzeichneten Abbildungen liegt beim Autor.

Rechtliche Hinweise - Haftungsausschluss

Die Benutzung dieses Skripts und die Umsetzung der darin enthaltenen Informationen erfolgt ausdrücklich auf eigenes Risiko. Der Autor kann für Schäden jeder Art, die sich durch die Nutzung der Unterlagen ergeben, aus keinem Rechtsgrund eine Haftung übernehmen. Haftungsansprüche gegen den Autor wegen Schäden materieller oder ideeller Art, die durch die Nutzung oder Nichtnutzung der Informationen bzw. durch die Nutzung fehlerhafter und/oder unvollständiger Informationen verursacht wurden, sind grundsätzlich ausgeschlossen. Rechts- und Schadenersatzansprüche sind daher ausgeschlossen. Dieses Skript wurde mit größter Sorgfalt erarbeitet. Der Autor übernimmt jedoch keine Gewähr für die Aktualität, Korrektheit, Vollständigkeit und Qualität der bereitgestellten Informationen. Druckfehler und Falschinformationen können nicht vollständig ausgeschlossen werden. Der Autor übernimmt keine Haftung für die Aktualität, Richtigkeit und Vollständigkeit des Inhalts dieses Skripts, ebenso nicht für Druckfehler. Es kann keine juristische Verantwortung sowie Haftung in irgendeiner Form für fehlerhafte Angaben und daraus entstandenen Folgen vom Autor übernommen werden.

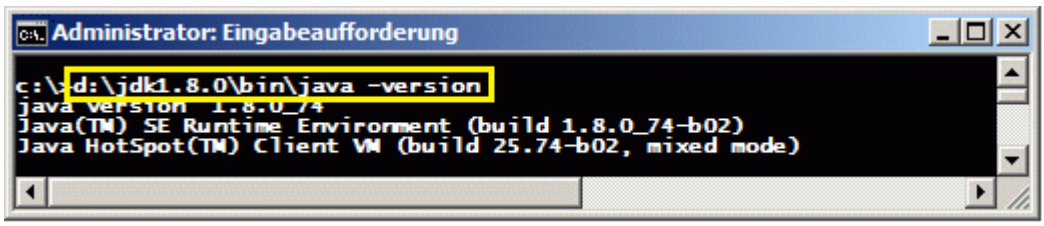
Inhaltsverzeichnis

1 Einführung in Java **2**
1.3 Software zur Programmierung in Java 2
1.3.1 Java Development Kit 2
1.3.2 Eclipse 9

13 Programmierung grafischer Benutzeroberflächen mit Eclipse WindowBuilder Editor **24**
13.1 Programmierung eines Fensters 24
13.2 Ereignisbehandlung 40
13.3 Modale Dialogfenster 48
13.3.1 Modale Fenster 48
13.3.2 JOptionPane Dialogfenster 51
13.3.3 JFileChooser Dateidialogfenster 57
13.4 Programmierung von GUI Komponenten mit dem WindowBuilder Editor 61
13.4.1 Struktur des vom WindowBuilder Editor erzeugten Quellcode . . 61
13.4.2 Beschriftungs- und Bedienelemente 70
13.4.3 Programmierung von Modellen für Daten 81
13.4.4 Elemente für Auswahlvorgänge 90
13.4.5 Editierbare Oberflächenelemente 109
13.5 Programme mit grafischer Benutzeroberfläche und Datenbankzugriff . . 125
13.5.1 Aufbau und Abbau einer Datenbankverbindung mit dem Logon-Panel 125
13.5.2 Ausführung von SQL-Anweisungen 132

1.3 Software zur Programmierung in Java

In diesem Kapitel wird vorausgesetzt, dass das Java Development Kit in einem Verzeichnis `D:\jdk1.8.0` installiert ist. Sie können dies überprüfen, in dem Sie folgenden Befehl eingeben:



Java ist eine verhältnismäßig junge Programmiersprache, deren Geschichte 1991 begann. Ausgangspunkt war ein Projekt bei der Firma **Sun Microsystems** zur Entwicklung von Software für Geräte der Konsumelektronik. Bezüglich dieser ursprünglichen Zielrichtung war das Projekt jedoch zunächst nicht erfolgreich.

In den Jahren 1993/94 verbreitete sich jedoch unabhängig von dieser Entwicklung das **World Wide Web (WWW)** und mit den ersten HTML-Browsern entstand die Möglichkeit zur graphischen Darstellung von Informationen im Internet. Im Rahmen dieser Entwicklung erkannte man die durch Java im Bereich des Internet eröffneten Möglichkeiten. Ende 1995 kam mit dem Browser **Netscape Navigator 2.0** der erste Browser auf den Markt, der Java bei der Anzeige von WWW-Seiten umsetzen konnte. Dementsprechend wurde bei Sun Microsystems weiter an Java gearbeitet und zeitgleich mit dem Netscape Navigator 2.0 erschien Anfang 1996 der erste Java-Compiler, das **Java Development Kit (JDK) 1.0** von Sun Microsystems. Im April 2009 wurde Sun Microsystems vom Oracle Konzern gekauft, so dass Java aktuell ein Oracle Produkt ist.

Das Java Development Kit wurde von Sun Microsystems schnell und umfassend weiterentwickelt und wird kostenlos verteilt.

1.3.1 Java Development Kit

Das Java Development Kit (JDK) stellt neben dem Java-Compiler auch die virtuelle Maschine bereit, die für die Ausführung von Java-Programmen benötigt wird. Daher wird zumindest die Laufzeitumgebung (Java Runtime Environment JRE) aus dem JDK bei der Ausführung von Java-Programmen benötigt. Für die Softwareentwicklung bietet das JDK **keinerlei Bedienerkomfort** - man könnte auch sagen, daß der Bedienerkomfort des JDK der Arbeitsweise vor (!) der Einführung von Windows entspricht. Dementsprechend geschieht die Arbeit mit dem JDK auf einem Windows-Rechner auch in der sogenannten Konsolbox (unter Windows-Systemen zu erreichen über den Menüpunkt "Eingabeaufforderung"). Die verschiedenen angebotenen Entwicklungsumgebungen zu Java enthalten stets eine JDK-Version, auf die beim Compile zurückgegriffen wird. Damit wird auch bei der Verwendung einer Entwicklungsumgebung im Hintergrund stets eine JDK-Version verwendet.

1.3.1.1 Grundform des Aufrufs von Compiler und Virtueller Maschine (JVM)

Wir erklären die Grundform des Aufrufs von Compiler und Virtueller Maschine anhand des Beispielprogramms **bsp01004.Konsoladd**, das in Bezug auf den Java Quellcode abgesehen vom geänderten package-Namen ansonsten mit dem Beispielprogramm bsp01001.Konsoladd übereinstimmt. Das Programm bsp01004.Konsoladd.java können Sie wie auch alle anderen Beispielprogramme dieses Kurses auf zwei verschiedenen Wegen wie folgt finden:

Alle Programme sind nach folgendem Schema benannt: Ein Programm mit dem Namen **bspnnmmm.Progname** ist ein Beispielprogramm zu Kapitel nn. Den Quellcode **Progname.java** dieses Programms sowie alle zugehörigen Dateien finden Sie, indem Sie entweder

1. **auf der DVD** zu diesem Kurs im Verzeichnis `javakurs-beispiele\ebvnn\`**`jprnnmmm\bspnnmmm`** oder
2. **im FTP-Verzeichnis des Intranet-Servers des Institutes für Mathematik und Informatik (PC132W.bsnw.fb4.fh-bochum.de)** im Verzeichnis `eikelberg\javakurs\beispiele\ebvnn\`**`jprnnmmm\bspnnmmm`**.

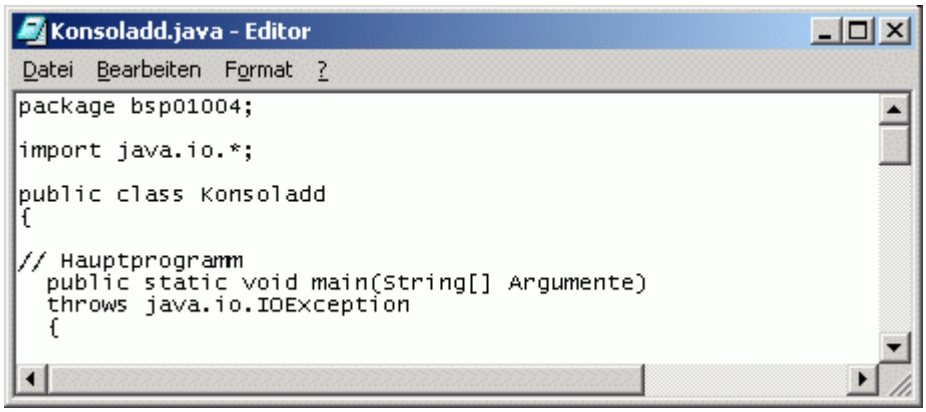
Bei der Java-Programmierung spricht man in der Regel statt von Programmen allgemeiner von sogenannten Klassen. Daher tritt im nachfolgenden Text oft allgemeiner die Bezeichnung **bspnnmmm.Klassenname** für die Quellcode-Datei **Klassenname.java** auf.

Der Server PC132W.bsnw.fb4.fh-bochum.de erlaubt in der Regel einen Zugriff via anonymous FTP (anonyme Anmeldung mit Benutzerkennung **anonymous** nach Eingabe von `ftp PC132W.bsnw.fb4.fh-bochum.de` in der Konsolbox oder aber nach Eingabe von `ftp://PC132W.bsnw.fb4.fh-bochum.de` in einem Browser eines beliebigen Rechners im Netzwerk der Hochschule Bochum bzw. bei Bestehen einer VPN-Verbindung zur Hochschule Bochum).

Der folgende Ablauf legt zugrunde, daß Sie das Verzeichnis `jpr01004` von der DVD oder vom FTP-Server in ein Verzeichnis `$daten\ebv01` auf einem Laufwerk Ihres Rechners mit dem Namen `E:` kopiert haben. Das Programm `Konsoladd.java` finden Sie dann im Verzeichnis

`E:\$daten\ebv01\jpr01004\bsp01004`

Sie können dann den Quellcode in `Konsoladd.java` wie in 1.2.1 beschrieben mit Hilfe eines Editors ansehen. Mit dem Windows-Editor **notepad** (Anwendung "Editor" in der Windows-Programmgruppe "Zubehör") erhalten Sie zum Beispiel folgende Bildschirmanzeige:

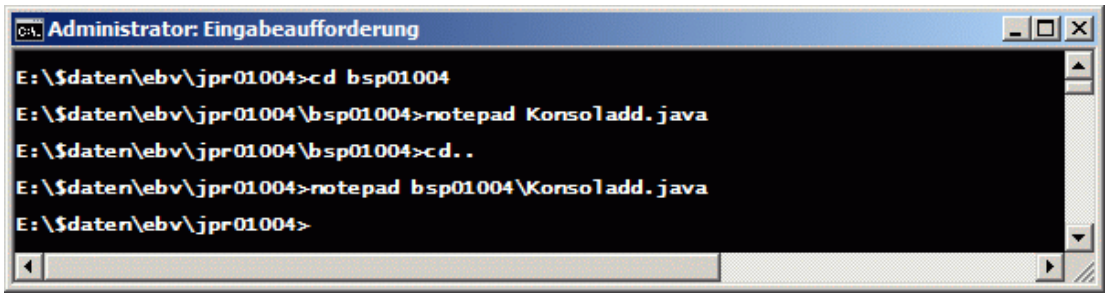


Sie können leicht sehen, daß der Quellcode von bsp01004.Konsoladd bis auf den neuen package-Namen identisch ist mit dem Quellcode von bsp01001.Konsoladd aus Kapitel 1.1 und 1.2.

Da die Arbeit mit dem JDK in der Konsolbox ("Eingabeaufforderung") geschieht, sei hier kurz der Aufruf von notepad aus der Konsolbox erklärt. Die **Syntax** (Beschreibung der sprachlichen bzw. "grammatischen" Regeln zur korrekten Eingabe eines Befehls oder einer Programmieranweisung) des **Aufrufs von notepad** lautet in der vollständigen Form wie folgt:

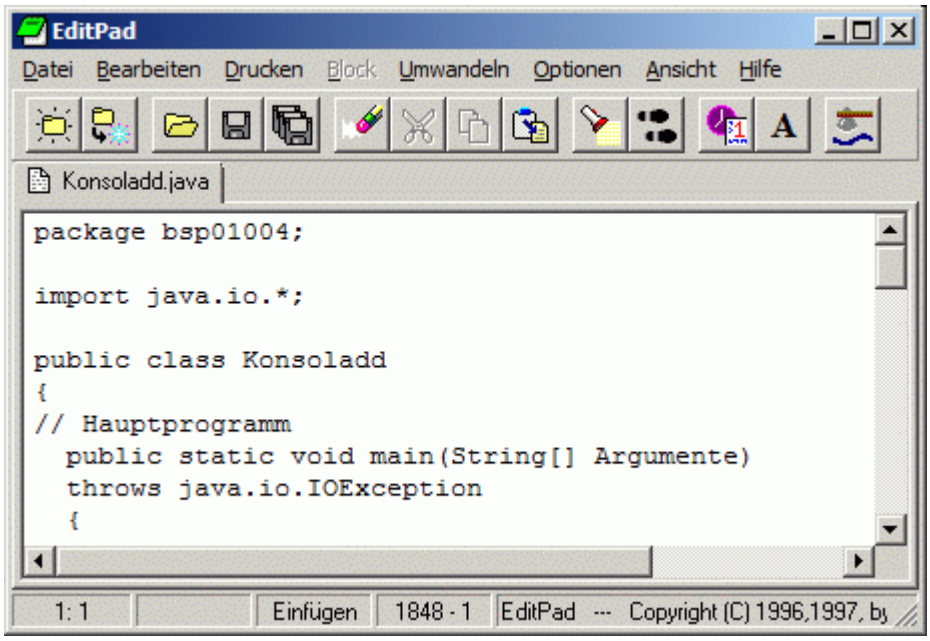
`notepad` *`laufwerk:\..pfadangabe..\dateiname`*`.java`

Dabei wird für *laufwerk* das aktuelle Laufwerk, für *..pfadangabe..* der Pfad, unter der die Datei zu finden ist, und für *dateiname* der Dateiname der Java Quelldatei ohne Zusatz eingesetzt. Es gelten also die üblichen Regeln für die Arbeit in einem Konsolbildschirm bzw. einer Konsolbox. Ist der aktuelle Pfad bereits gleich dem Pfad, in dem sich die Datei befindet, so lautet der Aufruf für die Datei Konsoladd.java zum Beispiel wie im folgenden Screenshot (Momentaufnahme einer Bildschirmanzeige):



Für die Arbeit in der Konsolbox sind also die üblichen Kenntnisse für den Umgang mit dem Betriebssytem ohne grafische Benutzeroberfläche erforderlich. Statt des Microsoft Windows Editors notepad können im übrigen **andere leistungsfähigere Editoren** verwendet werden. Wichtig für die Programmierung ist mindestens die **Angabe der aktuellen Zeilen- und Spaltenanzahl**, in der sich der Cursor befindet. Hilfreich ist ferner eine **Syntaxeinfärbung**, also die farbliche Hervorhebung von Sprachelementen der verwendeten Programmiersprache. Um das zweite Kriterium erfüllen zu können, müssen dem Editor natürlich die aktuellen Sprach-Regeln der verwendeten Programmiersprache bekannt sein.

Das erste Kriterium, nicht jedoch das zweite Kriterium erfüllt zum Beispiel der *Postcard-Ware* Editor **EditPad**, der im im FTP-Verzeichnis `eikelberg\javakurs\download\editpad` des Intranetservers PC132W.bsnw.fb4.fh-bochum.de gefunden werden kann. Eine Bildschirmanzeige von Konsoladd.java mit EditPad sieht wie folgt aus:



Für die nachfolgenden Erklärungen wird vorausgesetzt, dass der Prompt in der Eingabeaufforderung auf das Java-Basisverzeichnis (also das jpr-Verzeichnis) positioniert ist, d.h. dass die Pfadangabe im Prompt mit dem Namen des jpr-Verzeichnisses endet:

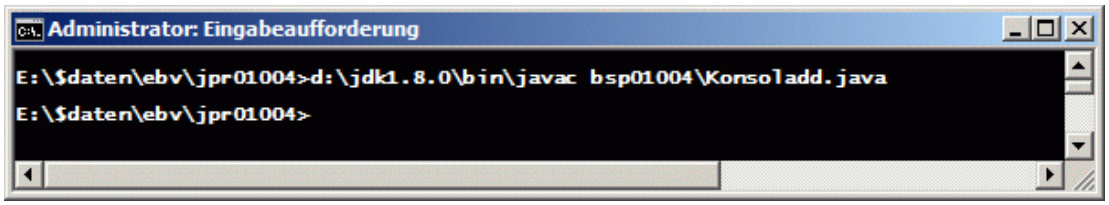
Nachdem die Java Quelldatei mit einem Editor erstellt oder abgeändert wurde, muß sie jetzt wie in Kapitel 1.2 erklärt **mit Hilfe eines Java-Compilers in Bytecode übersetzt** werden. Der Aufruf des JDK-Compilers lautet in vollständiger Form

`D:\jdk1.8.0\bin\javac` *`paketname\dateiname`*`.java`

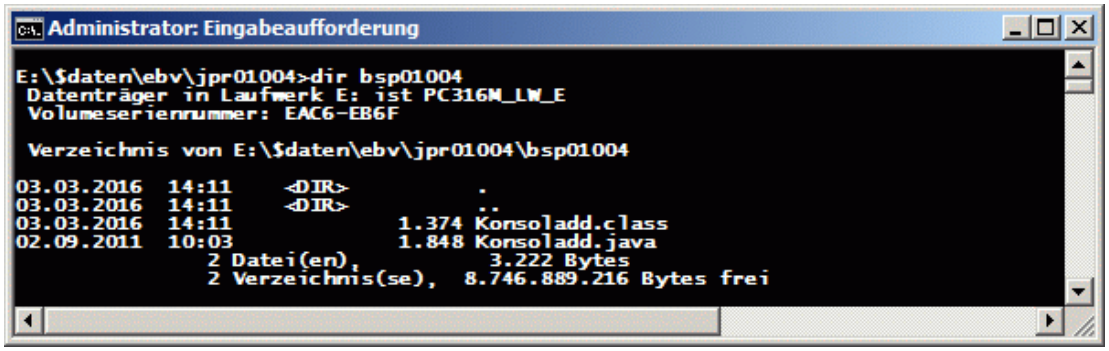
wobei `paketname` der in der **package-Anweisung** genannte Name des Programmpaketes ist und wobei der aktuelle Pfad in der Eingabeaufforderung auf das Basisverzeichnis eingestellt werden muß. Besteht der Paketname aus mehreren Namen `pname1.pname2. ... .pnamen`, so lautet der Aufruf des JDK-Compilers

`D:\jdk1.8.0\bin\javac` *`pname1\pname2\...\pnamen\dateiname`*`.java`

Das Verzeichnis `paketname` bzw. `pname1.pname2. ... .pnamen` wird im folgenden **Paketverzeichnis** genannt. Bezogen auf unser Beispiel erfolgt der Aufruf wie im folgenden Screenshot:



Wie man sieht hat der Java-Compiler die Eigenart, dass er bei einer fehlerfreien Übersetzung des Java-Programms keine Meldung ausgibt. Da beim Compile keine Fehler gefunden wurden, erstellt der Compiler eine Datei mit Bytecode mit dem Namen *`dateiname`*`.class`, die im Paketverzeichnis gespeichert wird. Bezogen auf unser Beispiel finden wir also im Paketverzeichnis bsp01004 zusätzlich zur java-Quelldatei eine Datei mit Namen `Konsoladd.class`:



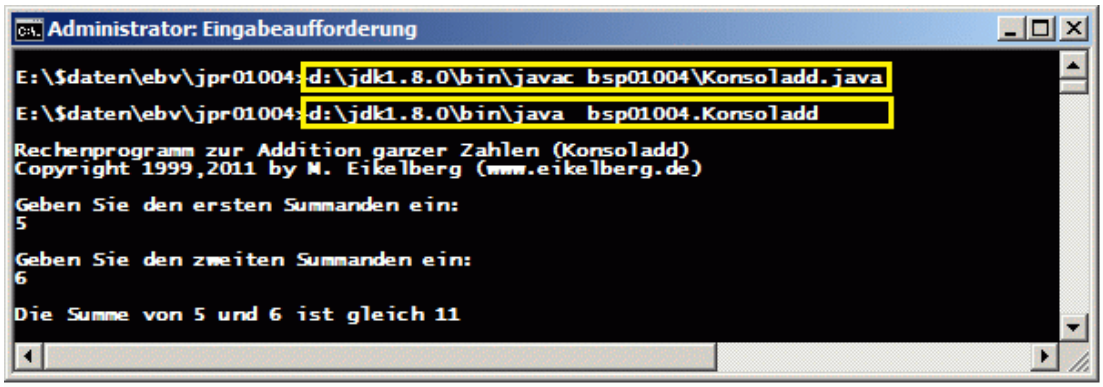
Ist die Java-Klasse erfolgreich compiliert, kann sie **zur Ausführung an die Java Virtual Machine (JVM)** übergeben werden. Dies geschieht durch Aufruf der JVM mit Hilfe des Befehls `java`. Die vollständige Syntax des Aufrufs der JVM lautet wie folgt:

```
D:\jdk1.8.0\bin\java paketname.dateiname
```

bzw. bei einem mehrstufigen Paketnamen

```
D:\jdk1.8.0\bin\java pname1.pname2. ... .pnamen.dateiname
```

Für das Beispiel bsp01004.Konsoladd geschieht der Compile sowie der anschließende Programmstart mit Hilfe der JVM wie folgt:

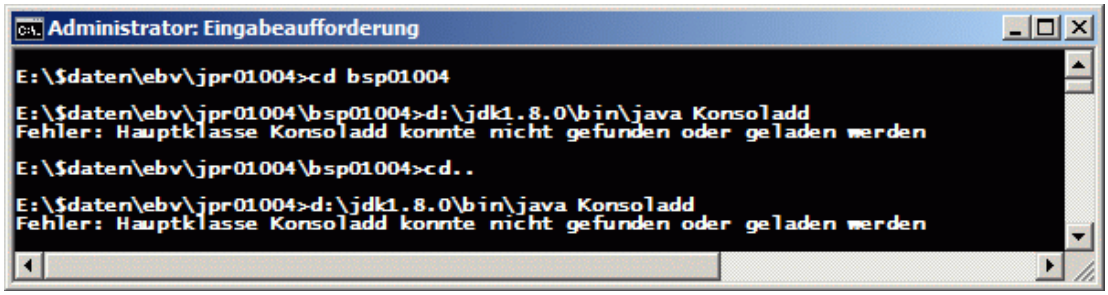


Man beachte folgende Unterschiede beim Aufruf von java und javac:

	Aufruf von javac.exe	**Aufruf von java.exe**
Aufruf	javac	java
Zeichen zwischen Paketname und Klassenname	Backslash	Punkt
Namenserweiterung	.java wird angegeben	.class wird nicht angegeben

<u>Hinweis:</u> Die Klassendatei wird von der JVM ausgehend vom Basisverzeichnis unter Verwendung der sich aus dem Paketnamen ergebenden Pfadangabe in einem entsprechenden Unterverzeichnis des aktuellen Verzeichnisses (nämlich dem Paketverzeichnis) gesucht! Ferner muss der beim Aufruf der JVM angegebene Paketname mit dem in der Java-Quelldatei in der package-Anweisung angegebenen Paketnamen übereinstimmen! Entsprechend muss der Prompt auf das Java Basisverzeichnis eingestellt sein.

Bei Nichtbeachtung dieser Tatsache ergeben sich folgende Fehler:



Man beachte in obigem Screenshot, dass nach Eingabe des letzten Befehls vom der JVM eine Fehlermeldung vom Typ "NoClassDefFoundError" ausgegeben wurde. Wenn man jedoch genau hinsieht, kann man erkennen, dass die JVM sehr wohl die Klasse Konsoladd.java gefunden hat, sich jedoch an der Package-Anweisung stört (wrong name: bsp01004/Konsoladd). Dies zeigt, dass die JVM stets vom Java-Basisverzeichnis aufgerufen werden muss, so dass der in der package-Anweisung genannte Pfad zwischen den Verzeichnis liegt, in dem die JVM aufgerufen wurde, sowie dem Verzeichnis, in dem die Klassendatei liegt.

1.3.1.2 Compile und Ausführung bei Verwendung von Klassen, die nicht zum JDK gehören

Sehr oft werden in Java-Programmen Klassen verwendet, die nicht zum Standardumfang des JDK gehören. In diesem Fall müssen folgende Punkte erfüllt sein:

- Die Klassen sollten in Form einer jar-Datei (eine Art Java-Archivdatei) bereitgestellt werden.
- Diese jar-Datei muss im Java-Basisverzeichnis gespeichert sein.
- Die jar-Datei muss beim Compile mit Hilfe der Compileroption **`-classpath`** bzw. **`-cp`** angegeben werden.
- Die jar-Datei muss bei der Ausführung mit Hilfe der Compileroption **`-classpath`** bzw. **`-cp`** angegeben werden.

Die Copileroptionen **`-classpath`** und **`-cp`** sind äquivalent, **`-cp`** ist lediglich die abgekürzte Schreibweise.

Beispiel für diese Situation ist das Projekt jpr01102. Dieses Projekt enthält die Quelldateien bsp01102\GuiAdd.java und

bsp01102\GuiAddFrame.java. In der Quelldatei bsp01102\GuiAddFrame.java befindet sich u.a. folgender Programmcode:

```
package bsp01102;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import tools01102.CopyrightPanel;
import tools01102.KommentarPanel;

public class GuiAddFrame extends JFrame
{
...
}
```

Dies bedeutet, dass die Bereitstellung der Klassen folgender Pakete im Quellcode von bsp01102\GuiAddFrame.java vorausgesetzt wird:

- java.awt
- java.awt.event
- javax.swing
- tools01102

Die ersten drei Pakete gehören zum Standardumfang des JDK und müssen nicht in jar-Dateien bereitgestellt werden. Dagegen gehört das Paket tools01102 nicht zum Standardumfang des JDK und muss daher in einer jar-Datei bereitgestellt werden. Die jar-Datei mit dem Paket tools01102 hat den Namen `tools01102.jar` und ist im Basisverzeichnis jpr01102 gespeichert.

Der Aufruf des Compilers lautet damit:

```
D:\jdk1.8.0\bin\javac -cp .;tools01102.jar bsp01102\GuiAddFrame.java
D:\jdk1.8.0\bin\javac -cp . bsp01102\GuiAdd.java
```

oder kürzer einfach

```
D:\jdk1.8.0\bin\javac -cp .;tools01102.jar bsp01102\*.java
```

Hinter der Compiler-Option -cp stehen also folgende Einträge, die durch ein Semikolon getrennt werden:

- . (ein Punkt) - steht für das Verzeichnis, von dem aus javac aufgerufen wurde, und deutet somit auf das Java-Basisverzeichnis
- tools01102.jar - Name der jar-Datei, die das Paket `tools01102` enthält. Diese Datei wird im Java-Basisverzeichnis gesucht.

In der Praxis tritt es oft auf, dass mehrere jar-Dateien genannt werden müssen. In diesem Fall werden alle erforderlichen jar-Dateien nach der classpath-Option genannt, jeweils durch ein Semikolon voneinander getrennt.

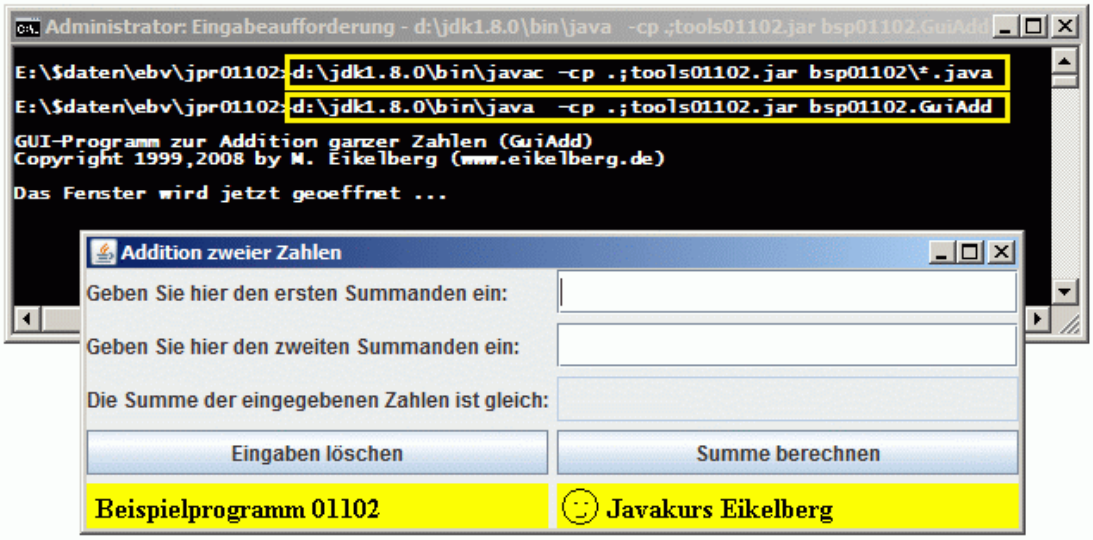
Auch beim Aufruf der JVM müssen die jar-Dateien wieder genannt werden. Beim Aufruf der JVM muss dabei die Klasse angegeben werden, die den mit

```
public static void main(String[] ...)
```

gekennzeichneten Abschnitt des Quellcode enthält. Im Fall des Beispielprojektes jpr01102 ist dies die Quelldatei GuiAdd.java. Entsprechend lautet der Aufruf der JVM wie folgt:

```
D:\jdk1.8.0\bin\java -cp .;tools01102.jar bsp01102.GuiAdd
```

Der nachfolgende Bildschirmausdruck zeigt die Wirkung der Aufrufe von java und javac:

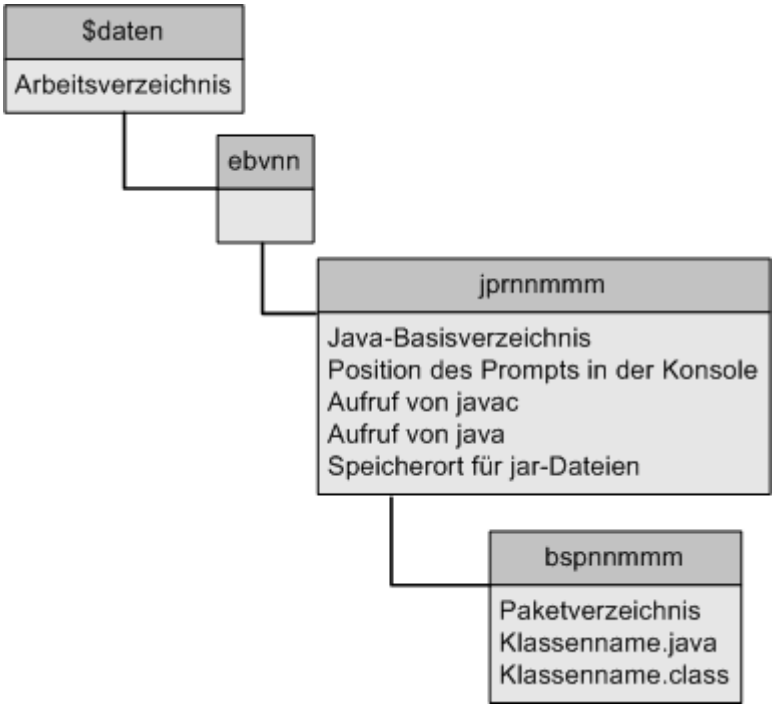


1.3.1.3 Regeln für den Aufruf von Compiler und Virtueller Maschine (JVM)

Bei der Anordnung der Java-Quelldatei sowie bei dem Aufruf von java und javac müssen also folgende Regeln unbedingt beachtet werden:

1. Für die Speicherung von Java-Programmen wählt man in der Regel jeweils ein separates Verzeichnis. Dieses Verzeichnis wird **Java-Basisverzeichnis** bzw. kurz **Basisverzeichnis** genannt.
2. Alle Pfadangaben im Zusammenhang mit der Java-Programmierung werden **relativ zum Basisverzeichnis** - also mit dem Basisverzeichnis beginnend - ermittelt.
3. Der in der **package-Anweisung** angegebene Paketname bestimmt das Unterverzeichnis des Basisverzeichnisses, in dem die jeweilige Java-Quelldatei bzw. die zugehörige Klassendatei gespeichert wird.
4. **Beim Aufruf von Compiler (javac) und JVM (java) muss der in der Konsolbox eingestellte Pfad im Basisverzeichnis positioniert werden.** Ferner müssen beim Aufruf von Compiler und JVM die Paketnamen geeignet angegeben werden.
5. In den Java-Klassen verwendete Klassen, die nicht selber programmiert wurden und die nicht zum Standardumfang des JDK gehören, müssen in Form von jar-Dateien bereitgestellt werden. Diese **jar-Dateien müssen im Java-Basisverzeichnis gespeichert werden**. Beim Compile sowie beim Programmaufruf müssen die jar-Dateien mit Hilfe der **Option -classpath** bzw. **Option -cp** angegeben werden.

Die obigen Regeln werden in folgender Grafik verdeutlicht. Dabei wird das Java-Basisverzeichnis in einem Verzeichnis mit Namen `ebvnn` angeordnet. Dieses Verzeichnis hat hier noch keine Bedeutung, wird jedoch für die in Kapitel 1.3.2 eingeführte Entwicklungsumgebung Eclipse benötigt. Das Verzeichnis `ebvnn` befindet sich seinerseits in einem Verzeichnis `$daten`. Dieses Verzeichnis stellt den Arbeitsbereich auf dem jeweils verwendeten System dar.



Das Beispielprogramm bsp01015.Konsoladd verdeutlicht noch einmal die Situation. Dieses Programm ist dem Beispielprogramm bsp01004.Konsoladd sehr ähnlich, verwendet jedoch eine Klasse des Paketes de.eikelberg.iotools. Dieses Paket wird im Java-Basisverzeichnis jpr01015 in einer jar-Datei mit Namen de_eikelberg_iotools.jar bereitgestellt (vgl. Kapitel 2.1). Daher geschieht der Compile sowie der Programmaufruf wie folgt:

```
Administrator: Eingabeaufforderung

E:\$daten\ebv\jpr01015>dir
 Datenträger in Laufwerk E: ist PC316M_LW_E
 Volumeseriennummer: EAC6-EB6F

 Verzeichnis von E:\$daten\ebv\jpr01015

03.03.2016  14:23    <DIR>          .
03.03.2016  14:23    <DIR>          ..
03.03.2016  14:20    <DIR>          bsp01015
24.06.2015  20:39            58.062 de_eikelberg_iotools.jar
               1 Datei(en),         58.062 Bytes
               3 Verzeichnis(se),  8.746.741.760 Bytes frei

E:\$daten\ebv\jpr01015>dir bsp01015
 Datenträger in Laufwerk E: ist PC316M_LW_E
 Volumeseriennummer: EAC6-EB6F

 Verzeichnis von E:\$daten\ebv\jpr01015\bsp01015

03.03.2016  14:20    <DIR>          .
03.03.2016  14:20    <DIR>          ..
03.03.2016  14:22             1.517 Konsoladd.class
03.09.2006  14:50             1.817 Konsoladd.java
               2 Datei(en),          3.334 Bytes
               2 Verzeichnis(se),  8.746.741.760 Bytes frei

E:\$daten\ebv\jpr01015>d:\jdk1.8.0\bin\javac -cp .;de_eikelberg_iotools.jar bsp01015\Konsoladd.java

E:\$daten\ebv\jpr01015>d:\jdk1.8.0\bin\java  -cp .;de_eikelberg_iotools.jar bsp01015.Konsoladd
--------------------------------------------------------------------------
    Rechenprogramm zur Addition ganzer Zahlen (Konsoladd)
    Copyright 2006 by M. Eikelberg (www.eikelberg.de)
--------------------------------------------------------------------------

Geben Sie den ersten Summanden ein:
5

Geben Sie den zweiten Summanden ein:
6

Die Summe von 5 und 6 ist gleich 11

Programm beenden mit RETURN !
```

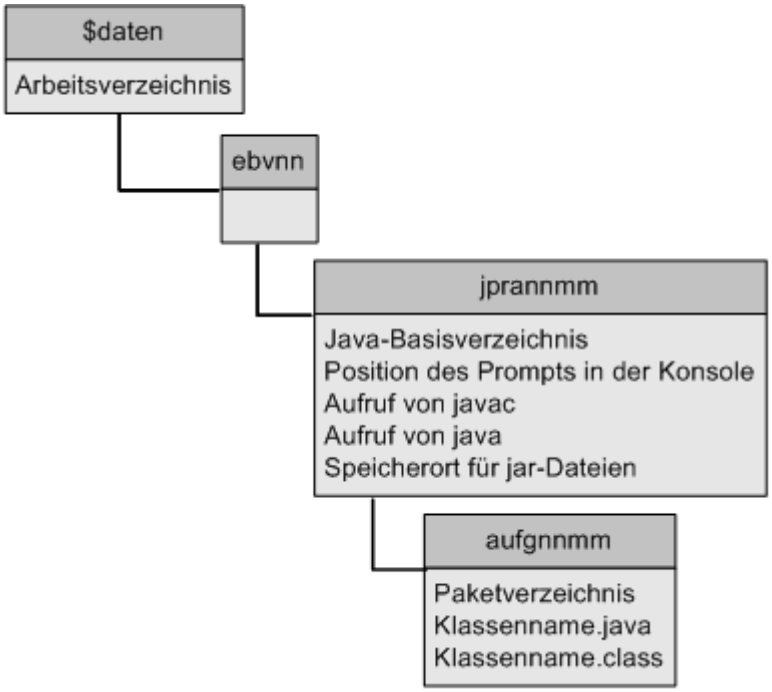
Compile

Programmstart

Hinweis: Für das Praktikum gelten folgende Namenskonventionen:

1. Das Basisverzeichnis zur Lösung der Übungsaufgabe *mm* zu Kapitel *nn* sollte stets `jpr`*nnmm* heißen.
2. Das Paketverzeichnis für die Java-Klassen der Lösung einer Übungsaufgabe mit Nummer *mm* und zu Kapitel *nn* sollte in der Regel mit dem Namen `aufg`*nnmm* benannt werden.

Für das Praktikum ergibt sich somit zunächst folgende Situation - dabei wird wieder in einem Verzeichnis `$daten` gearbeitet und es wird wieder in diesem Arbeitsbereich zunächst ein Unterordner `ebvnn` angelegt, in dem anschließend das Java-Basisverzeichnis angeordnet wird.



1.3.2 Eclipse

Das Programm **Eclipse** (Version 4.5.1) ist eine integrierte Entwicklungsumgebung für die Programmentwicklung mit dem JDK (standardmäßig JDK 1.8.0). Eclipse verwendet eine vom Benutzer einstellbare Version des JDK und verfügt über eine graphische Benutzeroberfläche. In der Grundversion bietet Eclipse neben vielen weiteren Funktionen bereits folgende Funktionalität:

- Graphische Benutzeroberfläche
- Verwaltung aller Dateien zu einem Software-Projekt in einer Projektumgebung.
- Hilfe bei der Erstellung der Dateistruktur zu einem Software-Projekt.
- Automatische Erstellung von Java-Quellcode für die Grundstruktur der verschiedenen Programmtypen.
- In die Oberfläche integrierter Editor.
- Zugriff auf Java Compiler und JVM über die Oberfläche von Eclipse.
- Markierung fehlerhafter Programmzeilen direkt im Editor bei Compile-Fehlern.
- Syntaxeinfärbung

Diese Funktionen erleichtern ganz wesentlich die Entwicklung und Weitergabe von Java Programmen. Sie sind typisch für integrierte Entwicklungsumgebungen und sehr hilfreich bei der Programmierung. Daher wird für die Durchführung dieses Java-Kurses die Entwicklungsumgebung Eclipse eingesetzt.

Eclipse ist ein Open Souce Programm und damit nicht kostenpflichtig. Es wird gemäß der **Eclipse Public License** lizensiert. Eclipse steht in der Tradition bewährter Entwicklungsumgebungen und wird von der Eclipse Organisation hergestellt und vertrieben. An der Eclipse-Organisation beteiligen sich Firmen wie Sun Microsystems, IBM und CodeGear/Borland. Da Eclipse mit verschiedenen Java-Versionen kombiniert werden kann und in vielfältiger Weise erweitert werden kann, stellt der Autor für die Arbeit mit diesem Java-Kurs speziell konfigurierte Eclipse-Versionen bereit. Die Verteilung dieser Versionen erfolgt über DVDs mit dem Titel "Software für den Unterricht Eikelberg". Diese Anleitung erkärt die Verwendung der Version "September 2016" dieser DVDs mit der Eclipse-Version 4.5.1 und der JDK-Version 1.8.0.

Hinweis: Beachten Sie die **zu anderen Entwicklungsumgebungen bereitgestellten Anleitungen im Anhang**!

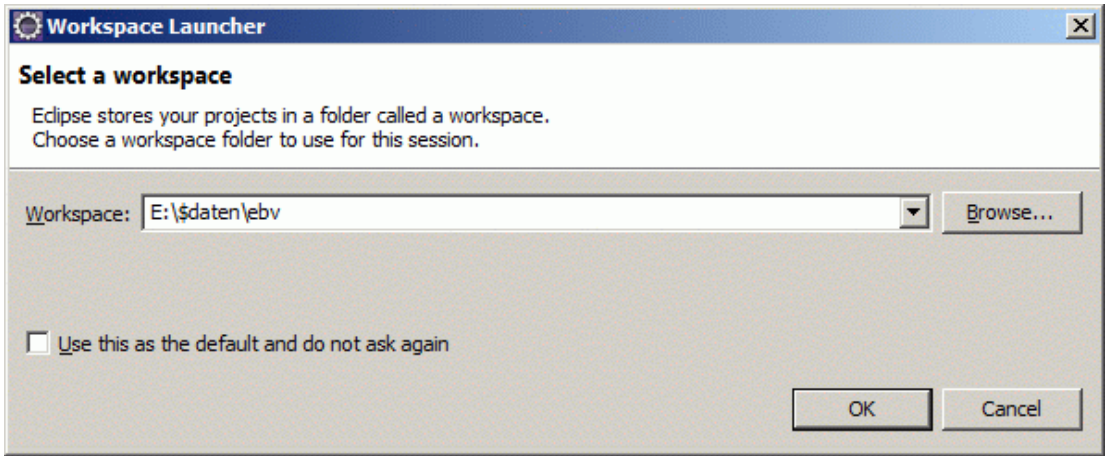
Auf der DVD mit der Software für den Unterricht Eikelberg Stand "September 2016" befinden sich 32 bit Programmversionen. Diese sind lauffähig unter Windows 7 bzw. 10 (32bit sowie 64bit).

1. **Start von Eclipse 4.5.1.**

   Alle Eclipse 4.5.1 Projekte müssen in einem gemeinsamen Verzeichnis angelegt werden:

   - Dieses Verzeichnis wird im Weiteren **Eclipse Basisverzeichnis** genannt.
   - Für dieses Verzeichnis wird in dieser Anleitung der Verzeichnisname "ebv" (**E**clipse **B**asisverzeichnis) gewählt. ***Man beachte, dass für die Beispiele und Musterlösungen dieses Kurses jedoch kapitelweise Eclipse Basisverzeichnisse mit dem Namen ebvnn angelegt wurden!***

   Im Eclipse Basisverzeichnis wird von Eclipse 4.5.1 projektübergreifend ein Verzeichnis mit den Namen ".metadata" angelegt. Ferner wird in diesem Verzeichnis für jedes Programmierprojekt ein Unterverzeichnis angelegt, das die Funktion des **Java-Basisverzeichnisses** übernimmt. Beim Start von Eclipse 4.5.1 wird standardmäßig zunächst der Name des Eclipse Basisverzeichnisses abgefragt. Dieser muss in folgendem Fenster eingetragen werden:



   Es startet Eclipse Version 4.5.1, auch Version "Mars.1" genannt.

   Schließen sie ggf. nach Abschluss des Programmstarts von Eclipse 4.5.1 die Registerkarte "Welcome".

   Hinweis: Falls beim Start von Eclipse das obige Fenster nicht erscheint, können Sie wie folgt in Eclipse konfigurieren, dass dieses Fenster beim Start wieder erscheint: Wählen Sie im Menü von Eclipse Window / Preferences, erweitern Sie im Fenster Preferences

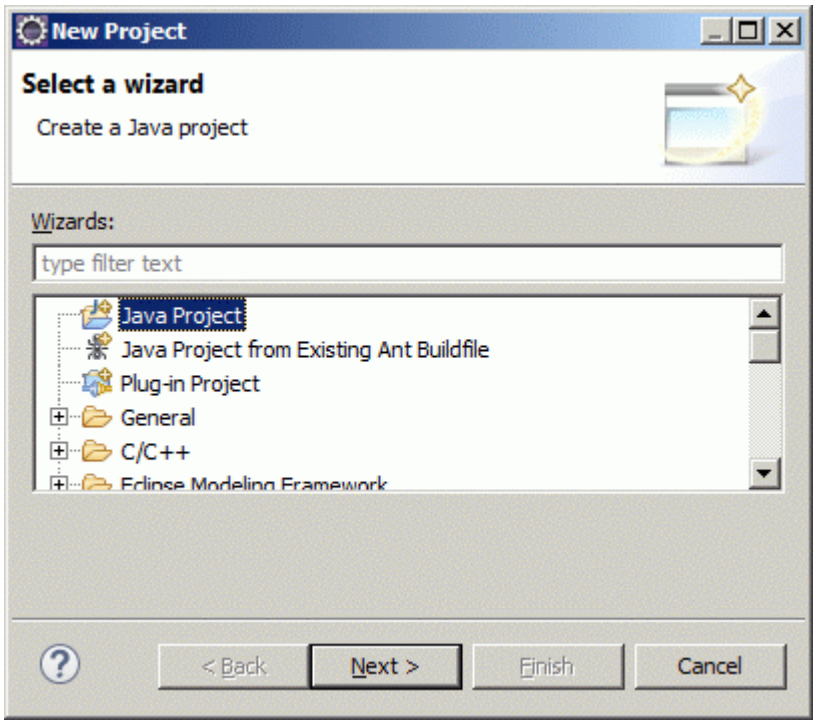
den Zweig General sowie den Zweig "Startup and Shudown" und markieren Sie das Blatt "Workspaces". Aktivieren Sie anschließend im rechten Bereich des Fensters die Option "Prompt for workspace on startup".

Mit dem beschriebenen Ablauf können Sie einen neuen Workspace anlegen oder mit einem bereits existierenden Workspace weiter arbeiten. Wenn Sie einen neuen Workspace angelegt haben, sollten Sie folgende Konfigurationseinstellungen vornehmen:

- Öffnen Sie im Eclipse-Menü Window / Preferences und erweitern Sie den Zweig Java / Installed JREs. Es sollte eine installierte Version des JRE angezeigt werden mit der Pfadangabe ("Location")
  `C:\eclipse-eikelberg-5` bzw. `D:\eclipse-eikelberg-5`
  je nachdem, in welches Laufwerk Ihre Eclipse-Version installiert wurde. Korrigieren Sie ggf. den vorhandenen Eintrag, in dem Sie den vorhandenen Eintrag markieren, auf den Button "Edit" klicken und die Eigenschaften des Eintrags geeignet ändern.
- Öffnen Sie im Eclipse-Menü Window / Preferences und erweitern Sie den Zweig General / Editors / Text Editors / Spelling und deaktivieren Sie die Option "enable spell checking". Hiermit deaktivieren Sie die (englische) Rechtschreibprüfung, die für Programme mit deutschen Kommentaren und deutschen Programmausgaben natürlich keine sinnvollen Ergebnisse ergibt.
- Öffnen Sie im Eclipse-Menü Window / Preferences und erweitern Sie den Zweig Java / Code Style / Formatter. Klicken Sie im rechten Bereich des Fensters auf den Button "Edit" und öffnen Sie so das Fenster "Profile ...". Erstellen Sie wie folgt ein Profil mit individuell eingestellten Formatierungsregeln für den Quellcode:
  - Stellen Sie im Fenster "Profile ..." in der Registerkarte "Indentation" die Tab policy um auf "Spaces Only" und tragen Sie in den Feldern "Indentation size" und "Tab size" jeweils den Wert 2 ein. Tragen Sie im Feld "Profile name" den Namen "eclipse-javakurs" ein und speichern Sie die gewählten Einstellungen mit Klick auf den Button "Apply".
  - Wechseln Sie nun in die Registerkarte "Braces". Ändern Sie die "Brace positions" für alle angebotenen Möglichkeiten um von "Same line" auf "Next line" und bestätigen Sie diese Änderungen mit Klick auf den Button "Apply".
  - Wechseln Sie in die Registerkarte "Comments" und wählen Sie folgende Optionen aus: "Perserve white space between code and line comments" / "Never indent line comments on first column" / "Never indent block comments on first column" / "Never join lines"
  - Wechseln Sie in die Registerkarte "Line Wrapping" und wählen Sie die Option "Never join already wrapped lines" aus. Markieren Sie im Zweig "Function Calls" das Blatt "Arguments" und stellen Sie die "Line wrapping policy" "Wrap all elements, every element on a new line" ein. Konfigurieren Sie die "Indentation policy" "Indent by one".
  - Wechseln Sie in die Registerkarte "Control Statements". Deaktivieren Sie die Option "Keep else if in one line". Aktivieren Sie im Feld "General" alle Optionen "Insert new line...".
  - Übernehmen Sie die Einstellungen mit Klick auf den Button "Apply". Schließen Sie danach das Fenster "Profile 'eclipse-javakurs'" mit Klick auf den Button OK.
- Schließen Sie anschließend das Fenster "Properties" ebenfalls durch Klick auf den Button OK.

2. **Anlegen eines Programmierprojektes**

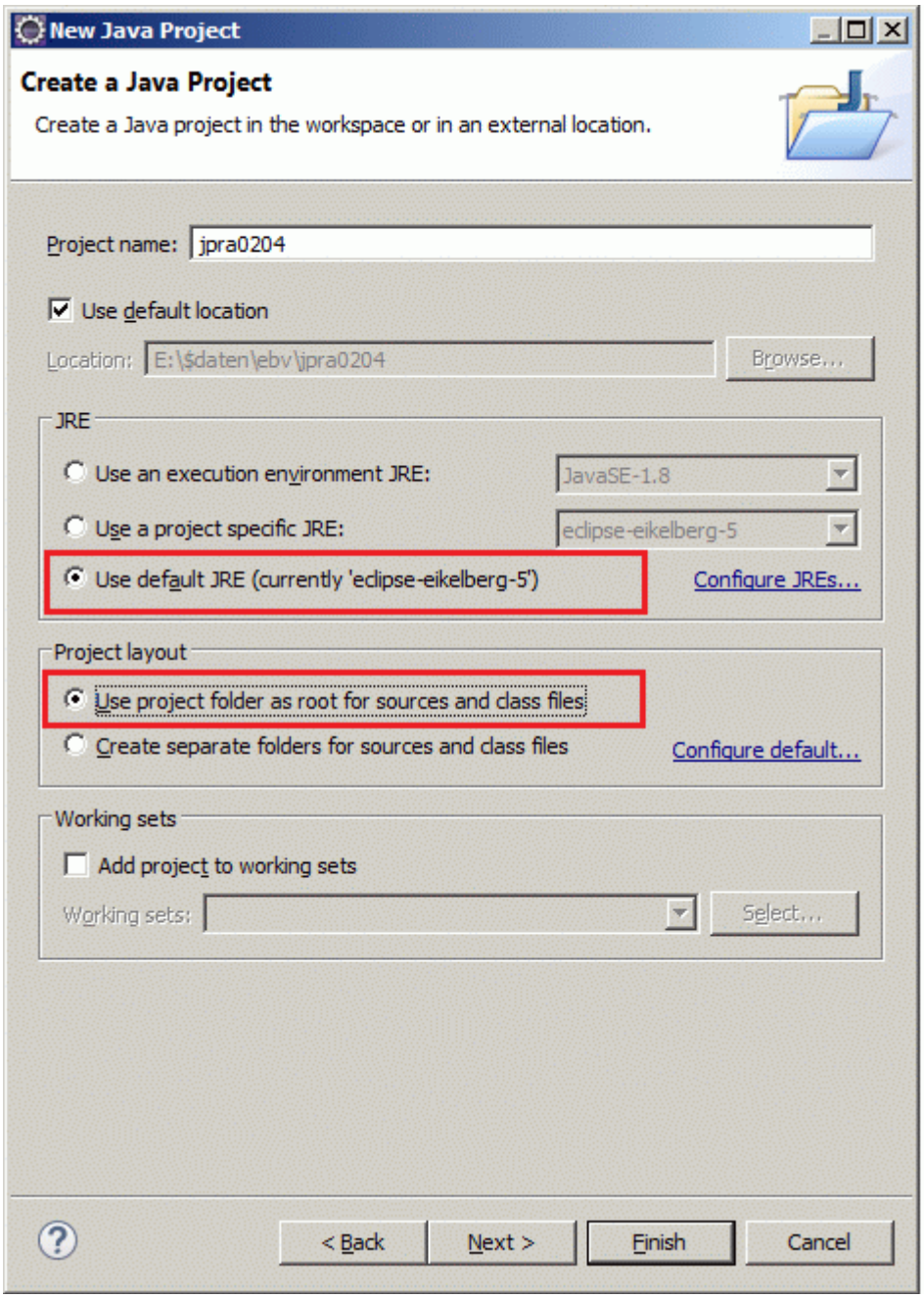
Wählen Sie im "File"-Menü die Auswahl "New" / "Project". Wählen Sie anschließend im Fenster "New Project" den Assistenten zum Anlegen eines Java-Projektes aus:



Wechseln Sie mit Klick auf den Button "Next" in das Fenster "New Java-Project". Tragen Sie dort zunächst den Namen des Basisverzeichnisses ein - dieser Name wird für die Aufgabe 4 in Kapitel 2 wie folgt gebildet:

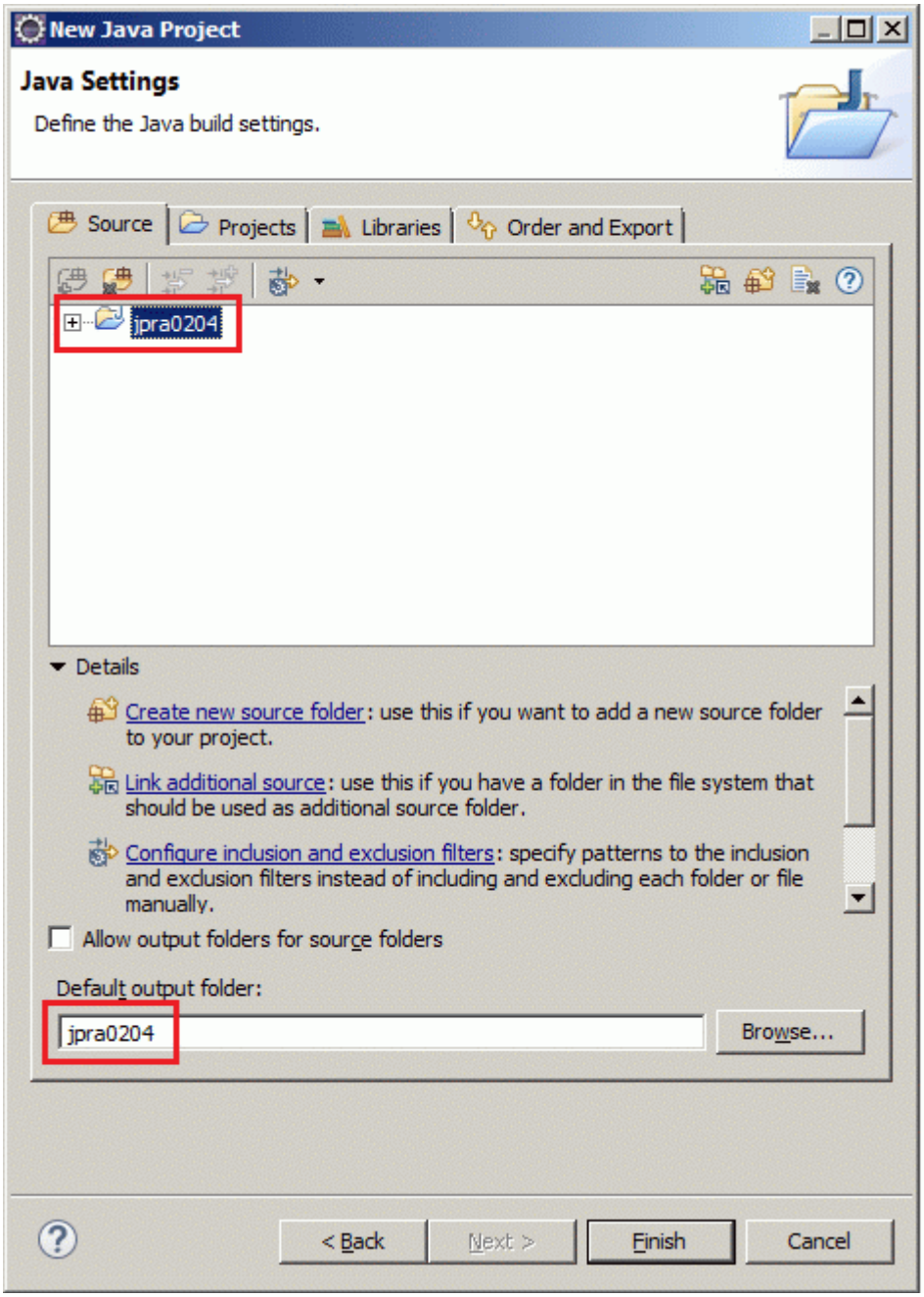
Feld	Eintrag für eine Lösung der Programmieraufgabe *mm* in Kapitel *nn*	Beispiel
Projektname	jpra*nnmm*	jpra0204
Verzeichnisname	volle Pfadangabe für das Java-Basisverzeichnis	e:\$daten\ebv\jpra0204

Im Fenster "New Java-Project" werden dann die folgenden Projekteinstellungen angezeigt:



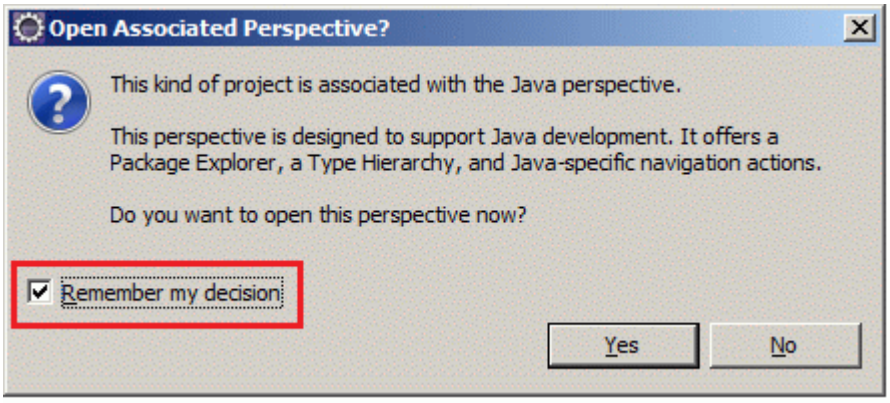
Achten Sie darauf, dass die Option "Use project folder as root for sources and class files" eingestellt ist. Ferner sollte Option "Use default JRE ..." ausgewählt werden. Wählen Sie ggf. diese Optionen aus!

Bestätigen Sie diese Konfiguration mit "Next" und wechseln Sie so in das Fenster "Java Settings". In diesem Fenster sollten folgende Einstellungen angezeigt werden:

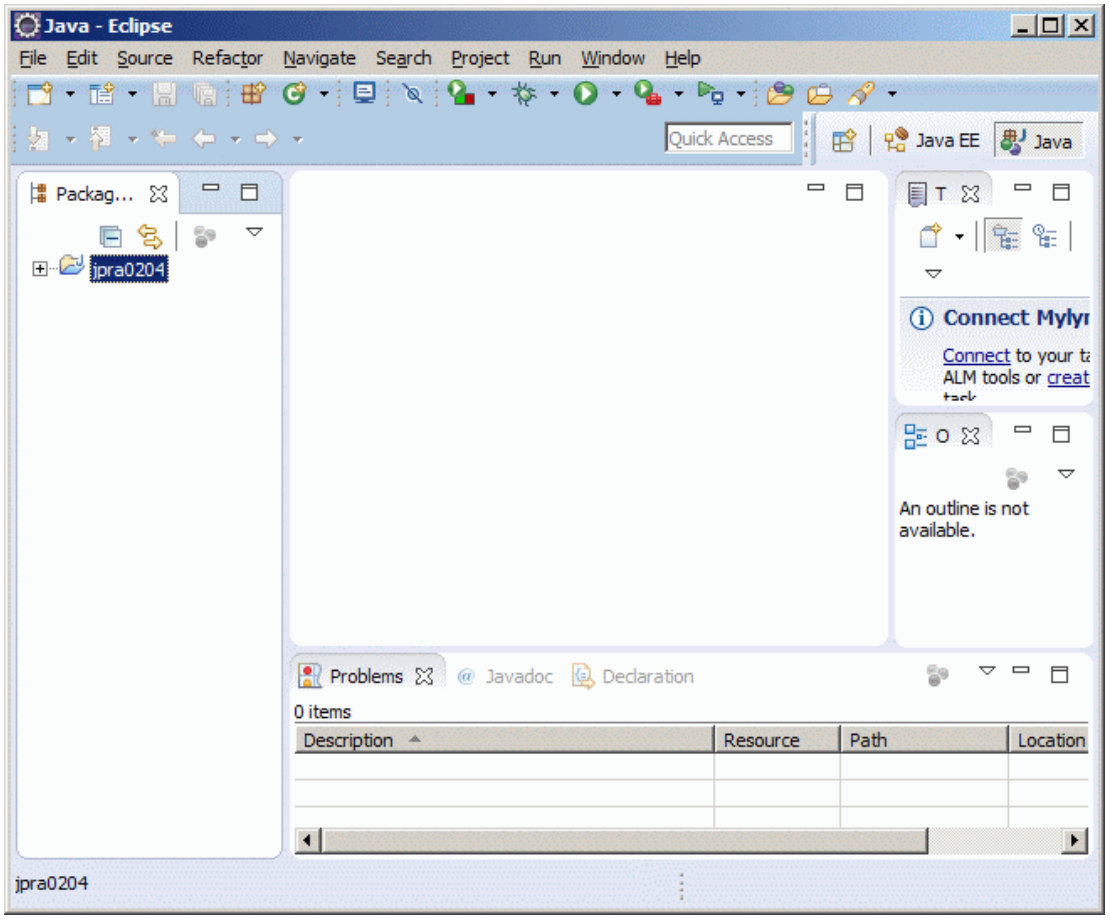


Achten Sie in diesem Fenster darauf, dass in diesem Fenster als souce folder sowie als default output folder jeweils der Ordner für Ihr Projekt angezeigt wird. Ist dies nicht der Fall, müssen Sie mit dem Button "Back" in das vorhergehende Fenster zurück wechseln und die Einstellung der Option "Use project folder as root for sources and class files" vornehmen.

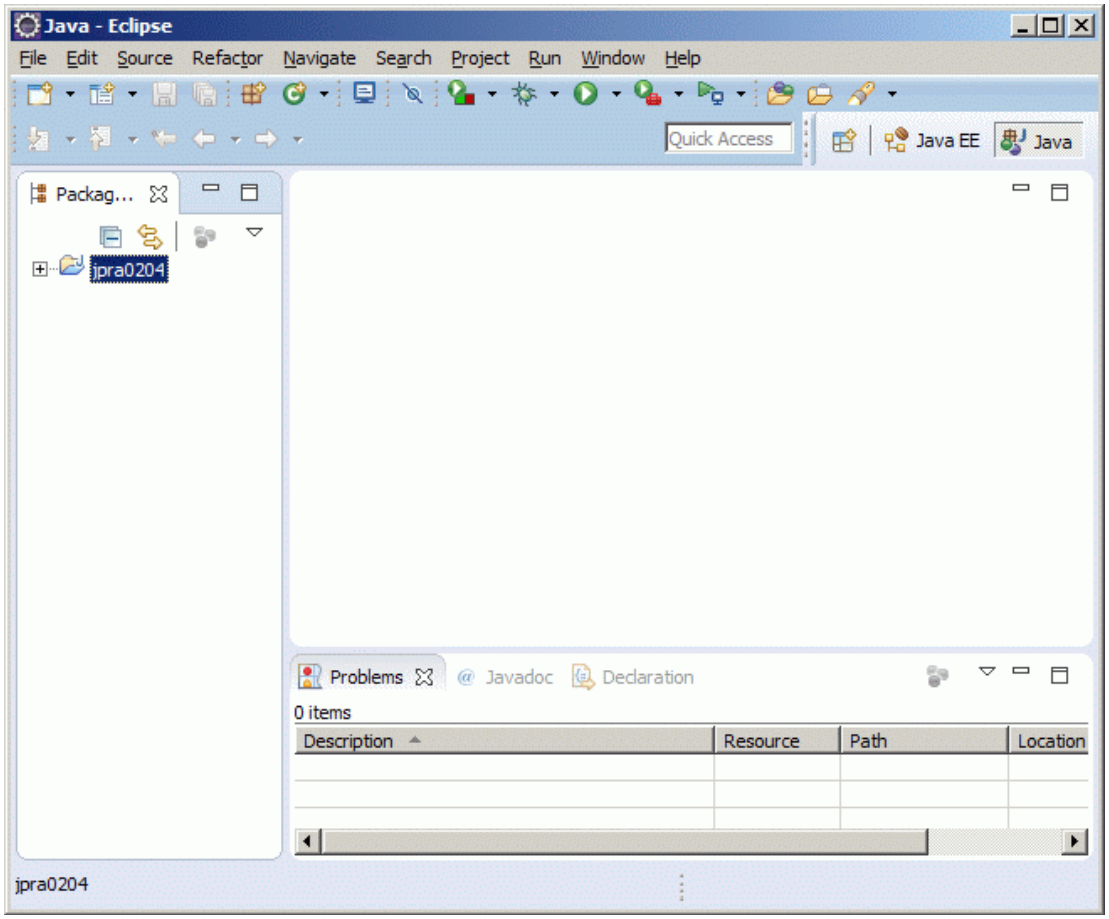
Mit den jetzt vorhandenen Einstellungen wird erreicht, dass Ihre Projektkonfiguration in Bezug auf die Ordnerstruktur den Standardeinstellungen des JDK entspricht. Beenden Sie den Assistenten mit Klick auf den Button "Finish". Gegebenenfalls erscheint die folgende Meldung, die Sie mit "Yes" bestätigen sollten und für die Sie die Option "remember my decision" einstellen können:



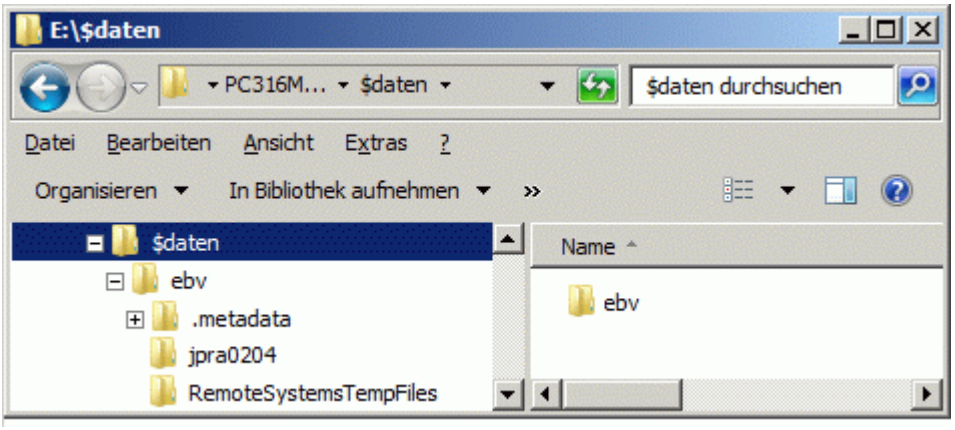
Ausgehend von den oben eingegebenen Informationen legt Eclipse 4.5.1 jetzt das Projekt an und zeigt dieses in einer Java-Perspektive an:



Eine Eclipse Perspektive ist eine Anordnung von Fensterelementen in Eclipse, die für einen bestimmten Zweck konfiguriert wurde. Eine Java-Perspektive ist also eine Fensterkonfiguration, die bei der Java-Programmierung verwendet wird. Eclipse kennt viele weitere Perspektiven, z. Bsp. die C/C++ Perspektive und die Debug Perspektive. In der vorliegenden Perspektive können die Registerkarten "Task List" und "Outline" geschlossen werden, so dass sich der Anzeigebereich für den Quellcode vergrößert:

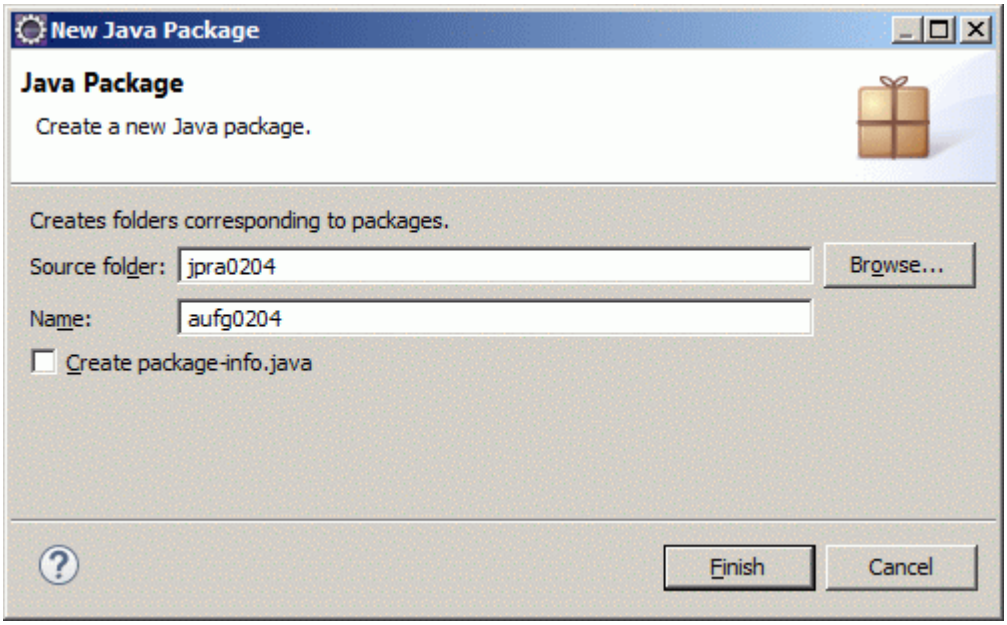


Nach Anlegen des Projektes ist im Eclipse Basisverzeichnis folgende Struktur entstanden:

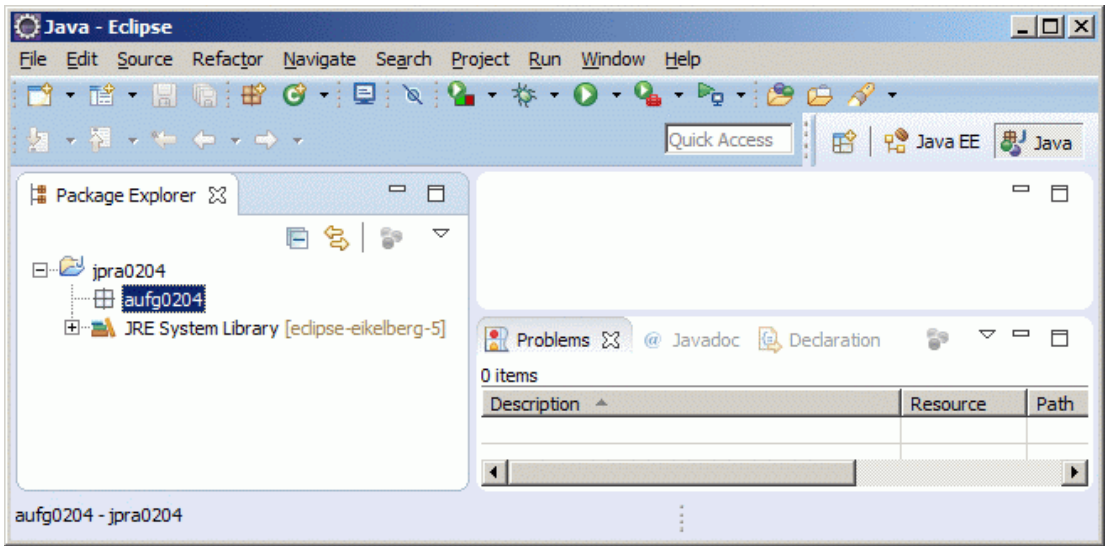


3. **Anlegen einer Quelldatei für eine Java-Klasse mit main-Methode (Hauptprogramm)**

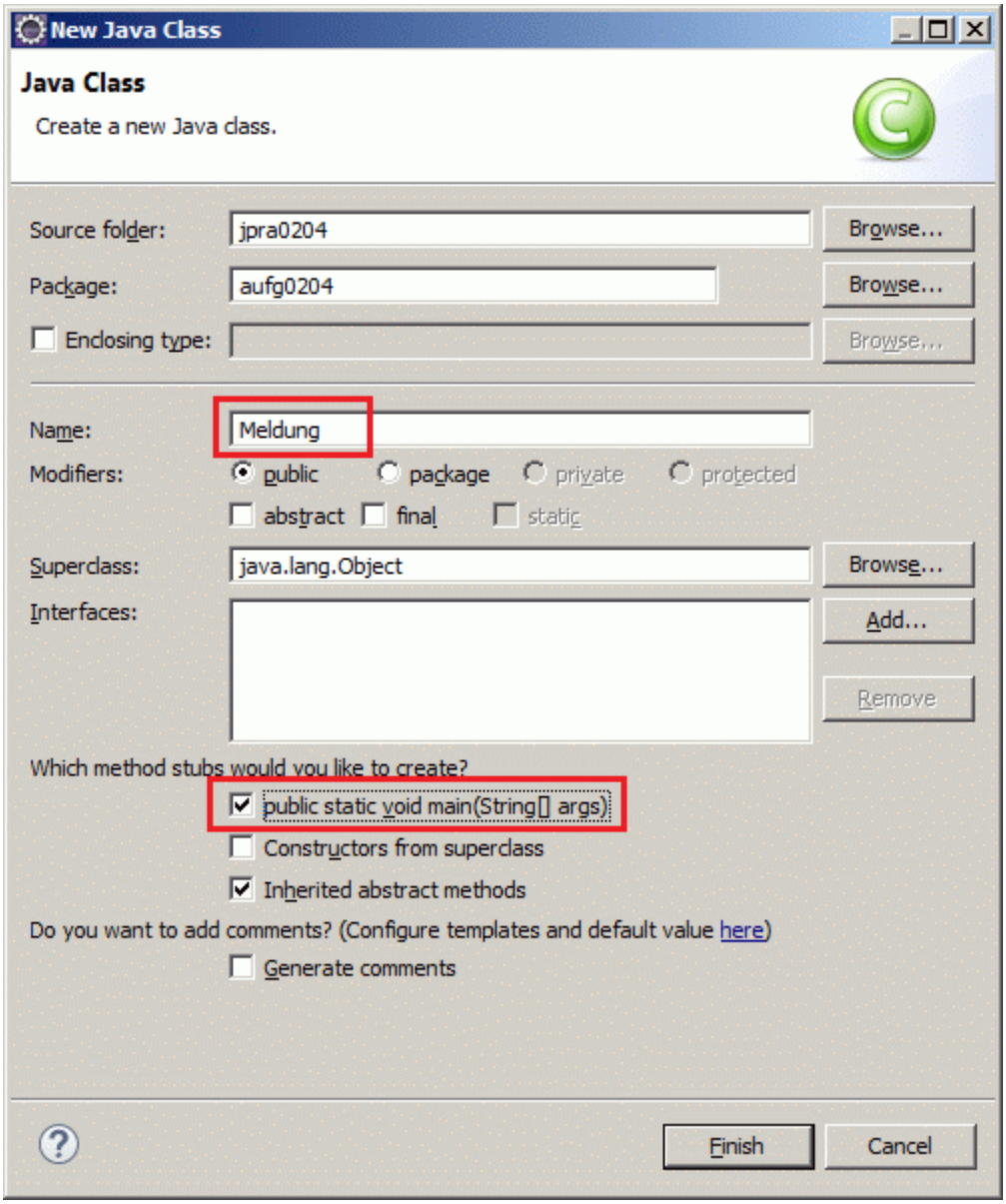
Markieren Sie im linken Bereich des Fensters im Package Explorer das Projekt jpra0204 und wählen Sie anschließend im "File"-Menü die Auswahl "New" / "Package". Es öffnet sich das Fenster "New Java-Package". Geben Sie in diesem Fenster im Feld "Name" den Paket-Namen Ihrer Java-Anwendung ein und klicken Sie auf den Button "Finish":



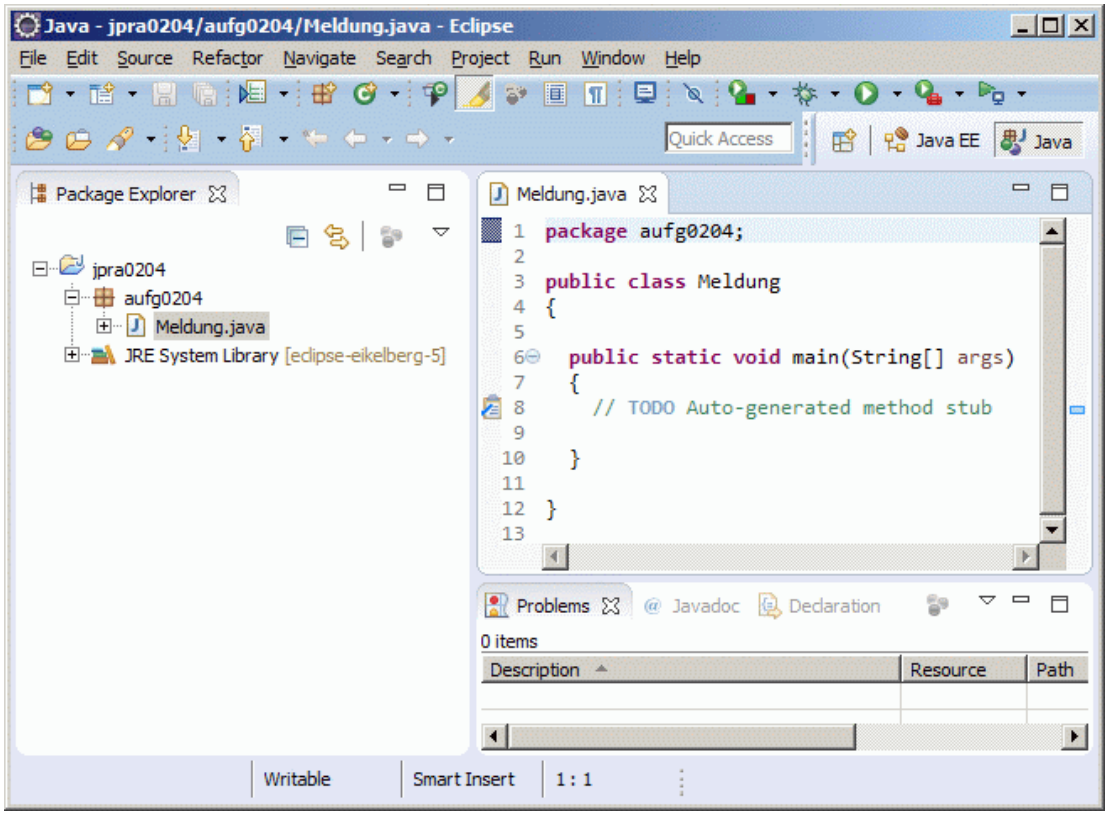
Im Eclipse wird das neu angelegte Paket wie folgt angezeigt:



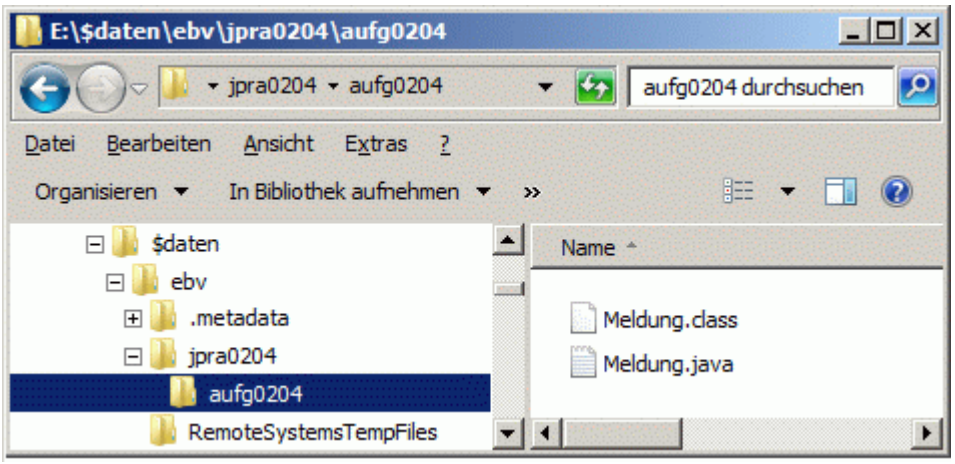
Markieren Sie im linken Bereich des Fensters im Package Explorer das Paket aufg0204 und wählen Sie anschließend im "File"-Menü die Auswahl "New" / "Class". Es öffnet sich das Fenster "New Java Class". Kontrollieren Sie, ob der Name des Java-Basisverzeichnisses (jpra0204) sowie der Paketname (aufg0204) korrekt eingetragen sind. Geben Sie den Namen der Klasse (Meldung) im Feld "Name" ein und wählen Sie aus, dass eine main-Methode erzeugt werden soll:



Bestätigen Sie Ihre Eingaben durch Klick auf den Button "Finish". Daraufhin wird die angeforderte Klasse erzeugt und in der Java-Perspektive angezeigt:



Im Eclipse Basisverzeichnis ist jetzt folgende Dateistruktur vorhanden:

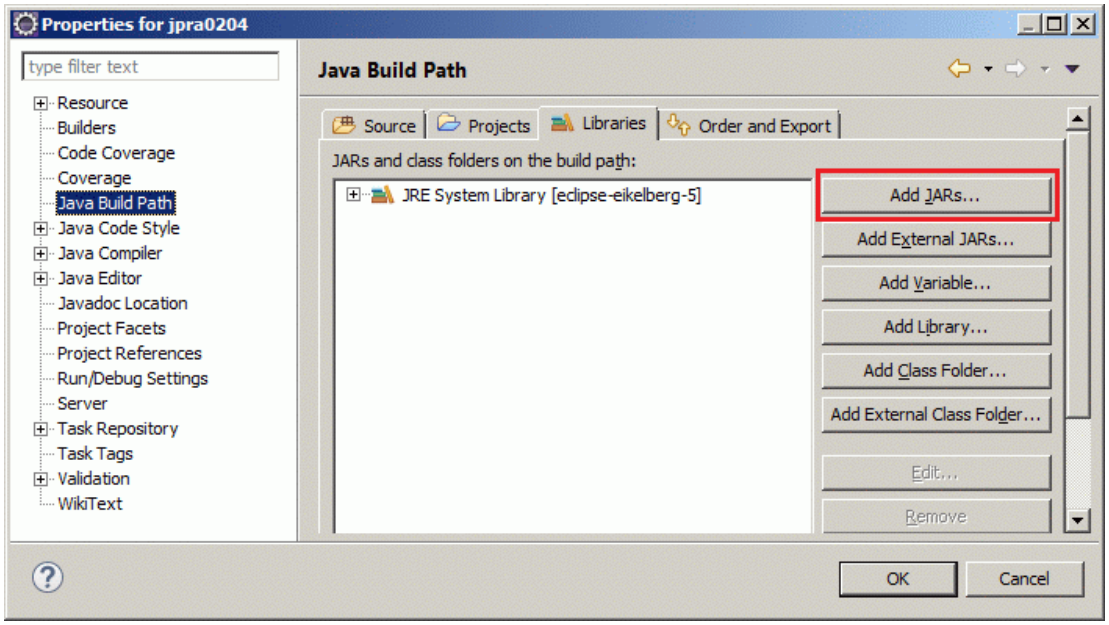


4. **Einfügen des Paketes de.eikelberg.iotools in das Projekt**

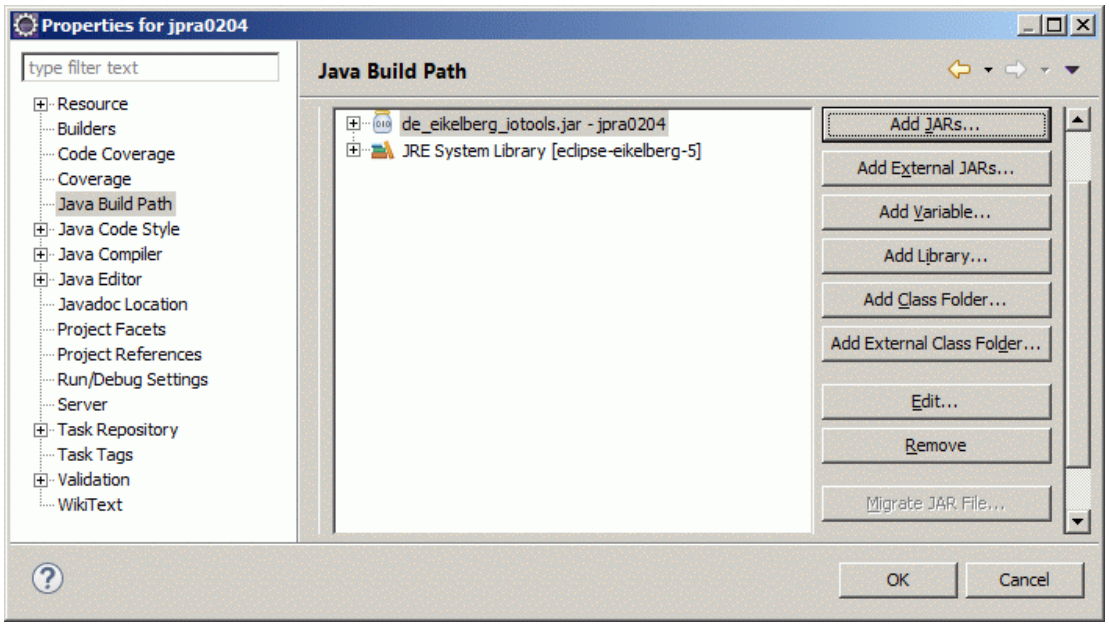
Für die Arbeit mit Eclipse 4.5.1 muss das Paket de.eikelberg.iotools als sogenannte JAR-Datei in das Java Basisverzeichnis eingefügt werden. Das Paket de.eikelberg.iotools liegt als JAR-Datei mit dem Namen de_eikelberg_iotools.jar vor. Laden Sie das **ZIP-Archiv mit der Datei de_eikelberg_iotools.jar** herunter und speichern Sie das Archiv im Java-Basisverzeichnis (also im Beispiel im Verzeichnis E:\$daten\ebv\jpra0204). *Die Datei de_eikelberg_iotools.jar darf nicht weiter bearbeitet bzw. muss auch nicht entpackt werden.*

> Da die jar-Datei de_eikelberg_iotools.jar nicht von Eclipse in das Java-Basisverzeichnis eingefügt wurde, ist diese Einfügung Eclipse nicht bekannt. Sie müssen daher an dieser Stelle das Java-Projekt im Package Explorer von Eclipse markieren, in das Sie die jar-Datei eingefügt haben. Öffnen Sie anschließend mit der rechten Maustaste das Kontextmenü für das Projekt und wählen Sie im Kontextmenü den Eintrag Refresh (alternativ Taste F5). Daraufhin sollte die jar-Datei im Package Explorer von Eclipse im entsprechenden Projekt angezeigt werden.

Markieren Sie in der Registerkarte "Package-Explorer" der Java-Perspektive das Projekt jpra0204 und wählen Sie im "Project"-Menü den Menüpunkt "Properties". Es öffnet sich das Fenster "Properties for jpra0204". Markieren Sie im linken Bereich des Fensters den Eintrag "Java Build Path" und wechseln Sie im rechten Bereich des Fensters in die Registerkarte "Libraries":



Klicken Sie auf den Button "Add JARs" und fügen Sie die im Java-Basisverzeichnis gespeicherte jar-Datei de_eikelberg_iotools.jar hinzu. Verwenden Sie *nicht* den Button "Add External JARs"! Die jar-Datei wird daraufhin in der Registerkarte "Libraries" angezeigt:



Bestätigen Sie die Eingaben mit OK und wechseln Sie so wieder in die Java-Perspektive zurück.

Tipp: Das Kennzeichnen einer jar-Datei als Bibliothek kann auch über den Eclipse Package Explorer, Kontextmenü zur im Package Explorer angezeigten Jar-Datei, Menüauswahl Build Path/ Add to Build Path erfolgen.

5. **Codierung des Java-Programms**

Codieren Sie jetzt im mittleren Bereich der Java-Perspektive Ihr Java-Programm und vergessen Sie nicht, Ihre Quelldatei regelmäßig zu sichern. Im vorliegenden Beispiel - Lösung von Aufgabe 4 aus Kapitel 2 des Java-Kurses - entsteht nach Einfügen des in der Aufgabenstellung genannten Java-Quellcodes sowie nach geeigneter Änderung der Anordnung der Klammern folgendes Java-Programm:

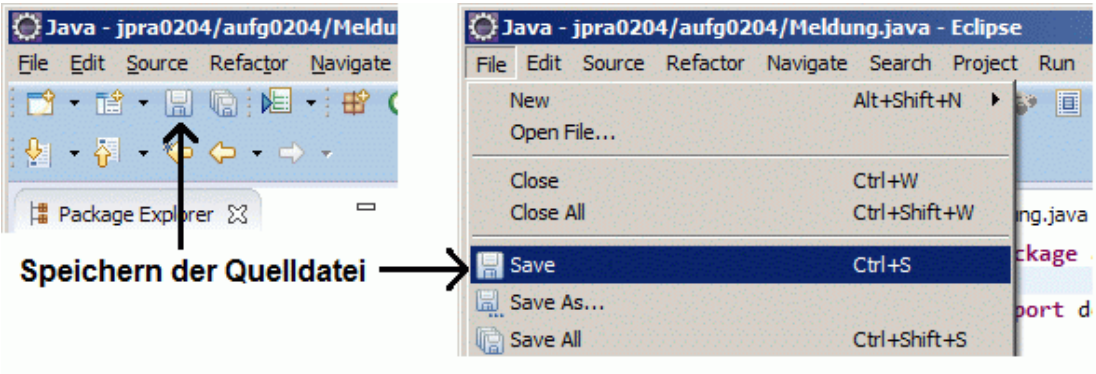
```
Meldung.java
1  package aufg0204;
2
3  import de.eikelberg.iotools.*;
4
5  public class Meldung
6  {
7
8    public static void main(String[] args)
9    {
10     ScreenOut.kPrintln();
11     ScreenOut.kPrintln("Hier spricht das Programm aufg0204.Meldung!");
12   }
13 }
```

Hinweis: Wenn Sie später Ihr Programm in der Eingabeaufforderung von Windows mit der Java Virtuellen Maschine (JVM) ausführen wollen, müssen Sie beim Aufruf der JVM wie folgt als Suchpfad das Java-Basisverzeichnis sowie den Namen der JAR-Datei zusätzlich mit angegeben:

```
Administrator: Eingabeaufforderung
E:\$daten\ebv\jpra0204>d:\jdk1.8.0\bin\java -cp .;de_eikelberg_iotools.jar aufg0204.Meldung
Hier spricht das Programm aufg0204.Meldung!
E:\$daten\ebv\jpra0204>
```

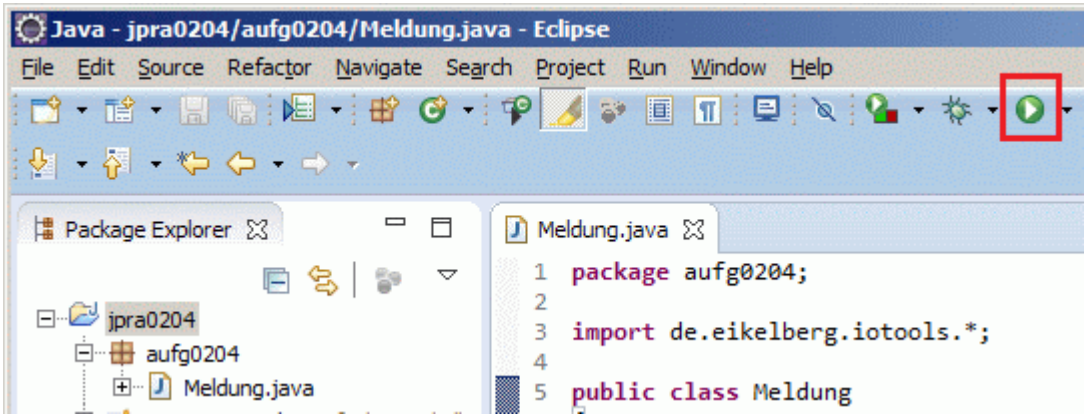
6. **Compile**

Eine Besonderheit von Eclipse ist, dass der Compiler nicht direkt gestartet werden kann. Vielmehr wird der Compiler stets gestartet, wenn die Quelldatei gespeichert wird (Strg+S bzw. Button "Save" bzw. Menüauswahl "File / Save":



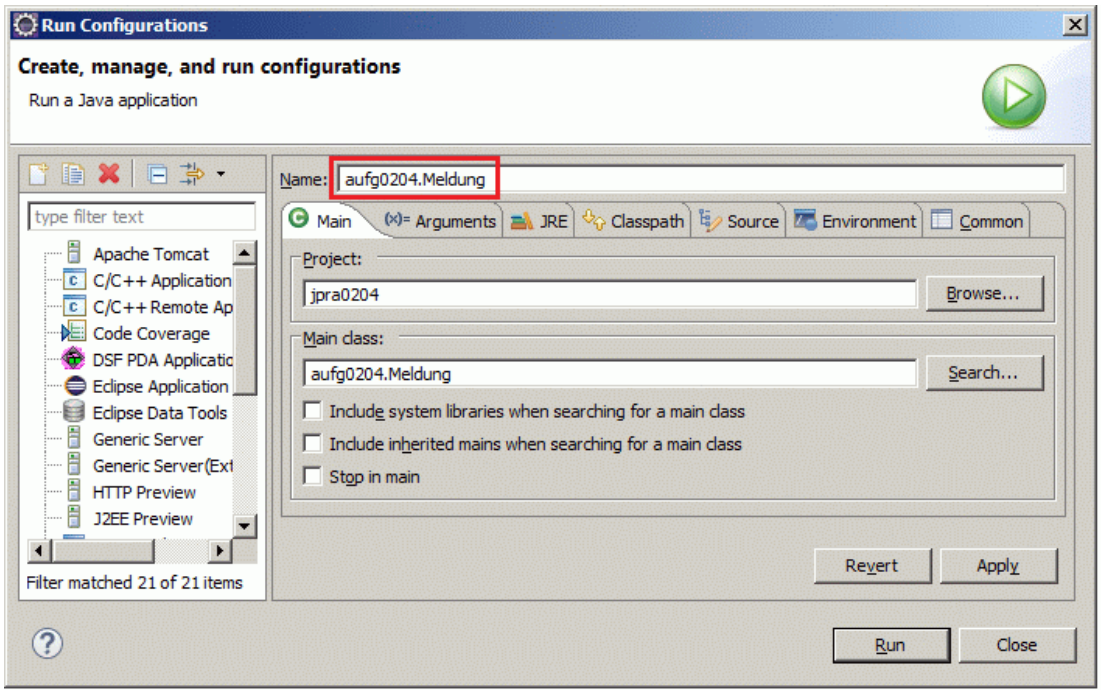
7. **Ausführen des Programms**

Die Ausführung des Programms erfolgt nach Klick auf den Button "Run".

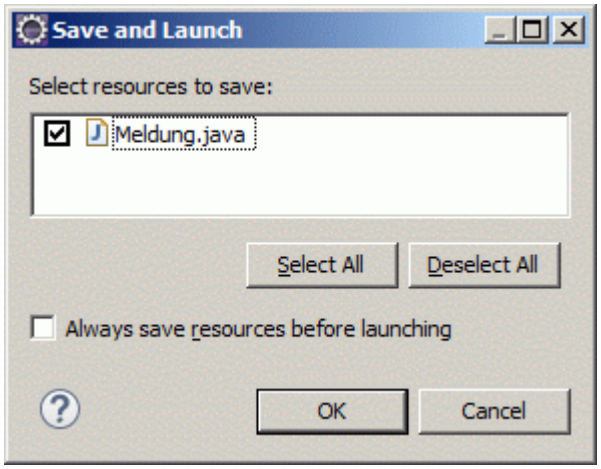


Daraufhin wird eine Lauf-Konfiguration ("Run Configuration") im Hintergrund erzeugt und das Programm mit Hilfe dieser Lauf-

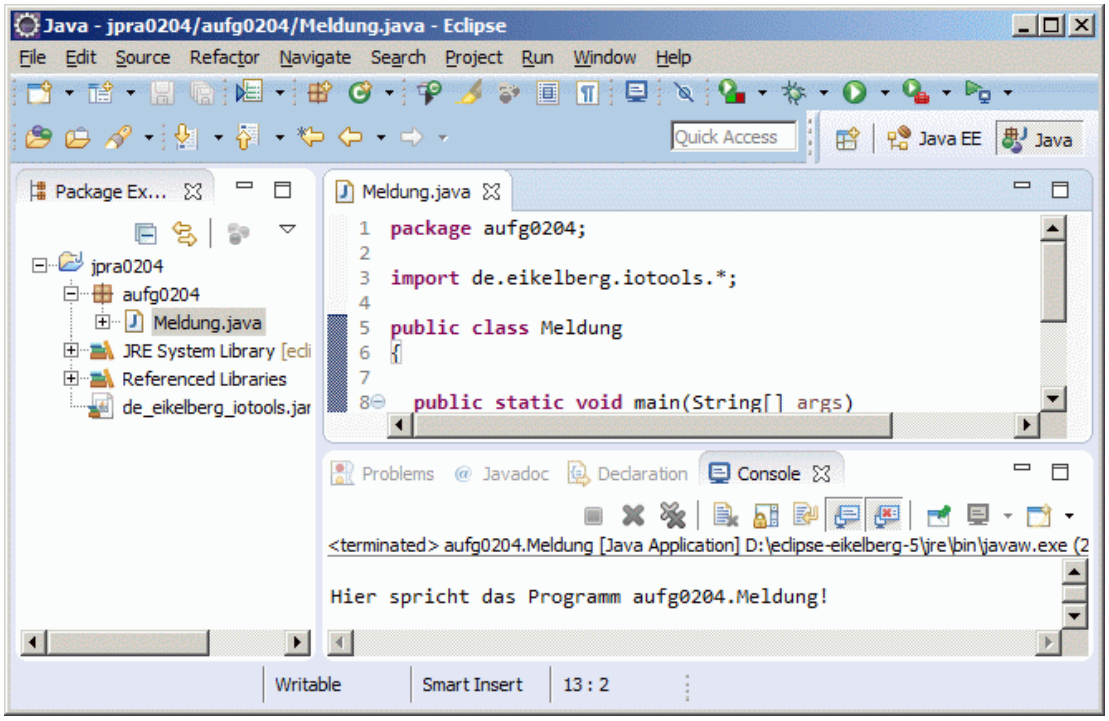
Konfiguration gestartet. Sie können diese Lauf-Konfiguration anpassen nach Klick auf den schwarzen Pfeil direkt neben dem grünen Run-Pfeil und Auswahl des Menüpunktes "Run Configurations". Im Fenster "Run" können Sie dann zum Beispiel wie folgt den Namen der Lauf-Konfiguration um den Paketnamen ergänzen:



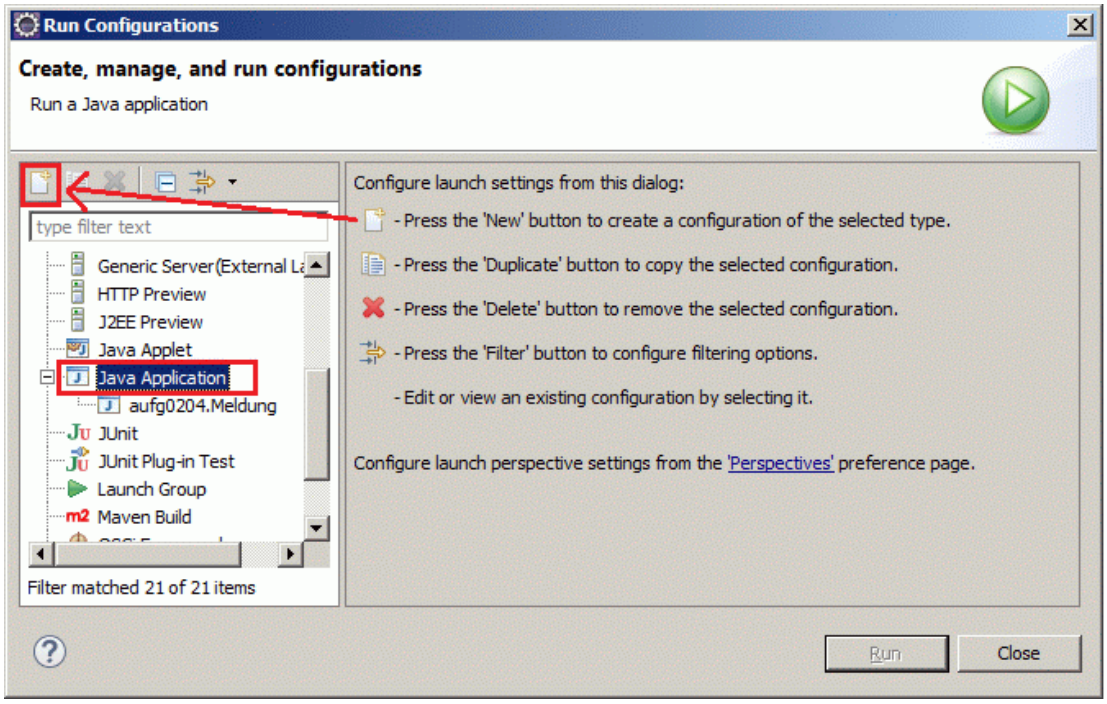
Aktivieren Sie Ihre Einträge durch Klick auf den Button Apply. Mit Klick auf den Button "Run" wird aus dem Fenster "Run" die Ausführung des Java-Programms gestartet. Falls sich der Quelltext des Programms nach dem letzten Speichern geändert hat, wird vor der Ausführung des Programms noch einmal die Speicherung der Quelldatei (und damit auch ein erneuter Compile) angestoßen:



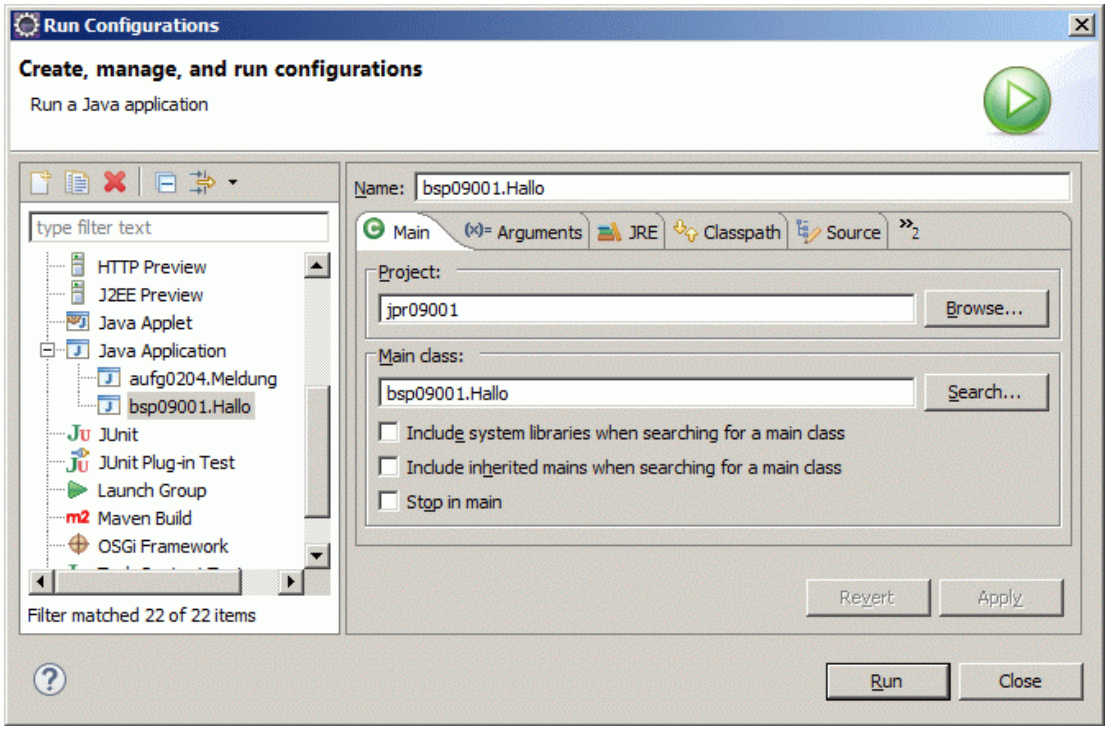
Im unteren Bereich der Java-Perspektive erscheint eine Registerkarte "Console", in der die Konsolaus- und eingabe des Programms erfolgt:



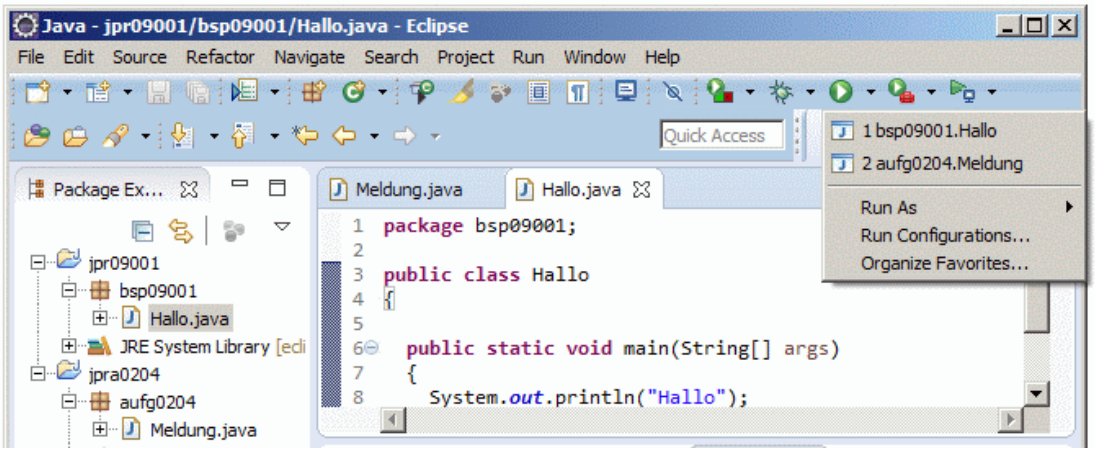
Hinweis: Werden mehrere Projekte im Eclipse 4.5.1 angezeigt, muss jedes auszuführende Programm in Eclipse 4.5.1 eingetragen werden. Dies geschieht nach Auswahl des Menüpunktes "Run..." im Menü "Run". Sie gelangen dann in das Fenster "Run" (s.o.).



Nach Markieren des Eintrags "Java-Application" im linken Bereich des Fensters und nach Klick auf die Schaltfläche "New" (s.o.) kann dann eine weitere main-Methode bzw. eine weitere Java-Anwendung eingetragen werden:



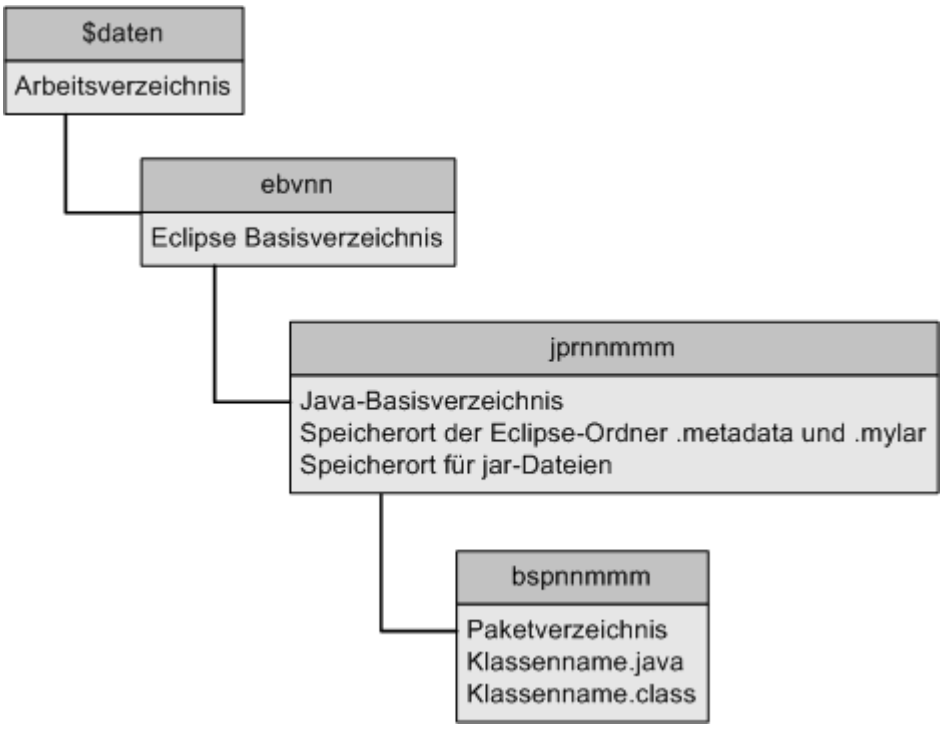
Öffnet man das Menü zum Button "Run", werden dann alle eingetragenen Java-Anwendungen zur Auswahl angeboten:



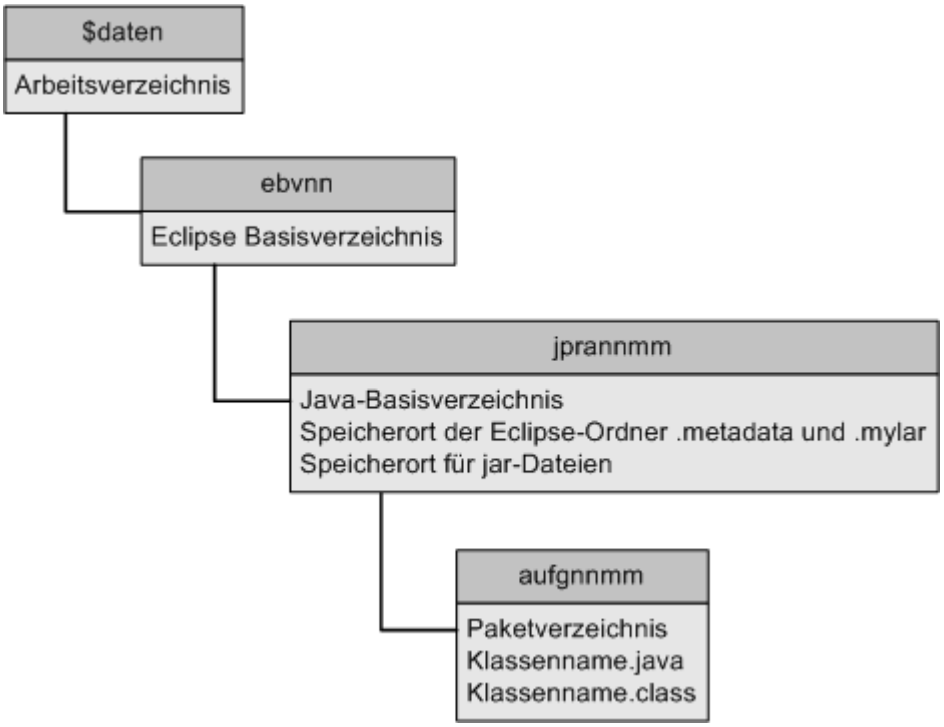
8. **Schließen und erneutes Öffnen eines Projektes**

   Das Schließen und Öffnen eines Projektes im Eclipse geschieht mit dem Kontextmenü des jeweiligen Projektes (Menüpunkte "Close Project" bzw. "Open Project "). Wird das zu öffnende Projekt nicht im Eclipse 4.5.1 angezeigt wird, kann es mit dem Menüpunkt "Import..." des "File"-Menüs importiert werden, was dann auch zur Anzeige des Projektes führt.

Die folgende Grafik fasst noch einmal die wesentlichen bei der Arbeit mit Eclipse zu beachtenden Punkte bezüglich Dateistruktur und Namenskonvention zusammen. Dabei ist zunächst die **Namensstruktur für die Beispiele** berücksichtigt. Man beachte, dass für die Beispiele zu diesem Kurs für jedes Kapitel ein eigenes Eclipse-Basisverzeichnis gewählt wurde. Statt des Namens **ebv** wurde für die Beispiele zu Kapitel ***nn*** der Verzeichnisname **ebv*nn*** gewählt. Damit ergibt sich für die Beispiele zu einem Kapitel folgende Struktur:



Für das Praktikum ist es ebenfalls empfehlenswert, für jedes Kapitel ein eigenes Eclipse-Basisverzeichnis anzulegen und dieses ebenfalls mit dem Namen **ebv*nn*** zu benennen. Insgesamt muss für das Praktikum entsprechend folgende Situation erreicht werden:



Zusammenfassung und Rezept: Der Ablauf zum Anlegen eines Eclipse-Projektes und zum Anlegen einer Java-Quelldatei geschieht (für die Lösung der Aufgaben zu Kapitel 1 und 2) in folgenden Schritten:

1. **Start der Entwicklungsumgebung** Eclipse
2. **Anlegen eines Eclipse Projektes**
3. **Anlegen einer Quelldatei** für eine Java-Klasse mit main-Methode (Hauptprogramm)
4. **Einfügen von Paketen** in jar-Dateien (z. Bsp. de.eikelberg.iotools) in das Projekt
5. **Codierung des Java-Programms**
6. **Speicherung der Quelldateien und Compile**
7. **Ausführen** des Programms
8. **Speichern der Dateien** des Eclipse-Projektes und **Schließen des Projektes**

13 Programmierung grafischer Benutzeroberflächen mit Eclipse WindowBuilder Editor

13.1 Programmierung eines Fensters

13.1.1 Ziel von Kapitel 13

Ziel von Kapitel 13 ist es, alternativ zu Kapitel 3 eine Methode vorzustellen, die die Programmierung von grafischen Benutzeroberflächen mit Hilfe eines sogenannten GUI-Builders ermöglicht. Ein GUI-Builder ist ein grafisches Entwicklungstool für grafische Benutzeroberflächen. Der in diesem Kapitel verwendete GIU-Builder mit dem Namen WindowBuilder Editor war in den Jahren 2003 bis 2010 ein Produkt von Instantiations Inc.. Das Java-Geschäft von Instantiations wurde 2010 von Google übernommen und im gleichen Jahr wurde das Produkt WindowBuilder Editor von Google kostenlos zur Verfügung gestellt. WindowBuilder Editor wird allgemein als sehr hochwertige Software angesehen und soll Bestandteil des Eclipse-Projektes werden, gleichwohl weiterhin von Google finanziell unterstützt, da WindowBuilder Editor weiterhin die Basis für das Google-interne GUI-Builder Produkt sein soll.

Neben anderen Systemen zur Programmierung von grafischen Benutzeroberflächen in Java unterstützt WindowBuilder Editor auch die Verwendung des Swing-Paketes. Im Folgenden wird diese Funktionalität zum Erzeugen grafischer Benutzeroberflächen mit dem Swing-Paket (javax.swing) betrachtet.

Wie jedes Tool wird für den effektiven Einsatz des Tools ein geeignetes Konzept benötigt. Das für Kapitel 13 gewählte Konzept lässt sich wie folgt kurz beschreiben:

- **Programmierung der GUI mit WindowBuilder Editor unter Verwendung von Panels als Container**
  Bereiche der grafischen Benutzeroberfläche (GUI) werden stets in sogenannnten Panels zusammengefasst. Diese Panels bestehen aus einem Stück GUI und sind abgeleitet von java.awt.Component. Eine Komponente stellt ein Stück grafische Benutzeroberfläche dar. Alle Elemente der GUI sind direkt oder indirekt von java.awt.Component abgeleitet. Ein Panel ist somit eine Komponente ohne Rahmen. Panels können frei ineinander verschachtelt werden. Eine mehrfache Verwendung eines Panels in einem Programm ist ebenfalls möglich (Details hierzu in Kapitel 3.3.2 und 3.3.3). Das in diesem Kapitel dargestellte Konzept beinhaltet, **dass die grafische Benutzeroberfläche in Form von Panels erzeugt wird, die mit dem Eclipse WindowBuilder Editor erzeugt und bearbeitet wurden.**

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JPanel
```

- **Einfügen der Panels in Standard-Fenster**
  Die Anwendungen in Kapitel 13 bestehen stets aus einem Panel, das in ein Standard-Fenster eingefügt wird. Dieses Standard-Fenster ist ein Objekt der Klasse **`de.eikelberg.guitools.ESinglePanelFrame`**. Diese Klasse ist von der Klasse `javax.swing.JFrame` abgeleitet, ihr Programmcode wird weiter unten erklärt. Die Anordnung mehrerer Panels sowie die Technik zur Panelverschachtelung muss bei Bedarf in Kapitel 3.3.2 und 3.3.3 nachgelesen werden. Damit in einfacher Art und Weise Anwendungen mit Menüsteuerung und mehreren hintereinandergelegten Panels programmiert werden können, wird zudem in Kapitel 15.2 die Klasse **`de.eikelberg.guitools.EFrame`** bereit gestellt. Die Anwendung der Klassen dieser Klasse wird in Kapitel 15.2 erklärt.

- **Anwendungsstart und Fensteraufbau in einer Standard-Klasse Programmstart**
  Ein Fenster ist stets ein Objekt einer Fensterklasse. Entsprechend wird eine Standard-Klasse `Programmstart.java` bereit gestellt, in der die main-Methode enthalten ist und in der der erstmalige Fensteraufbau erfolgt.

> Der Quellcode des Paketes de.eikelberg.guitools ist **offen zugänglich** und im Beispielprojekt jpr15020 enthalten. Das Paket de.eikelberg.guitools ist **frei verwendbar**, es wird aber vom Autor **keinerlei Haftung** übernommen. Eine Übersicht über die Funktionalität dieses Pakets sowie die Anwendung weiterer Klassen des Paketes finden Sie in Kapitel 15. Das Paket de.eikelberg.guitools wird bereitgestellt entsprechend den Regelungen der **Common Public License Version 1.0 (CPL)** (siehe Lizenztext im Beispiel jpr15020).

13.1.2 Die Standard-Klasse für den Fensteraufbau

In diesem Kapitel werden die beiden Standard-Klassen für den Fensteraufbau und den Programmstart erklärt:

Die wichtigste Klassen für den Fensteraufbau ist die Klasse **`ESinglePanelFrame`**, in die das Panel eingebettet wird und die das Fenster aufbaut. Da sie für die Programme in Kapitel 13 niemals verändert werden muss, wird sie als Klasse im Paket de.eikelberg.guitools bereit gestellt. Im Projekt jpr13001 ist der Programmcode der Klasse de.eikelberg.guitools.ESinglePanelFrame sichtbar enthalten. Die im Programmcode fett hervorgehobenen Zeilen sind dabei Anweisungen, die standardmäßig in jeder von JFrame abgeleiteten Fensterklasse enthalten sein sollten. Die übrigen Zeilen sind individuelle Zeilen, die die Fensteranzeige verbessern:

```
package de.eikelberg.guitools;

import ...

/**
 * Die Klasse ESinglePanelFrame öffnet ein Fenster, in das ein einziges
 * Panel eingebettet werden kann. Die Referenz auf das Fenster und der
 * im Kopf des Fensters einzutragende Titel müssten beim Aufruf des Konstruktors
 * mit angegeben werden
 * @author Eikelberg
 * @since 2008-05-06
 */
public class ESinglePanelFrame extends JFrame
{
  private static final long serialVersionUID = 710L;
/**
 * Der Konstruktor baut das Fenster auf.
 * @param mainPanel Referenz auf das Panel, das im Fenster angezeigt werden
 * soll. Dieses Panel sollte eine feste Größe besitzen
 * @param titel Titel, der im Kopf des Fensters eingetragen werden soll
 */
  public ESinglePanelFrame(JPanel mainPanel,String titel)
  {
    System.out.println("\nFensteraufbau mit de.eikelberg.guitools Rel 2.1.0...");

// Auf den Inhaltsbereich des Fensters zugreifen und LayoutManager setzen
    Container myCP = this.getContentPane();

//  Layout festlegen - es wird nur ein JScrollPane-Objekt eingefügt
    myCP.setLayout(new GridLayout(1,1));

// JScrollPane-Objekt erzeugen und einzubettendes Panel einfügen
    JScrollPane myScrollPane = new JScrollPane(mainPanel);

// JScrollPane in die ContentPane einfügen
    myCP.add(myScrollPane);

// Text in die Kopfzeile des Fensters eintragen
    this.setTitle(titel);

// Größe des eingebetteten Panels feststellen
    int panelBreite = mainPanel.getWidth();
    int panelHoehe = mainPanel.getHeight();
// Positionsdaten für das Fenster bestimmen
    FensterAnordnung fa;
    fa = this.berechneFensterPosition(this, panelBreite,panelHoehe);
// Fenster anordnen
    this.setLocation(fa.getXPosition(),fa.getYPosition());
    this.setSize(fa.getFensterBreite(),fa.getFensterHoehe());

// Window-Listener für das Schließen der Anwendung hinzufügen
    this.addWindowListener(
      new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          dispose();
        }
      }
    );

// Anzeige des Fensters
   this.setVisible(true);
  }

/**
 * Methode berechnet die Positionsdaten für die Anordnung des Fensters, so dass
 * dieses mittig auf dem Desktop angeordnet wird. Hierzu werden folgende
 * Konfigurationseinstellungen aus der Datei config/fenster.properties gelesen
 * (alle Angaben in Pixel):
 * fenster.rahmen.hoehe - gesamte Höhe des unteren und oberen Fensterrahmens
 * fenster.rahmen.breite - gesamte Breite des linken und rechten Fensterrahmens
```

```
 * fenster.menu.hoehe - Höhe, die für die Anzeige eines Menüs benötigt wird
 * scrollbalken.breite - Breite des Scrollbalkens bei zu kleinem Fenster
 * scrollbalken.hoehe - Höhe des Scrollbalkens bei zu kleinem Fenster
 * @param mf Referenz auf das Fenster
 * @param panelBreite Breite des eingebetteten Panels
 * @param panelHoehe Höhe des eingebetteten Panels
 * @return
 */
  private FensterAnordnung berechneFensterPosition(
      ESinglePanelFrame mf, int panelBreite, int panelHoehe)
  {
// Einstellungen für die Fensterkonfiguration aus der
// Konfigurationsdatei lesen
    Properties myprop = null;
    int rahmenHoehe=0,rahmenBreite=0,menueHoehe=0;
    int scrollPaneRahmenBreite=1,scrollPaneRahmenHoehe=1;
    String propertiesDatei = "config/fenster.properties";
    try
    {
      FileInputStream ein = new FileInputStream(propertiesDatei);
      myprop = new Properties();
      myprop.load(ein);        // Properties lesen
      ein.close();          // Properties-Datei schließen
      rahmenHoehe = Integer.parseInt(myprop.getProperty("fenster.rahmen.hoehe"));
      rahmenBreite = Integer.parseInt(myprop.getProperty("fenster.rahmen.breite"));
      menueHoehe = Integer.parseInt(myprop.getProperty("fenster.menu.hoehe"));
    } catch (Exception e)
    {
      System.out.println(
        "\nFehler beim Einlesen oder Auswerten der Properties-Datei "+
        propertiesDatei+"!\n");
// Standard-Werte setzen, falls obiges Einlesen nicht gelingt
      rahmenHoehe=30;
      rahmenBreite=20;
      menueHoehe=20;
    }
// Fenstergröße berechnen
    int fensterBreite = panelBreite+rahmenBreite+2*scrollPaneRahmenBreite;
    int fensterHoehe = panelHoehe+rahmenHoehe+menueHoehe+2*scrollPaneRahmenHoehe;
// Breite und Höhe des Desktops in Pixel feststellen
    int desktopHoehe = Toolkit.getDefaultToolkit().getScreenSize().height;
    int desktopBreite = Toolkit.getDefaultToolkit().getScreenSize().width;
// Koordinaten für die linke obere Ecke des Fensters bestimmen
    int xPosition = (desktopBreite-fensterBreite)/2;
    int yPosition = (desktopHoehe - fensterHoehe)/2;
// berechnete Fensteranordnung in einem Objekt zurückgeben
    FensterAnordnung fa = new FensterAnordnung(
       xPosition,yPosition,fensterBreite,fensterHoehe);
    return fa;
  }

/**
 * Klasse zur Speicherung der Postitionsdaten eines Fensters
 */
  private class FensterAnordnung
  {
    private int xPosition;
    private int yPosition;
    private int fensterBreite;
    private int fensterHoehe;
    protected FensterAnordnung(
        int xPosition, int yPosition, int fensterBreite, int fensterHoehe)
    {
      this.xPosition = xPosition;
      this.yPosition = yPosition;
      this.fensterBreite = fensterBreite;
      this.fensterHoehe = fensterHoehe;
    }
    protected int getXPosition() { return this.xPosition; }
    protected int getYPosition() { return this.yPosition; }
    protected int getFensterBreite() { return this.fensterBreite; }
    protected int getFensterHoehe() { return this.fensterHoehe; }
```

```
  }
}
```

Wie aus obigem Quellcode ersichtlich ist, können die Größenangaben für

- Fensterrahmen-Hoehe
- Fensterrahmen-Breite
- Höhe eines ggf. eingefügten Menübalkens

in einer Konfigurationsdatie config/fenster.properties eingestellt werden. Diese kann zum Beispiel für das klassische Windows-Design mit folgenden Werten gefüllt sein:

```
; Dickte und Höhe des Fensterrahmens passend zum Betriebssytem eintragen
; Es muss die Summe von unterem und oberem Rahmen bzw. von linkem und rechtem
; Rahmen eingetragen werden, z. Bsp. 30 Pixel für Rahmenhöhe, 20 Pixel für
; Rahmenbreite, 0 Pixel für den Menübalken
; Hier die Einstellungen für Windows 7 mit klassischem Design und ohne Menübalken:
fenster.rahmen.hoehe = 28
fenster.rahmen.breite = 9
fenster.menu.hoehe = 0
```

13.1.3 Die Klasse für den Programmstart

Die für jedes Java-Programm benötigte Klasse mit der main-Methode heißt in Kapitel 13 stets `Programmstart.java`. Diese Klasse sollte folgende Abschnitte enthalten

1. Zu Beginn sollte eine geeignete Startmeldung ausgegeben werden.
2. In der Klasse Programmstart.java kann der Fenstertitel festgelegt werden.
3. Ferner muss in der Klasse Programmstart.java ein Objekt der Klasse mit dem Panel instanziiert werden.
4. Zum Abschluss muss ein Objekt der Fensterklasse instanziiert werden.

Die Klasse Programmstart.java muss also für jedes Programm geeignet angepasst werden. Für das Projekt jpr13001 enthält sie zum Beispiel folgenden Programmcode:

```
package bsp13001;

import de.eikelberg.guitools.ESinglePanelFrame;

public class Programmstart
{

  public static void main(String[] args)
  {
// 1. Startmeldung
    System.out.println(
      "\nBeispielprogramm zum Aufbau eines Fensters");
    System.out.println(
      "Copyright 2008 by M. Eikelberg (www.eikelberg.de)\n");

// 2. Fensterbeschriftung festlegen
    String titel = "Haupt-Fenster der Anwendung";

// 3. Panel instanziieren
    MainPanel mp = new MainPanel();

// 4. Fenster instanziieren und dabei Referenz auf das Panel sowie
// die Fensterbeschriftung an den Konstruktor übergeben
    new ESinglePanelFrame(mp,titel);
  }
}
```

13.1.4 Grundprinzip der Programmierung eines Panels mit WindowBuilder Editor

In diesem Abschnitt wird zunächst das Grundprinzip der Programmierung eines Panels mit WindowBuilder Editor erläutert. Die Codierung der Ereignisbehandlung sowie der Zugriff auf die Komponenten der GUI aus verarbeitenden Klasse wird später erklärt (siehe Kapitel 13.3).

Konfiguration von WindowBuilder Editor

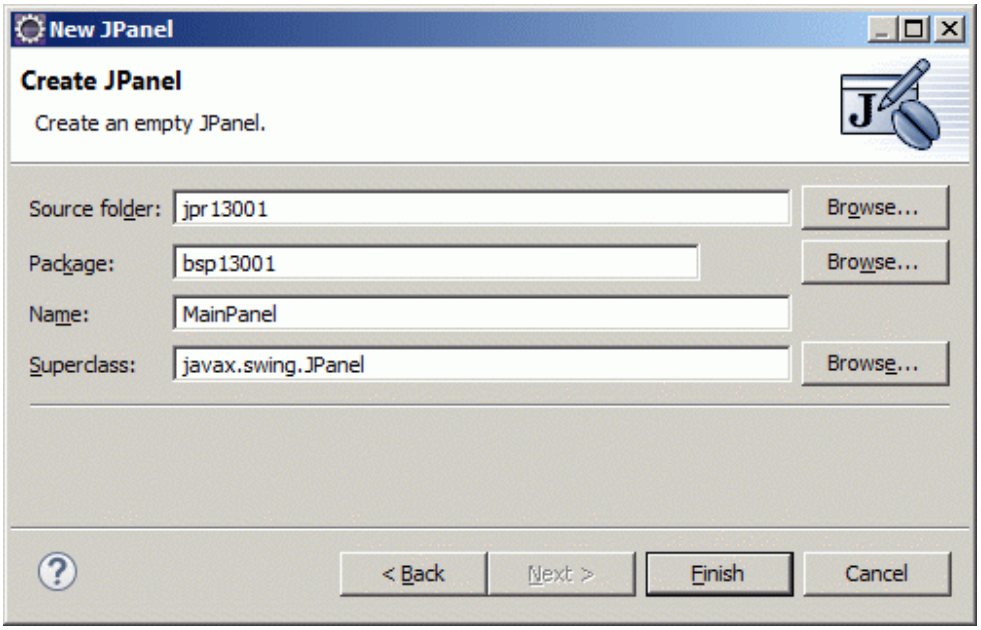
Für die Verwendung von WindowBuilder Editor entsprechend den Erläuterungen in diesem Kapitel muss WindowBuilder Editor geeignet konfiguriert werden. Dies geschieht wie folgt:

- Öffnen Sie in Eclipse im Window-Menü den Menüpunkt Preferences. Es öffnet sich das Fenster "Preferences"
- Markieren Sie den Zweig WindowBuilder und wählen Sie im rechten Teil des Fensters die Option "Format source code (and reparse) on editor save" aus.
- Erweitern Sie im Zweig WindowBuilder und wählen Sie Swing / Code Generation. Im rechten Bereich des Fensters "Preferences" wird das Panel "Code Generation" angezeigt.
- Deaktivieren Sie die Option "Use the existing code generation settings when they can be deduced".
- Stellen Sie als "method name for new statements" den Namen `initialize` ein.
- Wechseln Sie im Bereich "default code generation settings" in die Registerkarte "Field" und wählen Sie die Option "prefix field access with this" aus. Stellen Sie als "field modifier" private ein und konfigurieren Sie für die "statement generation" ein Erzeugen eines eigenen Blocks für jede Komponente, indem Sie die Registerkarte Block auswählen. Ürnehmen Sie diese Einstellungen mit Klick auf den Button "Apply".
- Wechseln Sie jetzt in dem Bereich "default code generation settings" in die Registerkarte "Lazy". Konfigurieren Sie für den "method modifier" den Wert protected und übernehmen Sie diese Einstellungen mit "Apply".
- Wechseln Sie jetzt im Zweig WindowBuilder / Swing / Code Generation in das Blatt "Event handlers" und stellen Sie sicher, dass die Option "Create anonymous class" eingestellt ist.
- Wechseln Sie jetzt im Zweig WindowBuilder / Swing / Code Generation in das Blatt "Variables". Stellen Sie in der Registerkarte "Type specific" entsprechend der Tabelle in Kapitel 13.1.5 die Standard-Namen und Acronyme für die Komponenten der grafischen Benutzeroberfläche ein. Aktivieren Sie dabei für alle Komponenten die Option "As field".

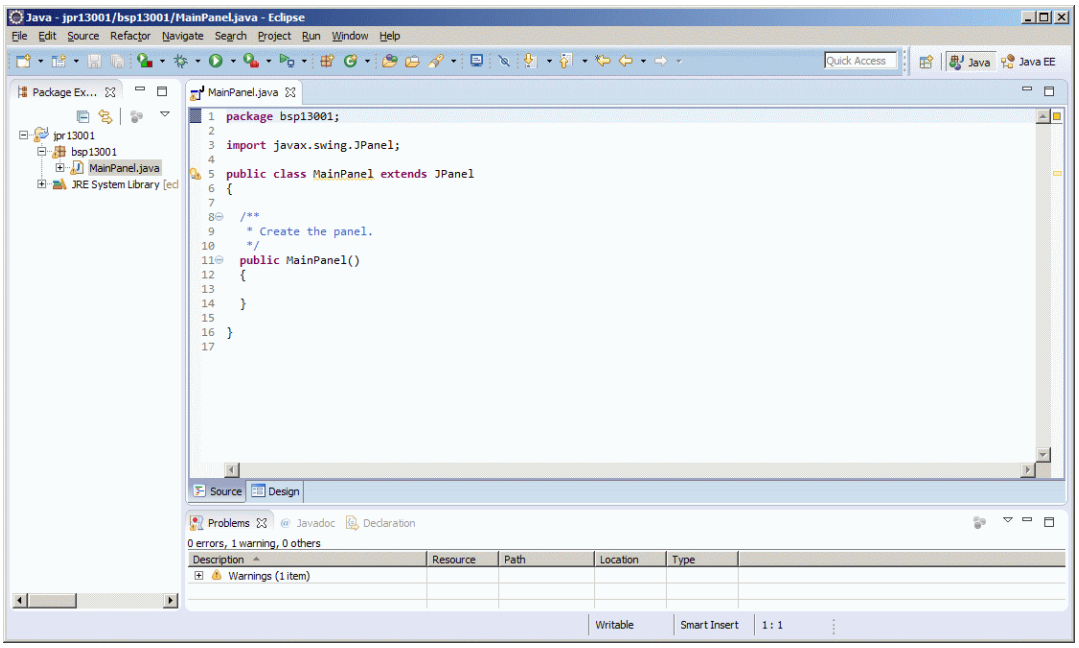
Anlegen eines Panels mit WindowBuilder Editor

Das Anlegen des Panels geschieht wie folgt:

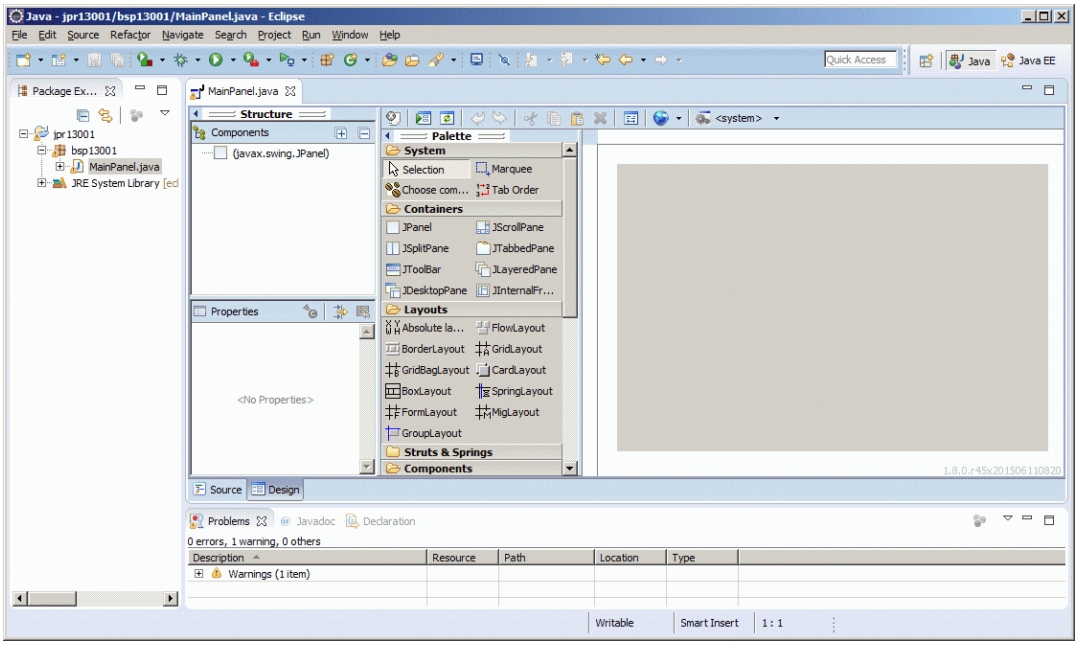
- Markieren Sie im Package Explorer von Eclipse das Paket, in das die Panel-Klasse eingefügt werden soll und öffen Sie durch Klick mit der rechten Maustaste das Kontextmenü des Paketes.
- Wählen Sie im Kontextmenü New / Other
- Wählen Sie im Fenster New den Zweig WindowBuilder / Swing Designer / JPanel und Klicken Sie auf "Next".
- Tragen Sie im Fenster "New JPanel" den Namen des Panels ein (MainPanel). Als Superclass sollte die Klasse javax.swing.JPanel eingetragen bleiben. Dadurch wird die neue Klasse von javax.swing.JPanel abgeleitet.



- Bestätigen Sie Ihre Angaben mit Klick auf den Button "Finish". Daraufhin wird in Eclipse wie folgt das Panel angezeigt:



- Nach Auswahl der Registerkarte "Design" wird das Panel im GUI Builder angezeigt:

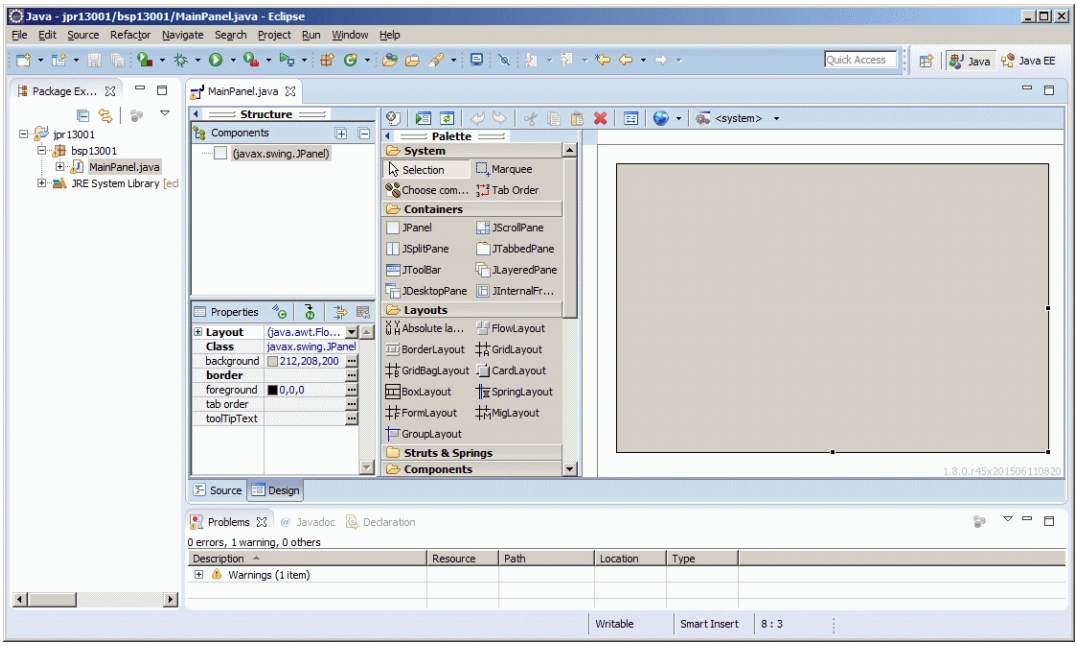


- Spezifisch für den WindowBuilder Editor sind folgende Komponenten in der GUI von Eclipse hinzugekommen:
  - Die Anzeige der Klasse MainPanel.java erfolgt in zweifacher Weise: Am unteren Rand der mit dem Namen des JPanels beschrifteten Registerkarte (MainPanel.java) befinden sich zwei Registerkarten "Source" und "Design". Wechselt man in die Registerkarte "Source", so wird der Quellcode des JPanels angezeigt. Dieser kann hier auch modifiziert werden. In der Registerkarte "Design" wird das Panel im WYSIWYG-Modus dargestellt (**w**hat **y**ou **s**ee **i**s **w**hat **y**ou **g**et). Man erhält dort also eine Vorschau auf die beim Programmlauf angezeigte GUI. Das Besondere ist dabei, dass in der Vorschau eine Bearbeitung erfolgen kann, die simultan im Quellcode nachgeführt wird.
  - In der Abschnitt "Palette" der Registerkarte "Design" werden alle Komponenten (Buttons, Panels etc) der GUI werden als sogenannte Java Beans bereitgestellt. Im diesem Bereich ist eine schnelle Navigation durch die der GUI hinzugefügten Komponenten möglich. Java Beans können ausgewählt, in das Panel eingefügt und konfiguriert werden. Alle so generierten Änderungen des JPanels werden simultan im Quellcode nachgeführt.
  - In der Abschnitt "Structure" der Registerkarte "Design" werden die Komponenten der grafischen Benutzeroberfläche angezeigt. Wenn man eine Komponente markiert, werden die Eigenschaften (Properties) der entsprechenden Komponente angezeigt. Diese können im Bereich "Properties" geändert werden, wobei sich die Änderungen in der WYSIWYG-Anzeige und im Quellcode simultan nachgeführt werden.

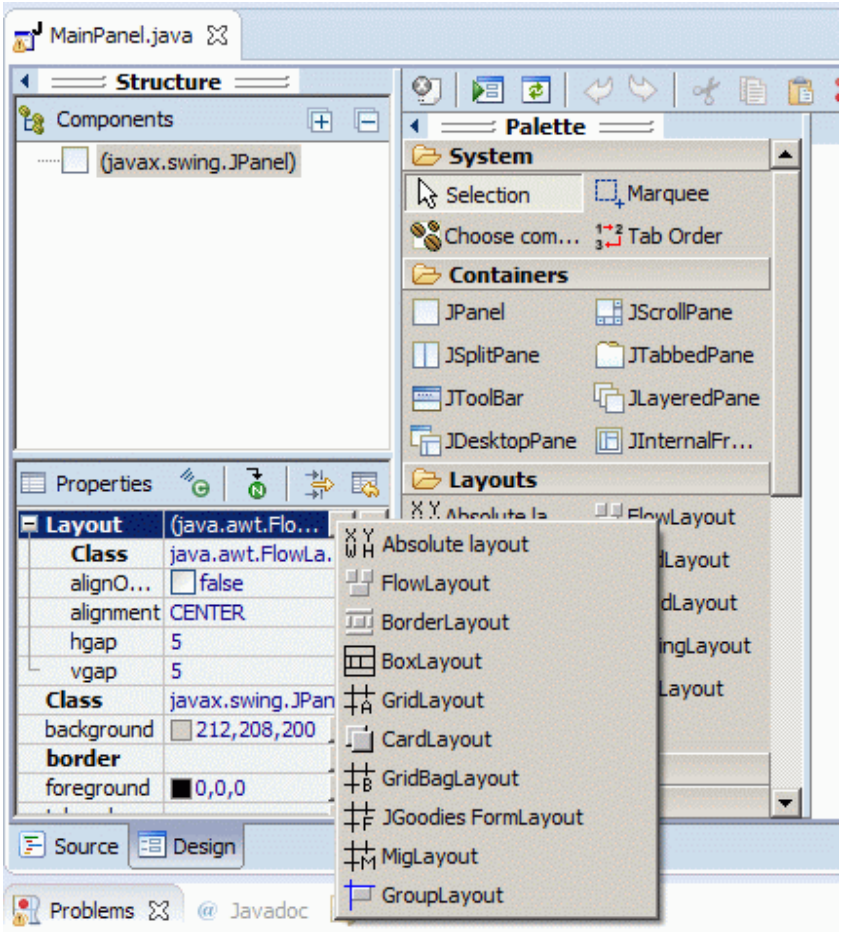
Konfiguration des Panels mit der Registerkarte Properties

Wie bereits erwähnt können die Eigenschaften des Panels in der Registerkarte Properties eingestellt werden. Hierzu geht man wie folgt vor:

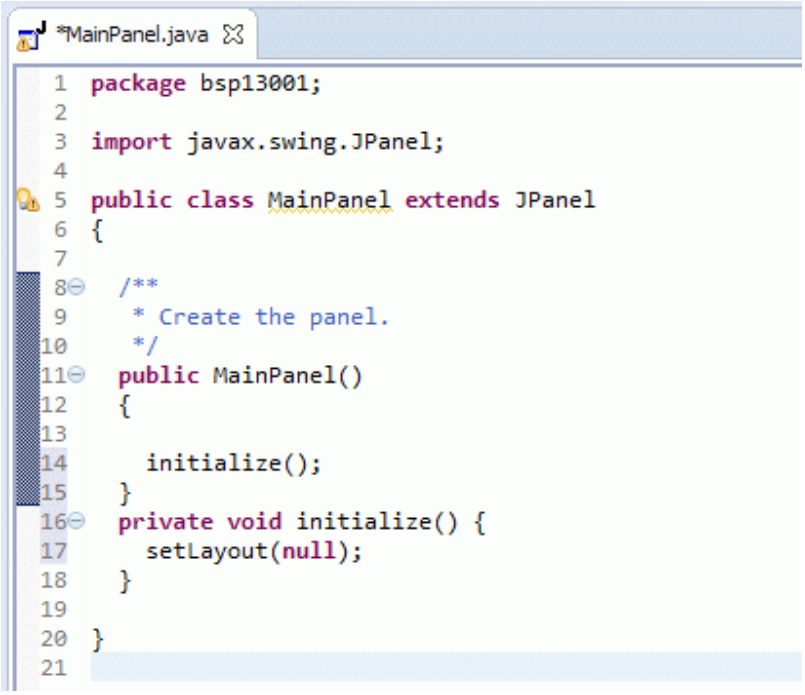
- **Zugriff auf die Eigenschaften des Panels**
  Man markiere zunächst in der WYSIWYG-Anzeige die Komponente, die konfiguriert werden soll - hier also das Panel. Im Panel "Properties" erscheinen daraufhin die Eigenschaften der Komponente:



- **Konfiguration des Layouts**
  Abhängig vom Typ des Werts kann dieser Wert im Oberflächenbereich "Propierties" durch Mausklicks in einem Bereich der Zeile zum entsprechenden Wert geändert werden. Hier soll zunächst eine Eigenschaft des Panels geändert werden - als Layout soll das Null-Layout eingestellt werden (was die weitere Bedienung des WindowBuilder Editors vereinfacht). Erweiteren Sie den Zweig "Layout" und klicken Sie auf den abwärtsgerichteten Pfeil neben dem voreingestellten Wert java.awt.FlowLayout. Daraufhin werden die einstellbaren Layouts angezeigt:



Wählen Sie das "Absolute Layout" aus. Daraufhin wird im Quellcode das Null-Layout eingestellt:

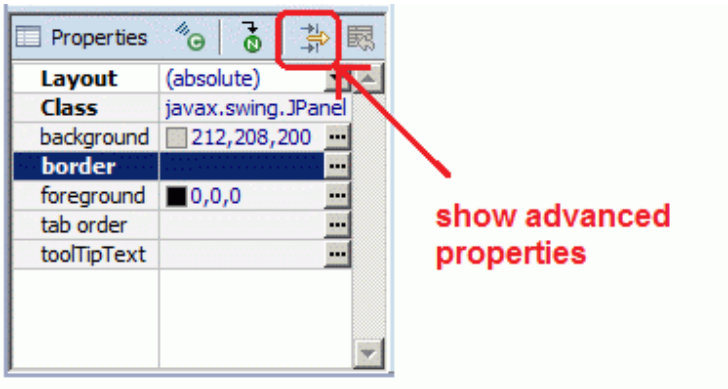


```
*MainPanel.java
1  package bsp13001;
2
3  import javax.swing.JPanel;
4
5  public class MainPanel extends JPanel
6  {
7
8    /**
9     * Create the panel.
10    */
11   public MainPanel()
12   {
13
14     initialize();
15   }
16   private void initialize() {
17     setLayout(null);
18   }
19
20 }
21
```

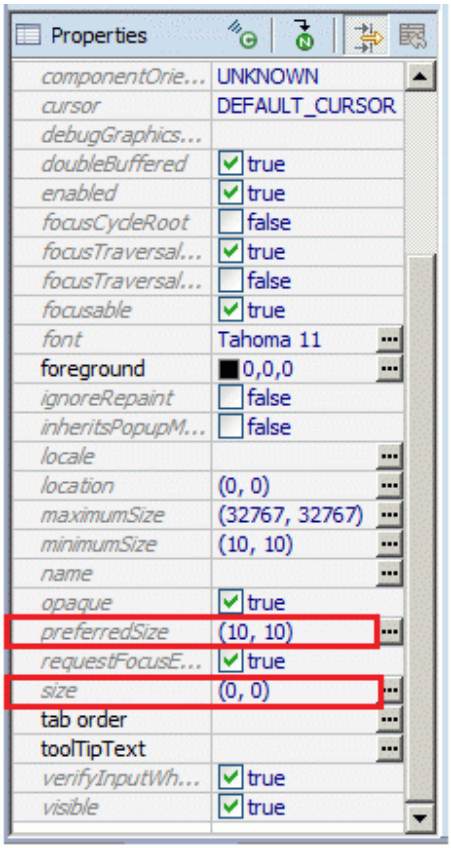
- **Konfiguration der Eigenschaften "size" und "preferredSize"**

Konfigurieren Sie stets die Properties "size" und "preferredSize" und stellen Sie dabei für beide Eigenschaften den gleichen Wert ein:

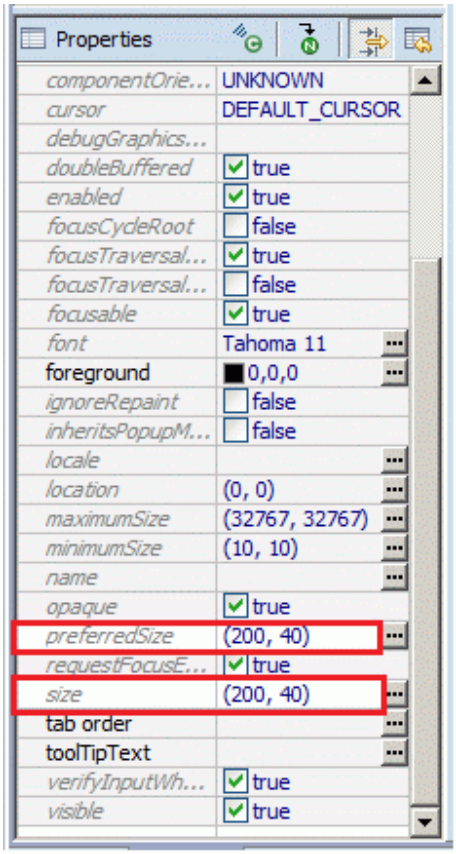
Klicken Sie auf den Button "show advanced properties", um weitere Eigenschaften des Panels anzuzeigen:



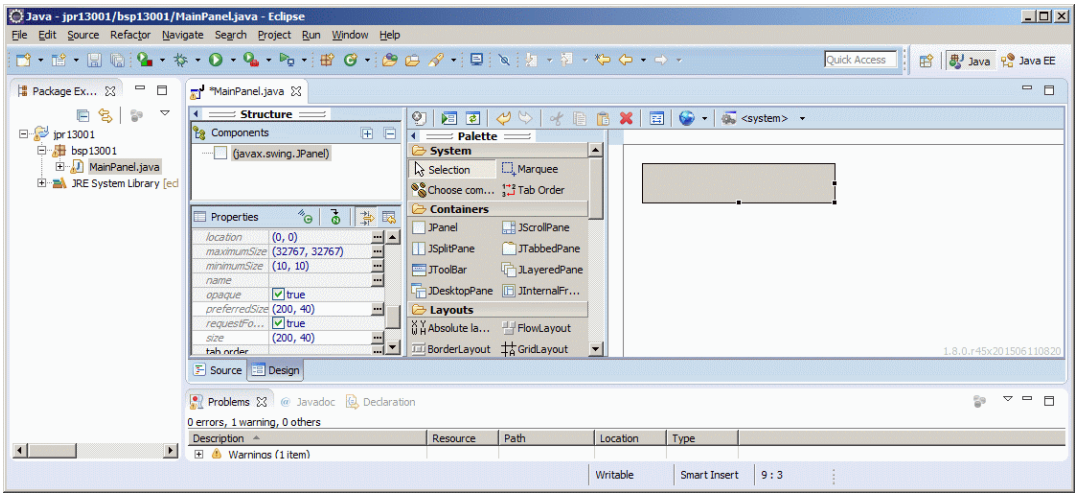
Daraufhin werden weitere Eigenschaften des Panels angezeigt, u.a. auch die Eigenschaften "size" und "preferredSize":



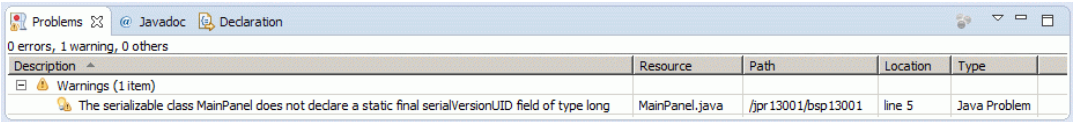
Klicken Sie auf den Button "..." neben den Werten von "size" und "preferredSize" und stellen Sie für beide Eigenschaften die gleichen Werte ein:



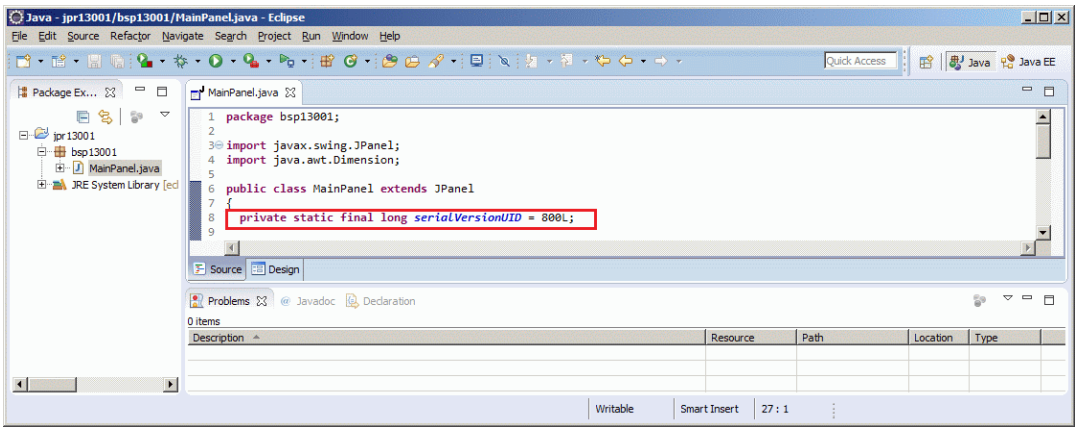
Als Ergebnis dieser Einstelllungen erhalten Sie ein leeren Panel mit grauer Farbe, das im Beispiel jpr13001 wie folgt angezeigt wird:



Hinweis: Bei der Bearbeitung des Panels mit dem WindowBuilder Editor kann es vorkommen, dass Warnungen entstehen. Diese Warnungen werden in dem Bereich angezeigt, der den Java Quellcode enthält. Sie können durch direkte Änderung des Quellcode behoben werden. Im vorliegenden Beispiel wird die Warnung angezeigt, dass eine Variable namens "serialVersionUID" fehlt:



Diese Warnung kann durch manuelles Einfügen der entsprechenden Variablen (ggf. unter Verwendung von "QuickFix") behoben werden. Für das vorliegende Beispiel bedeutet dies folgende manuelle Einfügung von Quellcode:



Nach diser Korrektur gemäß obigem Hinweis wird folgender Programmcode angezeigt:

```
package bsp13001;

import javax.swing.JPanel;
import java.awt.Dimension;

public class MainPanel extends JPanel
{
  private static final long serialVersionUID = 710L;

  /**
   * Create the panel.
   */
  public MainPanel()
  {
    initialize();
  }

  private void initialize()
  {
    setSize(new Dimension(200, 40));
    setPreferredSize(new Dimension(200, 40));
    setLayout(null);
  }

}
```

Der Programmcode enthält folgende Abschnitte:

- Konstruktor MainPanel() - ruft wie jeder Konstruktor zunächst den Superkonstruktor auf, also den Konstruktor der Klasse javax.swing.JPanel. Dieser Aufruf des Superkonstruktors ist im Quellcode nicht sichtbar. Anschließend wird die Methode initialize() aufgerufen, die die weitere Konfiguration des Panels vornimmt.
- Methode initialize() - führt die spezielle Konfiguration des Panels durch.

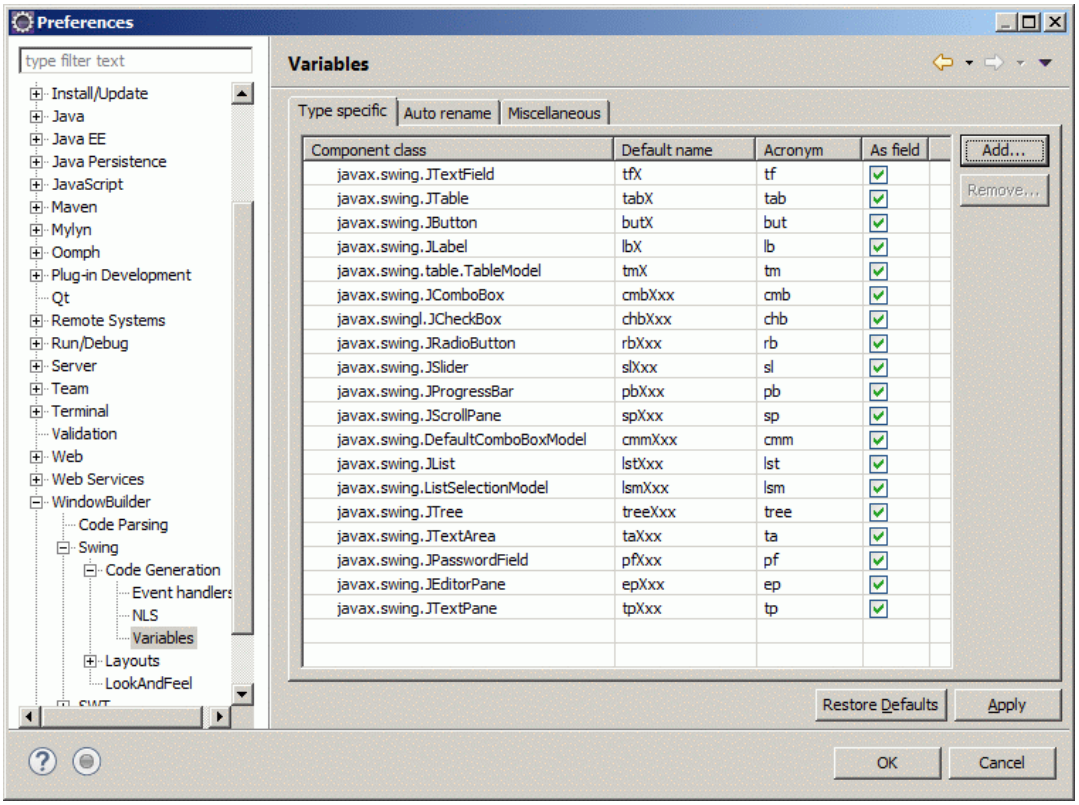
13.1.5 Grundelemente für die inhaltliche Gestaltung von Fenstern

Bis zum Ende von Kapitel 13.3 sollen in ein Fenster lediglich folgende Komponenten (also Bestandteile der grafischen Benutzeroberfläche) eingefügt werden. Die folgende Tabelle zeigt Komponenten und empfohlene Namen für diese Komponenten:

Name der Komponente	Java-Klasse	Verwendungszweck	empfohlener Name für die Komponente	als Instanzvariable
Textfeld	javax.swing.JTextField	Ein- und/oder Ausgabefeld.	tf*Xxx*	ja
Tabelle	javax.swing.JTable	tabellarische Anzeige	tb*Xxx*	ja
Button	javax.swing.JButton	Button für Reaktionen auf Ereignisse, die vom Benutzer ausgelöst	but*Xxx*	ja

Label	javax.swing.JLabel	Beschriftung (in der Regel ohne Ereignisbehandlung).	lb*Xxx*	ja
Tabellenmodell	javax.swing.table.TableModel	Speicherung von Tabellen-Datein	tm*Xxx*	ja
Combobox	javax.swing.JComboBox	Markierungsfeld	cmb*Xxx*	ja
Checkbox	javax.swing.JCheckBox	Kombinationsfeld	chb*Xxx*	ja
Radiobutton	javax.swing.JRadioButton	Optionsfeld	rb*Xxx*	ja
Slider	javax.swing.JSlider	Schieberegler	sl*Xxx*	ja
Progressbar	javax.swing.JProgressBar	Fortschrittsbalken	pb*Xxx*	ja
Scrollpane	javax.swing.JScrollPane	Panel mit Scrollbalken	sp*Xxx*	ja
Combobox-Model	javax.swing.DefaultComboBoxModel	Datenmodell für Markierungsfeld	cmm*Xxx*	ja
List	javax.swing.JList	Listenfeld	lst*Xxx*	ja
List-Selection-Model	javax.swing.ListSelectionModel	Datenmodell für Listenfeld	lsm*Xxx*	ja
Tree	javax.swing.JTree	Baum	tree*Xxx*	ja
Textarea	javax.swing.JTextArea	mehrzeiliges Textfeld	ta*Xxx*	ja
Passwortfeld	javax.swing.JPasswordField	Eingabefeld für verdeckte Eingabe	pf*Xxx*	ja
Editorpane	javax.swing.JEditorPane	Editierfläche	ep*Xxx*	ja
Textpane	javax.swing.JTextPane	Textfläche	tp*Xxx*	ja

Nach Eintragung dieser Komponenten in Eclipse gemäß 13.1.4 sollten im Eclipse-Fenster "Properties" folgende Einträge angezeigt werden:



Vergessen Sie nicht, zum Abschluss Ihrer Eingabe diese Eingaben mit Klick auf den Button "Apply" zu bestätigen!

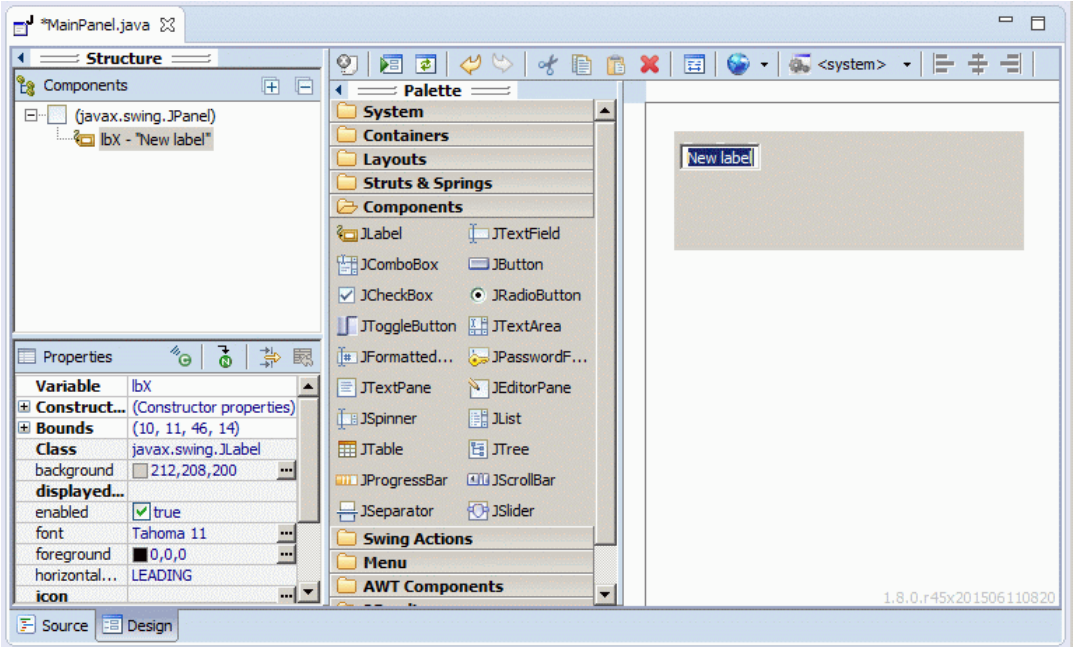
Alle genannten Komponenten können wie im nachfolgenden Abschnitt 13.1.6 beschrieben in einfacher Weise mit dem WindowBuilder Editor erzeugt und in die GUI eingefügt werden:

13.1.6 Einfügung von Komponenten in ein Panel

Hinweis: Ab Beispiel jpr13002 erfolgt der Fensteraufbau jeweils mit Hilfe der Klasse **de.eikelberg.guitools.ESinglePanelFrame**. Diese wird in der jar-Datei **de_eikelberg_guitools.jar** bereitgestellt. Die jar-Datei de_eikelberg_guitools.jar enthält das gesamte Paket de.eikelberg.guitools und muss ab Projekt jpr13002 in jedes Projekt mit eingebunden werden. Den gesamten Quellcode des Paketes de.eikelberg.guitools findet man im Projekt jpr15020. Eine Erläuterung des Paketes erfolgt in Kapitel 15.2.

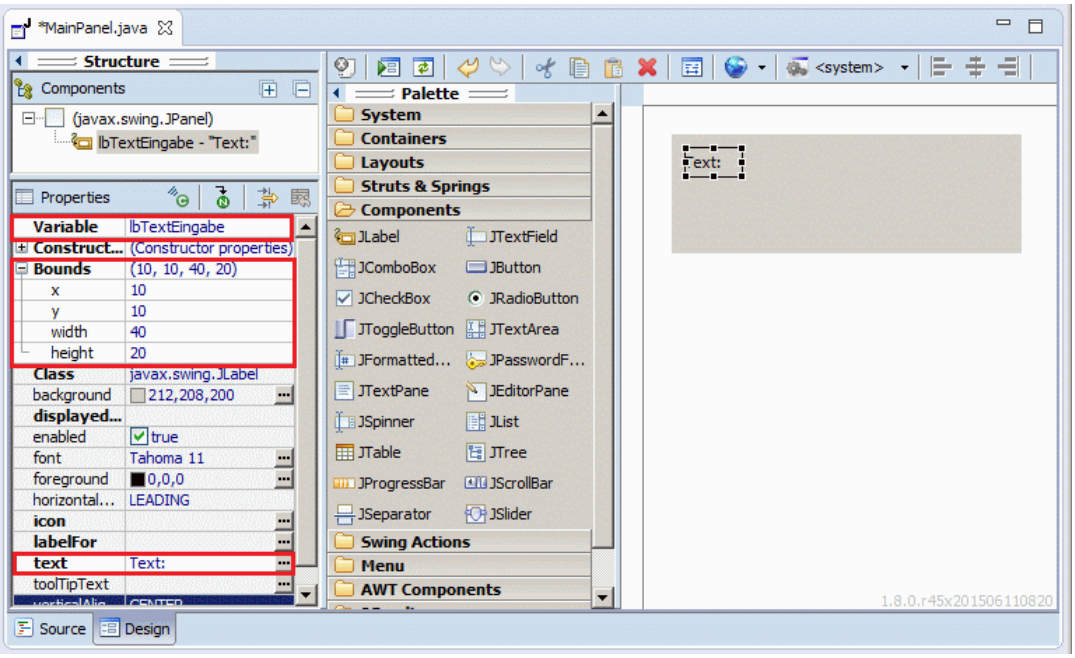
Das Einfügen von Komponenten in ein Panel mit WindowBuilder Editor ist denkbar einfach durchzuführen (vgl. Beispiel jpr13002):

- Man konfiguriere zunächst das Panel wie in 13.1.4 beschrieben.
- Stellen Sie für das Panel die Eigenschaften size und preferredSize ein auf den Wert (250, 85).
- Markieren Sie das Panel im Package Explorer und öffnen Sie dieses mit dem WindowBuilder Editor. Wechseln Sie in die Registerkarte "Design". Suchen Sie im Eingabebereich "Palette" den Abschnitt "Components" (dieser befindet sich weiter unten in der Liste). Kocken Sie einfach auf die Komponente JLabel und klicken Sie ein zweites Mal an den Ort im Panel, an dem das JLabel eingefügt werden soll. Daraufhin wird das Label in das Panel eingefügt:



- Ändern Sie nun im Abschnitt Properties die Eigenschaften der Komponente wie zum Beispiel
  - Variablennamen und Methodenname der get-Methode: "Variable" - zum Beispiel auf den Wert lbTextEingabe.
  - Ort und Größe: "Bounds" - zum Beispiel (x,y,width,heigt) = (10,10,40,20)
  - Beschriftung der Komponente: "text" - zum Beispiel auf den Wert "Text:"

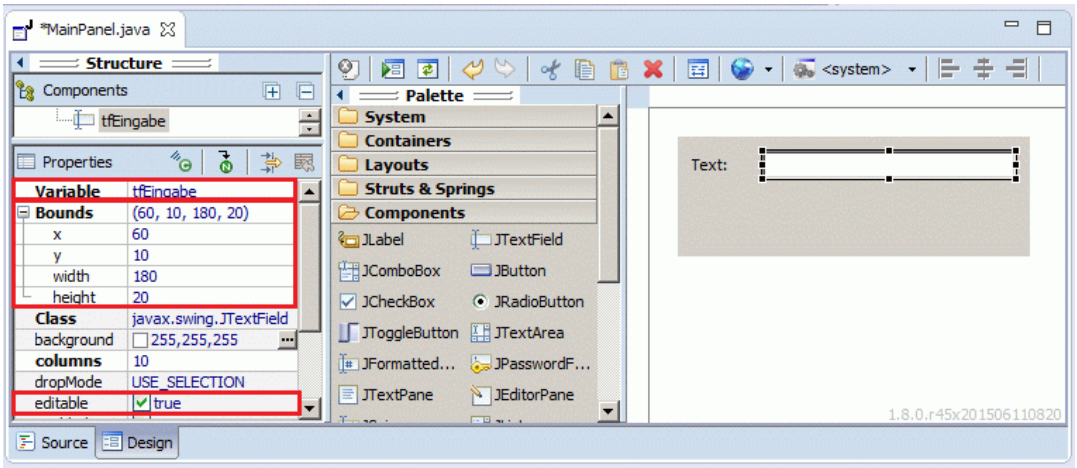
  Sie sollten dann folgende Anzeige erhalten:



Nach Einfügung einer Komponente vom Typ **JLabel** wird Programmcode wie in folgendem Ausschnitt von Beispiel jpr13002 erzeugt:

```
...

private JLabel lbTextEingabe;

...

private void initialize()
{
  setSize(new Dimension(250, 85));
  setPreferredSize(new Dimension(250, 85));
  setLayout(null);
  add(getLbTextEingabe());
}

protected JLabel getLbTextEingabe()
{
  if (lbTextEingabe == null)
  {
    lbTextEingabe = new JLabel("Text:");
    lbTextEingabe.setBounds(10, 10, 40, 20);
  }
  return lbTextEingabe;
}
```

Ein **JTextField** für die Eingabe - also mit der Möglichkeit einer Texteingabe - kann wie folgt eingefügt werden:



Für dieses Textfeld wird ein Programmcode wie in folgendem Ausschnitt aus Beispiel jpr13002 erzeugt:

```
...

private JTextField tfEingabe;

...

private void initialize()
{
  ...
  add(getTfEingabe());
}

protected JTextField getTfEingabe()
{
  if (tfEingabe == null)
  {
    tfEingabe = new JTextField();
    tfEingabe.setBounds(60, 10, 180, 20);
    tfEingabe.setColumns(10);
  }
  return tfEingabe;
}
```

Dieser Programmcode enthält folgende Abschnitte:

- Deklaration der JTextField-Komponente tfEingabe
- Hinzufügen der Komponente zum Panel in der Methode initialize()
- Codierung einer get-Methode, mit Hilfe derer beim ersten Aufruf die Komponente erzeugt und initialisiert wird und die die Referenz auf die Komponente zurückgibt.

Hinweis 1: Damit im Rahmen der Verarbeitung auf eine Komponente zugegriffen werden kann (z. Bsp. Lesen aus bzw. Schreiben in ein JTextField), ist Eclipse so konfiguriert worden, dass die Sichtbarkeit der get-Methode zur entsprechenden Komponete auf protected gesetzt wird!

Hinweis 2: Wenn ein JTextField lediglich zur Anzeige von Daten - also nur zur Ausgabe - verwendet wird, kann es wie folgt so konfiguriert werden, dass es sich optisch weniger vom Label unterscheidet:

- Man setze die Eigenschaft `editable` auf den Wert false.
- Man füge die nachfolgenden drei Anweisungen manuell in den Programmcode ein - sie bewirken, dass das Textfeld keinen Rahmen mehr hat und dass sowohl die Hintergrundfarbe als auch die Textfarbe identisch ist mit den jeweiligen Einstellungen des Panels:

```
//    die nächsten drei Zeilen sind manuelle Einfügungen
//    (Hintergrundfarbe, Farbe des Textes und Unterdrückung des Rands)
      tfEingabe.setBackground(this.getBackground());
      tfEingabe.setDisabledTextColor(Color.BLACK);
      tfEingabe.setBorder(null);
```

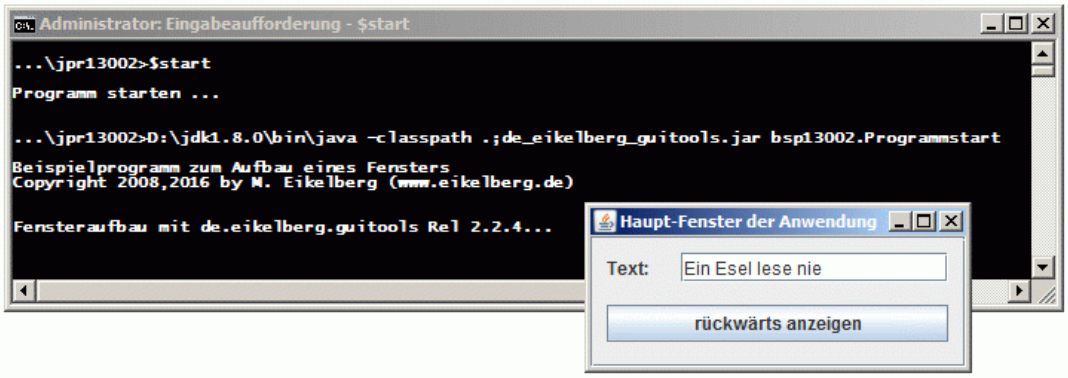
Analog zum JTextField wird für einen **JButton** ein folgender Programmcode eingefügt (vgl. Beispiel jpr13002):

```
...

private JButton butRueckwaerts;

...

private void initialize()
{
  ...
  add(getButRueckwaerts())
}

protected JButton getButRueckwaerts()
{
  if (butRueckwaerts == null)
  {
    butRueckwaerts = new JButton("r\u00FCckw\u00E4rts anzeigen");
```

```
      butRueckwaerts.setBounds(10, 45, 230, 25);
    }
    return butRueckwaerts;
  }
```

Hinweis: WindowBuilder Editor hat den in der GUI eingegebenen Text "rückwärts anzeigen" umgeformt in den oben sichtbaren Text, damit die deutschen Sonderzeichen auch auf nicht deutschen bzw. auf nicht-Windows-Systemen korrekt angezeigt werden.

Beispiel jpr13002 enthält bereits ein Label, ein Textfeld, einen Button sowie eine einfache Verarbeitung für Ereignisse zum Button, die in eine Klasse Verarbeitung.java ausgelagert ist. Daher wird folgendes Fenster angezeigt:



Nach Klick auf den Button "rückwärts anzeigen" wird der im Textfeld eingegebene Text in umgekehrter Reihenfolge angezeigt:

13.2 Ereignisbehandlung

Alle Programme zu Kapitel 13 enthalten folgende Bestandteile:

1. Programmstart.java - **Klasse mit der main-Methode**.
2. MainPanel.java - **JPanel-Klasse mit der GUI**. Diese Klasse wird in der Klasse Programmstart.java in ein Fenster der Klasse de.eikelberg.guitools.ESinglePanelFrame eingebettet.
3. Verarbeitung.java - **Klasse mit den verarbeitenden Methoden**.

Je nach Beispiel können vereinzelt andere als die aufgeführten Namen für die entsprechenden Klassen gewählt sein. Wegen Punkt 2 ist es jedoch stets erforderlich,

- dass in jedem Projekt die jar-Datei **de_eikelberg_guitools.jar** mit der Klasse ESinglePanelFrame eingebunden wird und
- dass in jedem Projekt für die Klasse ESinglePanelFrame in einem Unterverzeichnis **config** eine Konfigurationsdatei **fenster.properties** bereitgestellt wird, in der die Konfigurationseinstellungen für das ESinglePanelFrame-Fenster eingetragen sind.

Den Quellcode des Paketes de.eikelberg.guitools, die zugehörige jar-Datei sowie eine geeignete Konfigurationsdatei fenster.properties findet man im Beispielprojekt jpr15020.

13.2.1 Vorbereitung für die Einfügung der Ereignisbehandlung

In der Ereignisbehandlung wird die Referenz auf das Panel benötigt, aus dem gelesen bzw. in das geschrieben werden soll. Diese Referenz kann wie folgt in sehr einfacher Weise zur Verfügung gestellt werden. Dabei wird verwendet, dass private Instanzmethoden einer Klasse in inneren Klassen der Klasse uneingeschränkt verfügbar sind. Da in diesem Kapitel die Ereignisverarbeitung mit Hilfe von inneren Klassen programmiert wird, wird diese Eigenschaft wie folgt verwendet:

1. Manuelles Einfügen einer private Instanzmethode für den Zugriff auf die Referenz auf das Panel selbst.

Diese manuelle Einfügung ist in folgendem Ausschnitt des Quellcodes für das Panel bsp13002.MainPanel hervorgehoben:

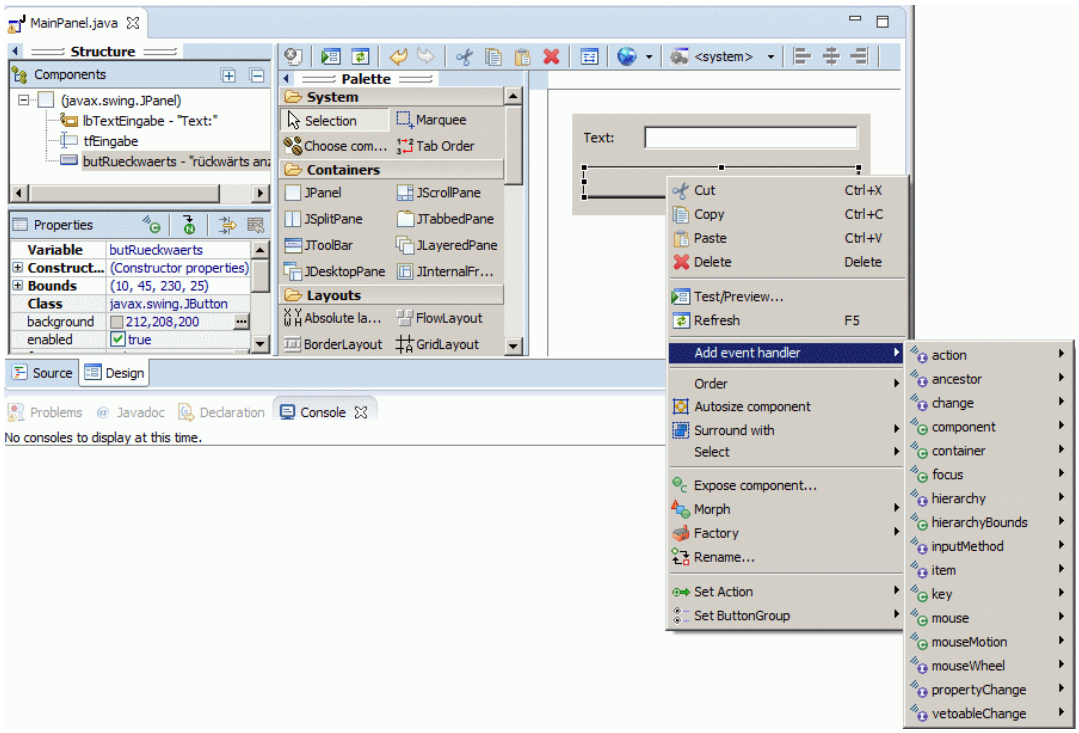
```
package bsp13002;

import javax.swing.JPanel;
...

public class MainPanel extends JPanel
{
  ...
// manuell eingefügt
  private MainPanel getMainPanel()
  {
    return this;
  }

  public MainPanel()
  {

    initialize();
  }

  private void initialize()
  {
    setSize(new Dimension(250, 85));
    setPreferredSize(new Dimension(250, 85));
    setLayout(null);
    ...
  }
  ...
}
```

Mit der eingefügten Methode kann nun auf die Referenz des Panels in der gesamten Klasse `MainPanel` sowie in allen inneren Klassen der Klasse `MainPanel` zugegriffen werden.

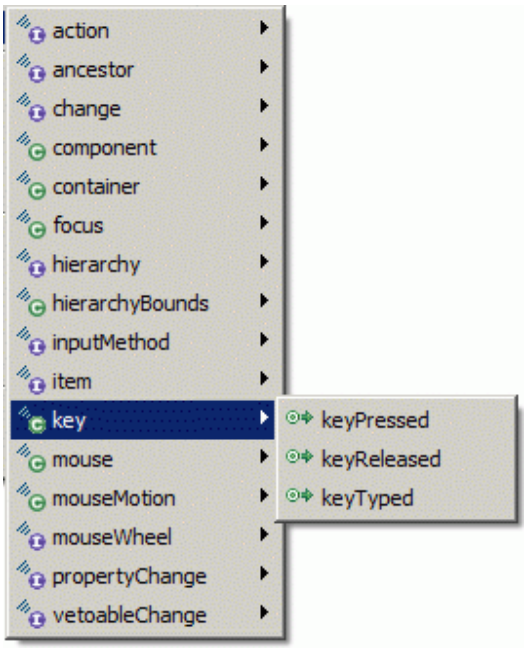
13.2.2 Hinzufügen einer Methode zur Ereignisbehandlung für eine Komponenten und für ein Ereignis

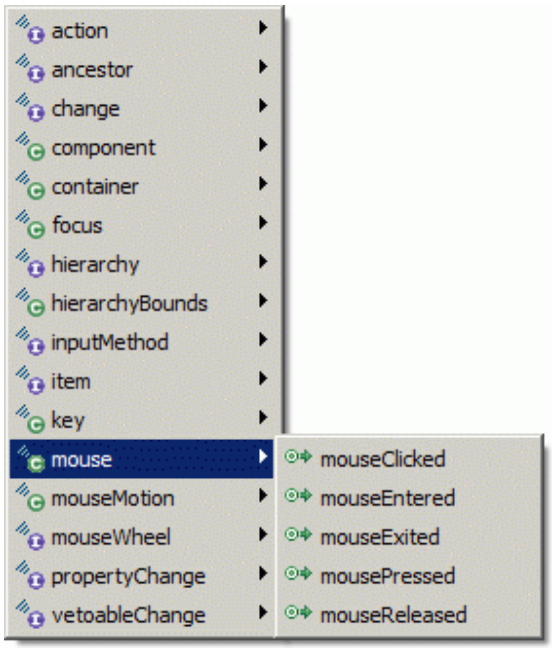
Das Hinzufügen von Methoden zur Ereignisbehandlung sollte mit Hilfe des Window geschehen, wenn das Panel mit der GUI ebenfalls mit dem WindowBuilder Editor erzeugt wurde. Beispiel jpr13002 zeigt das Vorgehen hierzu:

- Lassen Sie sich das Panel mit der Komponente (vgl. Kap. 13.1) im WindowBuilder Editor anzeigen und markieren Sie die Komponente, für die eine Ereignisbehandlung hinzugefügt werden soll.
- Öffnen Sie durch Klick mit der rechten Maustaste das Kontextmenü zur Komponente. Wählen Sie im Kontextmenü die Auswahl "Add event handler".



- Für jeden Event-Typ (z. Bsp. key- oder mouse-Event) werden jetzt die entsprechenden Ereignisse zur Auswahl angeboten:





In der Regel ist es völlig ausreichend, eine Behandlung von Tastaturereignissen (Key Events) und Mausereignissen (Mouse Events) zu programmieren.

Die möglichen Key und Mouse events sind:

- keyPressed - Taste wurde gedrückt
- keyReleased - Taste wurde losgelassen
- keyTyped - Taste wurde gerdrückt und wieder losgelassen

- mousePressed - Maustaste wurde gedrückt
- mouseReleased - Maustaste wurde losgelassen
- mouseClicked - Maustaste wurde gerdrückt und wieder losgelassen
- mouseEntered - Mauszeiger wurde in den Bereich der Komponente hinein bewegt
- mouseExited - Mauszeiger wurde aus dem Bereich der Komponente heraus bewegt

- Fügen Sie einen Key Event mit dem Ereignis "keyTyped" und einen Mouse Event mit dem Ereignis "mouseClicked" hinzu. Für den Key Event wird vom WindowBuider Editor in die get-Methode des Buttons folgender Quellcode eingefügt:

```
protected JButton getButRueckwaerts()
{
  if (butRueckwaerts == null)
  {
    butRueckwaerts = new JButton("r\u00FCckw\u00E4rts anzeigen");
    butRueckwaerts.addMouseListener(new MouseAdapter()
    {
      @Override
      public void mouseClicked(MouseEvent arg0)
      {
      }
    });
    butRueckwaerts.addKeyListener(new KeyAdapter()
    {
      @Override
      public void keyTyped(KeyEvent arg0)
      {
      }
    });
    butRueckwaerts.setBounds(10, 45, 230, 25);
  }
  return butRueckwaerts;
}
```

Zum Mouse Event gehört entsprechend folgender Quellcode:

```
protected JButton getButRueckwaerts()
{
  if (butRueckwaerts == null)
  {
    butRueckwaerts = new JButton("r\u00FCckw\u00E4rts anzeigen");
    butRueckwaerts.addMouseListener(new MouseAdapter()
    {
      @Override
      public void mouseClicked(MouseEvent arg0)
      {
      }
    });
    butRueckwaerts.addKeyListener(new KeyAdapter()
    {
      @Override
      public void keyTyped(KeyEvent arg0)
      {
      }
    });
    butRueckwaerts.setBounds(10, 45, 230, 25);
  }
  return butRueckwaerts;
}
```

Der im obigen Bildschirmausdruck markierte Quelltext enthält die **Ereignisbehandlung** zur jeweiligen Komponente sowie zum

ausgewählten Ereignis. Ein genaues Verständnis des erzeugten und oben markierte Quellcode ist nicht erforderlich. Wenn Sie sich dennoch für die Details zu diesem Quellcode interessieren, können Sie geeignete Erklärungen im Kapitel 3.5 (insbesondere in Kapitel 3.5.4) dieses Kurses nachlesen.

- Der vom WindowBuilder Editor erzeugte Quellcode muss jetzt so verändert werden, dass die für die Ereignisbehandlung vorgesehene Verarbeitung angestoßen wird. Hierzu muss die vorgesehene Verarbeitung in die Methoden keyTyped und mouseClicked eingefügt werden (siehe 13.2.3):

```
  public void keyTyped(KeyEvent arg0)
  {
// verarbeitende statische Methode aufrufen
    Verarbeitung.zeigeRueckwaertsAn(getMainPanel());
  }
```

Man sieht, dass die Methode `keyTyped(java.awt.event.KeyEvent arg0)` eine Methode einer namenlosen Klasse in der Klasse `MainPanel` ist. Da diese Klasse eine innere Klasse der Klasse `MainPanel` ist, kann in der Methode `keyTyped(java.awt.event.KeyEvent arg0)` auf die Instanzmethode `getMainPanel()` des Panels `MainPanel` zugegriffen werden. Das in diesem Kapitel vorgestellte Vorgehen ist somit eine sehr praktische und einfache Möglichkeit, die Referenz auf das Panel an die Verarbeitung weiterzugeben. Zudem ist diese Methode für jede Ereignisklasse sowie auf jedes Ereignis anwendbar.

13.2.3 Anbindung von verarbeitendem Quellcode an die Ereignisbehandlung

In der Verarbeitung zur Ereignisbehandlung wird in der Regel auf die Komponenten der GUI zugegriffen, um Eingaben einzulesen oder Ausgaben vorzunehmen. Wurde die GUI sowie die Ereignisbehandlung wie oben beschrieben mit Hilfe des WindowBuilder Editors erzeugt, sind zur Anbindung der Verarbeitung an die Ereignisbehandlung folgende Schritte erforderlich:

- **Übergabe der Referenz auf das Panel in der Ereignisbehandlung an die Verarbeitung**
  Gemäß Kapitel 13.2.2 kann die Referenz auf das Panel mit der GUI in der Ereignisbehandlung mit der Instanzmethode `getMainPanel()` des Panels ermittelt werden, da die Ereignisbehandlung in Form von inneren Klassen programmiert wird. Die von der Instanzmethode `getMainPanel()` zurückgegebene Referenz muss nun an die Verarbeitung weitergegeben werden. Man beachte dabei, dass beim Aufruf der Methode `getMainPanel()` der Zeiger this nicht codiert werden darf, da this in der Ereignisbehandlung auf die ereignisbehandelnde Klasse zeigt.

- **Aufruf der Verarbeitung zum Ereignis mit einer statischen Methode**
  Die Verwendung von statischen Methoden hat folgende Vorteile:

  - Die Verwendung von statischen Methoden (und statischen Variablen) ermöglicht es, dass aus verschiedenen Panels auf das gleiche Objekt zur Ereignisverarbeitung zugegriffen wird: Dadurch, dass sowohl die Methoden als auch die Variablen als statisch programmiert werden, kann auf diese Methoden und Variablen aus allen ereignisbehandelnden Programmabschnitten zugegriffen werden.
  - Der Einstieg in die Verarbeitung erfolgt statisch - wie bei Codierung einer main-Methode - und dieser Einstieg kann mit Hilfe eines Struktogramms - unter Beachtung des Prinzip der Trennung von Steuerung und Verarbeitung erfolgen. Mit anderen Worten kann die Verarbeitung nach den gleichen Prinzipien entworfen werden wie ein separates Konsolprogramm.

  Im vorliegenden Fall geschieht der Aufruf der Verarbeitung in der Ereignisbehandlung mit der Übergabe der Referenz auf das Panel wie folgt:

```
  public void keyTyped(KeyEvent arg0)
  {
// verarbeitende statische Methode aufrufen
    Verarbeitung.zeigeRueckwaertsAn(getMainPanel());
  }
```

Man beachte, dass der statischen Methode `zeigeRueckwaertsAn(MainPanel mp)` die Referenz auf das Panel als Parameter mit übergeben wird. Hierdurch wird es möglich, dass in der Verarbeitung eine Ein- und Ausgabe in der GUI erfolgen kann.

13.2.4 Grundlegende Schritte der Verarbeitung zu Ereignissen

Die Verarbeitung wurde im Beispiel jpr13002 wie folgt codiert:

```
package bsp13002;

public class Verarbeitung
{
// Deaktivieren des parameterlosen Konstruktors
  private Verarbeitung() { }
```

```
// Methode zur Ereignisbehandlung
  public static void zeigeRueckwaertsAn(MainPanel mp)
  {
    String ein = mp.getTfEingabe().getText();
    StringBuffer sb = new StringBuffer(ein);
    sb.reverse();
    String aus = sb.toString();
    mp.getTfEingabe().setText(aus);
  }
}
```

Im Quellcode von Beispiel jpr13002 sind alle - auch im Allgemeinen wichtigen - Schritte vorhanden:

1. **Deaktiveren des parameterlosen Konstruktors** - der parameterlose Konstruktor wird auf Sichtbarkeit private gesetzt, damit eine Objektbildung nicht mehr möglich ist. Gemäß obigem Konzept sollte ja der Aufruf einer verarbeitenden Methode aus der Ereignisbehandlung stets statisch erfolgen.

2. **Codierung der verarbeitenden Methode als statische Methode** - die verarbeitende Methode `zeigeRueckwaertsAn(MainPanel mp)` wurde gemäß dem erläuterten Konzept als statisch gekennzeichnet.

3. **Übergabe der Referenz auf das Panel an die verarbeitende Methode** - damit in der verarbeitenden Methode eine Ein- oder Ausgabe erfolgen kann, wird die Referenz auf das Panel mit der GUI an die verarbeitende Methode übergeben.

4. **Durchführung von Ein- und Ausgabevorgängen in der verarbeitenden Methode** - die für die Verarbeitung benötigten Daten werden in der verarbeitenden Methode aus der GUI gelesen:

   ```
   String ein = mp.getTfEingabe().getText();
   ```

Man beachte bei der Ein- und Ausgabe:

Eine Ein- oder Ausgabe in der GUI erfolgt stets im String-Format.

Entsprechend wird zunächst auf die Komponente der GUI zugegriffen, aus der gelesen werden soll:

```
javax.swing.JTextField tfEingabe = mp.getTfEingabe();
```

Anschließend wird aus der Komponente der eingegebene Text gelesen:

```
String ein = tfEingabe.getText();
```

Diese beiden Codezeilen können auch wie in Beispiel jpr13002 zu einer einzigen Programmzeile zusammengezogen werden:

```
String ein = mp.getTfEingabe().getText();
```

Analog zur Eingabe erfolgt eine Ausgabe ebenfalls in der Verarbeitung sowie über die Referenz auf das Panel. Im vorliegenden Beispiel jpr13002 wird der auszugebende Text in das bereits für das Einlesen verwendete JTextField geschrieben, für das die zugehörige get-Methode wie bereits oben erwähnt den Namen `getTfEingabe()` hat. Entsprechend geschieht die Ausgabe wie folgt:

```
String aus = ...;
mp.getTfEingabe().setText(aus);
```

Diese Ausgabe in der markierten Zeile ist entstanden durch Zusammenziehen der nachfolgenden beiden Programmzeilen zu einer einzigen Zeile:

```
javax.swing.JTextField tfEingabe = mp.getTfEingabe();
tfEingabe.setText(aus);
```

Die Ereignisbehandlung und die Verarbeitung zu diesem Ereignis geschieht somit insgesamt wie folgt (vgl. Beispiel jpr13002:)

Ereignisbehandlung in dem Panel mit der GUI:

```
package bsp13002;

import ...

public class MainPanel extends JPanel
{

  ...
  private JTextField tfEingabe;
  private JButton butRueckwaerts;

  ...
```

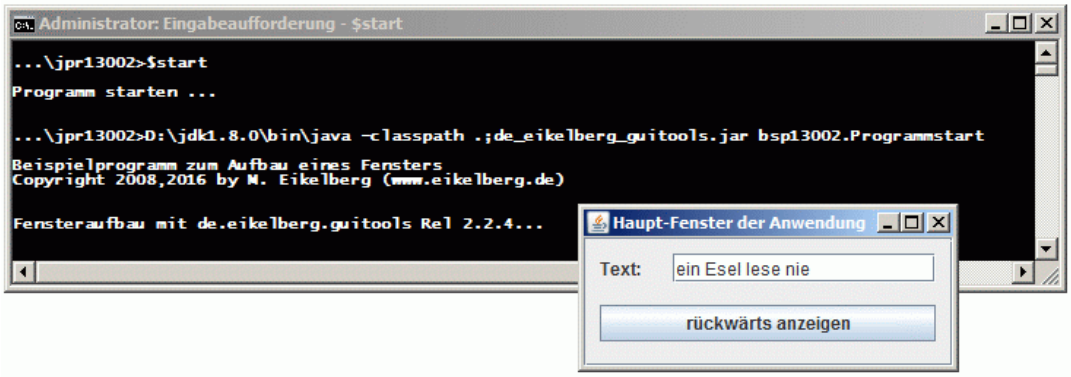
```
  /**
   * Create the panel.
   */
  public MainPanel()
  {

    initialize();
  }

  private void initialize()
  {
    setSize(new Dimension(250, 85));
    setPreferredSize(new Dimension(250, 85));
    setLayout(null);
    add(getLbTextEingabe());
    add(getTfEingabe());
    add(getButRueckwaerts());
  }

// manuell eingefügt
  private MainPanel getMainPanel()
  {
    return this;
  }

  protected JLabel getLbTextEingabe()
  {
    if (lbTextEingabe == null)
    {
      lbTextEingabe = new JLabel("Text:");
      lbTextEingabe.setBounds(10, 10, 40, 20);
    }
    return lbTextEingabe;
  }

  protected JTextField getTfEingabe()
  {
    if (tfEingabe == null)
    {
      tfEingabe = new JTextField();
      tfEingabe.setBounds(60, 10, 180, 20);
      tfEingabe.setColumns(10);
    }
    return tfEingabe;
  }

  protected JButton getButRueckwaerts()
  {
    if (butRueckwaerts == null)
    {
      butRueckwaerts = new JButton("r\u00FCckw\u00E4rts anzeigen");
      butRueckwaerts.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseClicked(MouseEvent arg0)
        {
          // verarbeitende statische Methode aufrufen
          Verarbeitung.zeigeRueckwaertsAn(getMainPanel());
        }
      });
      butRueckwaerts.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyTyped(KeyEvent arg0)
        {
          // verarbeitende statische Methode aufrufen
          Verarbeitung.zeigeRueckwaertsAn(getMainPanel());
        }
      });
      butRueckwaerts.setBounds(10, 45, 230, 25);
    }
    return butRueckwaerts;
```

```
  }
}
```

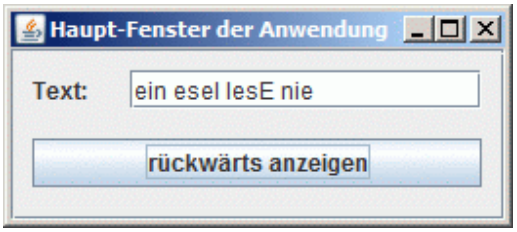
Verarbeitung zum Ereignis in einer verarbeitenden Klasse:

```
package bsp13002;

public class Verarbeitung
{
// Deaktivieren des parameterlosen Konstruktors
  private Verarbeitung() { }

// Methode zur Ereignisbehandlung
  public static void zeigeRueckwaertsAn(MainPanel mp)
  {
    String ein = mp.getTfEingabe().getText();
    StringBuffer sb = new StringBuffer(ein);
    sb.reverse();
    String aus = sb.toString();
    mp.getTfEingabe().setText(aus);
  }
}
```

Beispielprogramm jpr13002 dreht somit für einen im JTextField eingebenen Text nach einem Mausklick oder einem Tastendruck auf dem Button "rückwärts anzeigen" die Reihenfolge der im JTextField eingegebenen Schriftzeichen um:



Nach Klick auf den Button "rückwärts anzeigen" wird der eingebene Text rückwärts angezeigt:



Rezept zur Erstellung von Programmen mit grafischer Benutzeroberfläche gemäß Kap. 13

1. Einfügen der jar-Datei `de_eikelberg_guitools.jar` in das Eclipse-Projekt.
2. Anlegen eines Verzeichnisses `config` im Java-Basisverzeichnis. Kopieren der Datei `fenster.properties` in das Verzeichnis `config`.
3. Anlegen einer von javax.swing.JPanel abgeleiteten Klasse mit dem WindowBuilder Editor, z. Bsp. mit dem Namen MainPanel.
4. Einstellen der Eigenschaften `size` und `prefferedSize` von MainPanel auf den gleichen Wert, nämlich die Größe des Panels.
5. Einstellen der Eigenschaft `layout` von MainPanel, in dem dieser Eigenschaft der Wert `null` zugewiesen wird.
6. Einfügen der Methode
   `private MainPanel getMainPanel() { return this; }`
   in der Klasse MainPanel.
7. Anlegen einer Klasse mit einer main-Methode, z. Bsp. mit dem Namen Programmstart.
8. Einfügen einer Import-Anweisung für die Klasse de.eikelberg.guitools.ESinglePanelFrame in die Klasse Programmstart.

9. Einfügen folgender Schritte in die Klasse Programmstart:
    - Festlegen der Fensterbeschriftung:
      `String titel = "...";`
    - Instanziieren des MainPanels:
      `MainPanel mp = new MainPanel();`
    - Instanziieren des ESinglePanelFrame-Fensters:
      `new ESinglePanelFrame(mp,titel);`
10. Anlegen einer Klasse für die verarbeitenden Methoden, z. Bsp. mit dem Namen `Verarbeitung`.
11. Überschreiben des parameterlosen Konstruktors der Klasse `Verarbeitung` mit einem private parameterlosen Konstruktor.

Ein als Vorlage zu verwendendes Projekt, in dem alle obigen Schritte bereits ausgeführt sind, finden Sie bei den Musterlösungen im Projekt jpr13100. In diesem Muster-Projekt ist in der Methode initialize() der Klasse MainPanel bereits ein modales Fenster eingefügt, das eine Startmeldung anzeigt (vgl. Kap. 13.3).

Bei der Programmierung ist zu beachten:

- Auf die dem WindowBuilder Editor erzeugte Komponenten der GUI wird mit einer get-Methode zugegriffen. Soll aus der entsprechenden Komponente im Rahmen der Verarbeitung gelesen oder soll in die Komponenten geschrieben werden, so muss die Sichtbarkeit dieser get-Methode auf protected eingestellt sein. Dies ist bei Anwendung der obigen Anleitung gewährleistet.
- Die Verarbeitung erfolgt mit statischen Methoden der Klasse `Verarbeitung`. Diesen Methoden muss die Referenz auf das MainPanel, gespeichert in der Instanzvariablen `mp` der Klasse MainPanel, als Parameter übergeben werden.

13.3 Modale Dialogfenster

13.3.1 Modale Fenster

Modale Standard-Dialogfenster spielen eine wesentliche Rolle bei der Erstellung von Programmen mit grafischer Benutzeroberfläche. Sie werden zum Beispiel für Fehlermeldungen eingesetzt. Ferner kann man mit modalen Fenstern auch sogenannte "Assistenten" programmieren, die einen Benutzer durch einen Vorgang führen.

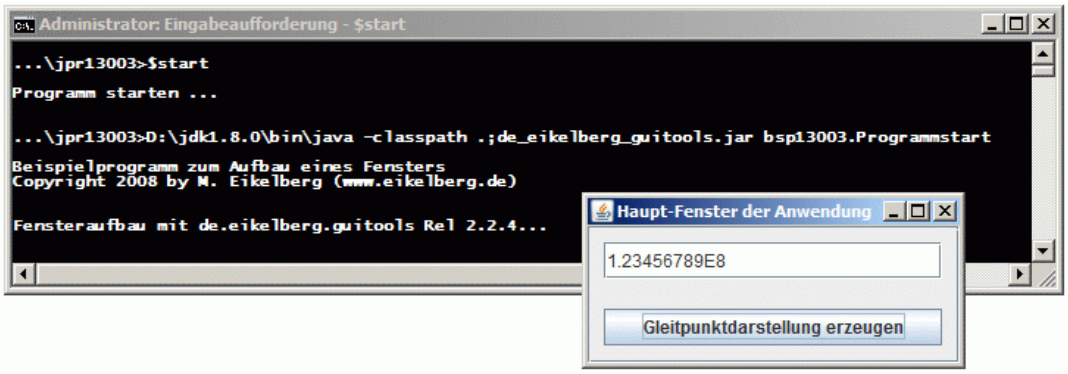
> Ein Fenster heißt **modal** bezüglich eines zweiten (übergeordneten) Fensters, wenn der Zugriff auf das übergeordnete Fenster solange gespert ist, wie das modale Fenster angezeigt wird. Erst nach Schließen des modalen Fensters wird der Zugriff auf das übergeordnete Fenster wieder möglich.

Modale Fenster können vom Programmierer selber erstellt werden (vgl. Kap. 3.8.1 und 3.8.3) oder aber mit Hilfe von standard Java-Klassen erzeugt werden (vgl. Kap 3.8.2 und 2.8.4).

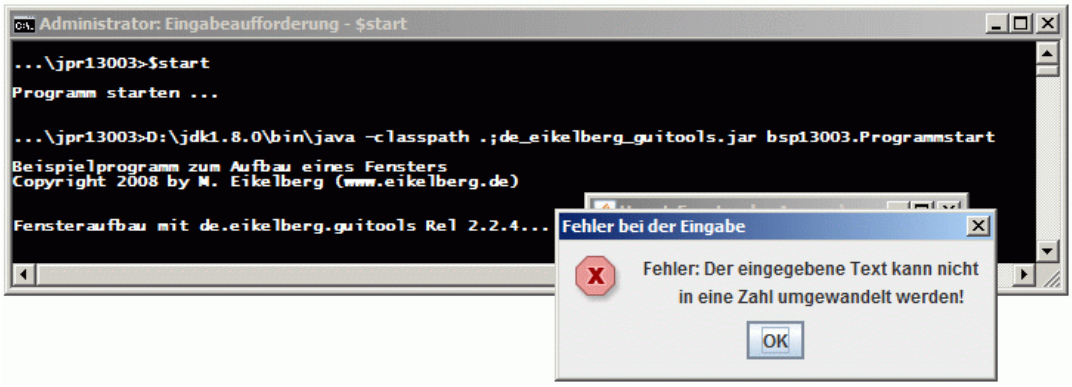
> Modale Standard-Fenster können mit den Klassen **javax.swing.JOptionPane** sowie **javax.swing.JFileChooser** erzeugt werden. Die Klasse JOptionPane stellt verschiedene Arten von Dialogfenstern zur Verfügung, die Klasse JFileChooser das sogenannte Dateiauswahlfenster

Erstes Beispiel für ein JOptionPane-Dialogfenster

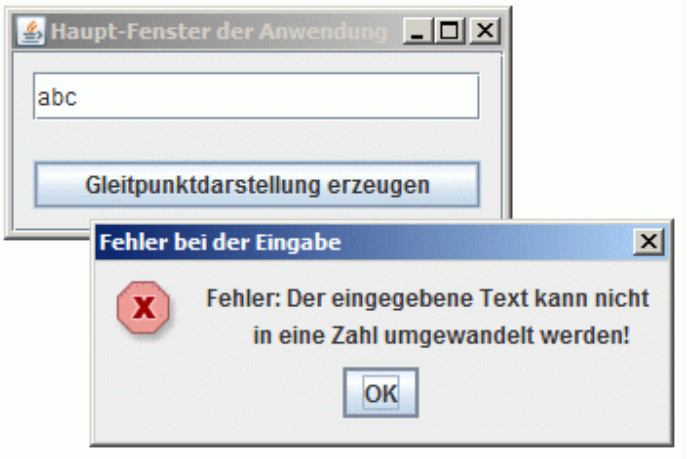
Das Beispielprojekt jpr13003 enthält ein einfaches GUI-Programm mit einem JOptionPane-Dialogfenster. Dieses Programm stellt ein Textfeld zur Verfügung. Wird in dieses Textfeld eine Zahl eingegeben, so wird diese Zahl in eine Gleitpunktzahl umgewandelt und in umgewandelter Form wieder im Textfeld angezeigt:



Wird dagegen keine Zahl eingegeben, so erscheint ein JOptionPane Standard-Fenster, das das Hauptfenster des Programms sperrt:



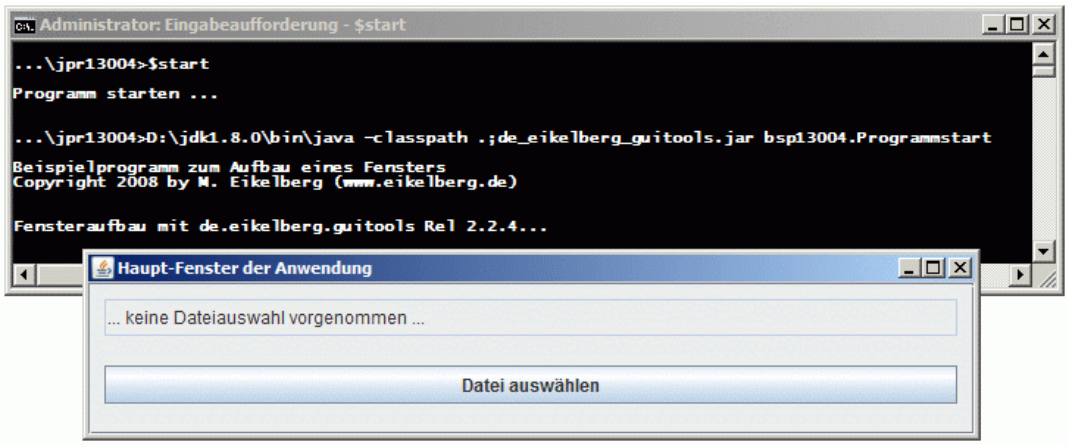
Die Sperrung des Hauptfensters kann man deutlicher an der grauen Einfärbung der Kopfzeile des Hauptfensters erkennen, wenn man das JOptionPane-Fenster etwas zur Seite bewegt:



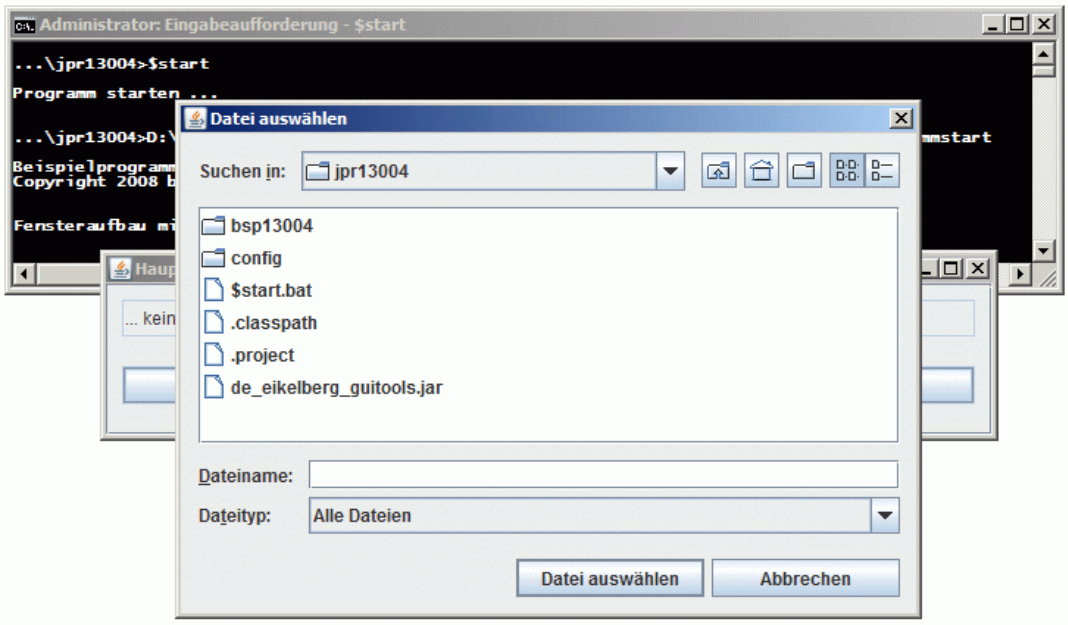
Die Programmierung von JOptionPane-Dialogfenstern wird in Kapitel 13.3.2 im Detail erklärt.

Erstes Beispiel für ein JFileChooser-Dialogfenster

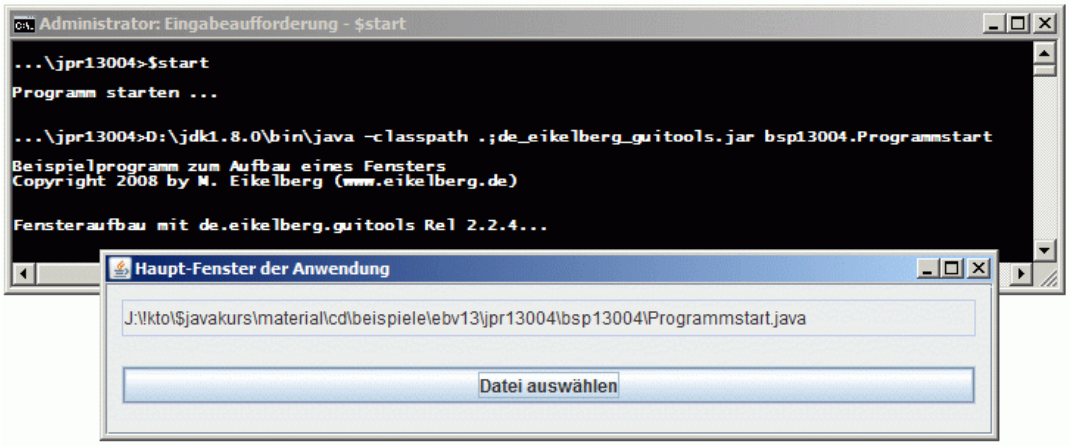
Beispiel jpr13004 ist ein erstes Beispiel für ein JFileChooser-Dialogfenster. Es öffnet zunächst ein Fenster mit einem Button "Datei auswählen".



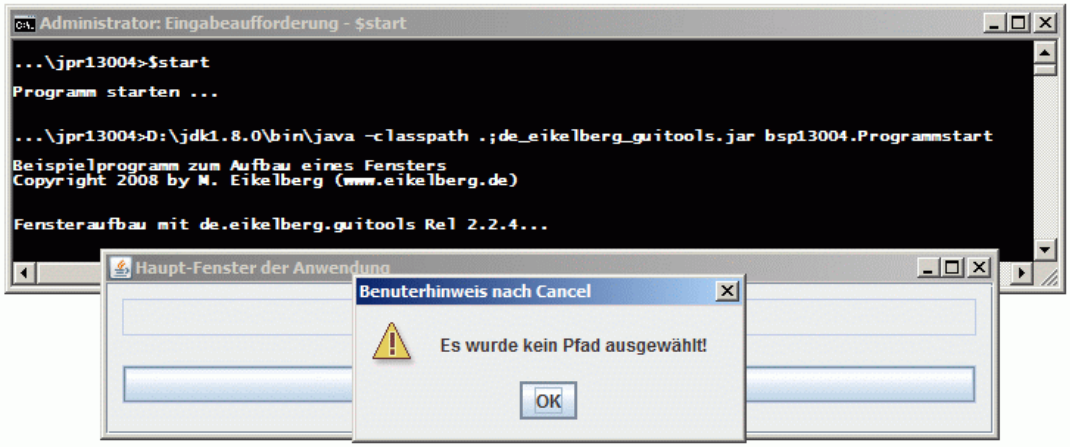
Nach Klick auf den Button "Datei auswählen" öffnet sich ein JFileChooser-Dialogfenster, in dem der Pfad zu einer Datei ausgewählt werden kann. Während der Anzeige dieses Fensters ist der Zugriff auf das Hauptfenster des Programms gesperrt, wie man wieder an der grauen Einfärbung der Kopfzeile des Fensters erkennen kann:



Wird Dateiauswahl vorgenommen, so wird der Pfad zur ausgewählten Datei nach Schließen des JFileChooser-Dialogfensters im Hauptfenster der Anwendung angezeigt:



Erfolgt keine Dateiauswahl, so wird ein JOptoinPanel-Dialogfenster mit einer Fehlermeldung angezeigt:



Die Programmierung von JFileChooser-Dialogfenstern wird in Kapitel 13.3.3 im Detail erklärt.

13.3.2 JOptionPane Dialogfenster

Standard-Dialogfenster vom Typ JOptionPane stellen einen sehr oft angewendeten Bestandteil des Swing-Paketes dar. Die wichtigsten Standard-Dialogfenster der Klasse JOptionPane sind

- **Meldungs-Dialog** - MessageDialog
- **Bestätigungs-Dialog** - ConfirmDialog
- **Optionsauswahl-Dialog** - OptionDialog
- **Eingabe-Dialog** - InputDialog

Alle Fenster der Klasse JOptionPane sind **modal**.

Grundsätzlich stellt die Klasse JOptionPane vollständig programmierte Dialogfenster zur Verfügung, die lediglich über Angaben in den Methoden konfiguriert werden, die die Fenster aufbauen. Die Methoden zum Aufbau der Dialogfenster sind sämtlich statische Methoden der Klasse JOptionPane und heißen wie folgt:

- **JOptionPane.showMessageDialog(...)** - erzeugt Meldungs-Dialogfenster
- **JOptionPane.showConfirmDialog(...)** - erzeugt Bestätigungs-Dialogfenster
- **JOptionPane.showOptionDialog(...)** - erzeugt Options-Dialogfenster
- **JOptionPane.showInputDialog(...)** - erzeugt Eingabe-Dialogfenster

Als Parameter der obigen vier statischen Methoden treten folgende Variablen und Konstanten auf:

- **Konstanten zur Festlegung des Meldungstyps**
  Der Meldungstyp hat ein Einfluss auf die Auswahl der von Java automatisch in die Dialogfenster eingefügten Symbole. Es gibt folgende Meldungstypen:
  - **JOptionPane.ERROR_MESSAGE**
  - **JOptionPane.INFORMATION_MESSAGE**
  - **JOptionPane.WARNING_MESSAGE**
  - **JOptionPane.QUESTION_MESSAGE**
  - **JOptionPane.PLAIN_MESSAGE**

  Diese fünf Konstanten sind vom Datentyp int, können also wie folgt verwendet werden:

  ```
  int meldungstypWahl = JOptionPane.ERROR_MESSAGE;
  ```

- **Konstanten zur Festlegung der Aktionsoption**
  Die Aktionsoption steuert das Einfügen von Buttons "YES", "NO", "CANCEL" und "OK" in die Dialogfenster. Es gibt folgende vier Aktionsoptionen:
  - **JOptionPane.DEFAULT_OPTION** - ergibt OK-Button und ggf. zusätzlich den CANCEL-Button
  - **JOptionPane.YES_NO_OPTION**
  - **JOptionPane.YES_NO_CANCEL_OPTION**
  - **JOptionPane.OK_CANCEL_OPTION**

  Auch die Aktionsoptionen sind vom Datentyp int und können daher wie folgt verwendet werden:

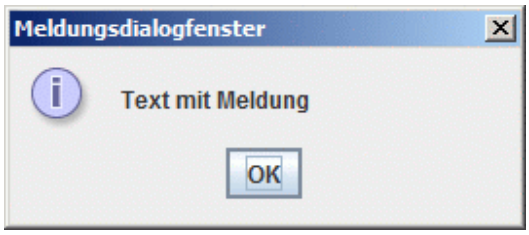
  ```
  int aktionsoptionsWahl = JOptionPane.YES_NO_CANCEL_OPTION;
  ```

Die Methoden für den Aufbau eines JOptionPane-Dialogfensters können bis zu 8 Parameter besitzen. Die für diese Parameter angegebenen Werte beeinflussen sich stellenweise gegenseitig, weshalb es nicht immer ganz einfach ist, die geeigneten Werte zu bestimmen. Das Beispielprogramm im Projekt jpr13005 (s.u.) hilft Ihnen bei der Auswahl der richtigen Parameter. Bevor auf dieses Programm eingegangen wird, sollen jedoch anhand von Projekt jpr13006 Beispiele zu den vier genannten Dialogfenster-Typen betrachtet werden:

- **Meldungs-Dialog**
  Beispiel-Programmcode aus bsp13006.Verarbeitung:

  ```
  JOptionPane.showMessageDialog(
    mp,                           // Referenz auf die zu sperrende Komponente
    "Text mit Meldung",           // Meldungstext
    "Meldungsdialogfenster",      // Fenstertitel
    JOptionPane.INFORMATION_MESSAGE
                                  // Meldungstyp, beeinflusst das Symbol
  );
  ```

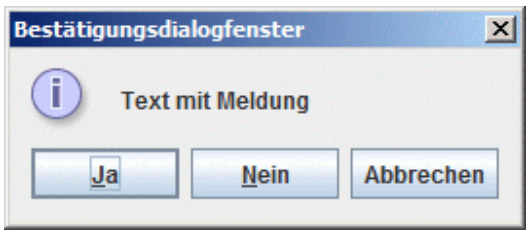
  Dieser Programmcode ergibt folgendes Dialogfenster:



- **Bestätigungs-Dialog**
  Beispiel-Programmcode aus bsp13006.Verarbeitung:

```
int eingabe = JOptionPane.showConfirmDialog(
    mp,                             // Referenz auf die zu sperrende Komponente
    "Text mit Meldung",             // Meldungstext
    "Bestätigungsdialogfenster", // Fenstertitel
    JOptionPane.YES_NO_CANCEL_OPTION,
                                    // Aktionsoption
    JOptionPane.INFORMATION_MESSAGE
                                    // Meldungstyp, beeinflusst das Symbol
);
if (eingabe == JOptionPane.CLOSED_OPTION) eingabe = 2;
                                    // Rückgabe -1 bei Schließen des Fensters
String[] antworten = {"Ja","Nein","Abbrechen"};
JOptionPane.showMessageDialog(
    mp,                             // Referenz auf die zu sperrende Komponente
    "Eingegeben wurde \""+antworten[eingabe]+"\"",
                                    // Meldungstext
    "Anzeige der Eingabe",          // Fenstertitel
    JOptionPane.INFORMATION_MESSAGE
                                    // Meldungstyp, beeinflusst das Symbol
 );
```

Da die Aktionsoption YES_NO_CANCEL_OPTION gesetzt wurde, werden im Dialogfenster Buttons mit den Beschriftungen "Ja", "Nein" und "Abbrechen" angezeigt:



Der Rückgabewert eines Bestätigungsdialogfensters ist eine int Zahl, deren Wert gleich Null, Eins oder Zwei ist. Das Schließen des Dialogfensters führt allerdings zur Rückgabe des Wertes -1, der in obigem Programmcode in den Wert 2 umgesetzt wird. Anschließend wird im obigen Programmabschnitt der Rückgabewert mit Hilfe eines String-Arrays in Text umgesetzt und entsprechend in einem Meldungsdialogfenster angezeigt:

- **Optionsauswahl-Dialog**
  Beispiel-Programmcode aus bsp13006.Verarbeitung:

```
String[] optionen = {"Option 1","Option 2","Option 3","Option 4"};
int eingabe = JOptionPane.showOptionDialog(
    mp,                             // Referenz auf die zu sperrende Komponente
    "Text mit Meldung",             // Meldungstext
    "Optionsdialogfenster",         // Fenstertitel
    JOptionPane.YES_NO_CANCEL_OPTION,
                                    // Aktionsoption - ohne Wirkung (!)
    JOptionPane.INFORMATION_MESSAGE,
                                    // Meldungstyp, beeinflusst das Symbol
```

```
        null,                           // Standard-Symbole verwenden
        optionen,                       // String-Array mit den anzuzeigenden Optionen
        optionen[1]                     // vorausgewählt einzustellende Option
    );
    String ausgabe = "";
    if (eingabe == JOptionPane.CLOSED_OPTION)
                                        // Rückgabe -1 bei Schließen des Fensters
      ausgabe = "Es ist keine Auswahl erfolgt";
    else
      ausgabe = "Ausgewählt wurde "+optionen[eingabe];
    JOptionPane.showMessageDialog(
        mp,                             // Referenz auf die zu sperrende Komponente
        ausgabe,                        // Meldungstext
        "Anzeige der Auswahl",          // Fenstertitel
        JOptionPane.INFORMATION_MESSAGE
                                        // Meldungstyp, beeinflusst das Symbol
    );
```

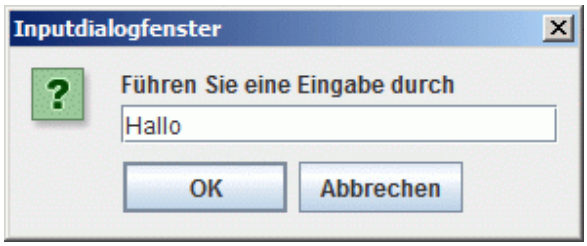
Im Optionsdialogfenster werden die im String-Array optionen genannten Optionen angezeigt. Voreingestellt wird die Option mit dem Index 1, also die Option 2. Entsprechend erhält man folgende Anzeige:

Das Einlesen der Benutzereingaben erfolgt wieder über den int Rückgabewert der Methode, der im obigen Beispiel auf der int Variablen eingabe gespeichert wird. Beim Schließen des Fensters über den Schließen-Button im Fensterkopf ist der Rückgabewert wieder gleich -1, was bei der Codierung des Meldungstextes in obigem Beispiel berücksichtigt wurde. Wird eine Option ausgewählt, so wird diese im nachfolgenden Meldungsdialogfenster angezeigt:

- **Eingabe-Dialog ohne Vorgaben:**
  Beispiel-Programmcode aus bsp13006.Verarbeitung:

```
    String eingabe = JOptionPane.showInputDialog(
        mp,                             // Referenz auf die zu sperrende Komponente
        "Führen Sie eine Eingabe durch",
                                        // Meldungstext
        "Inputdialogfenster",           // Fenstertitel
        JOptionPane.QUESTION_MESSAGE
                                        // Meldungstyp, beeinflusst das Symbol
    );
//  Auswerten der Eingabe
    String ausgabe = "";
    if (eingabe == null)
      ausgabe = "Es ist keine Auswahl erfolgt!";
    else
      ausgabe = "Eingegeben wurde \""+eingabe+"\"";
    JOptionPane.showMessageDialog(
        mp,                             // Referenz auf die zu sperrende Komponente
        ausgabe,
                                        // Meldungstext
        "Anzeige der Eingabe",          // Fenstertitel
        JOptionPane.INFORMATION_MESSAGE
                                        // Meldungstyp, beeinflusst das Symbol
    );
```

Das Meldungsdialogfenster enthält ein Fragezeichen-Symbol, da der Meldungstyp QUESTION_MESSAGE eingestellt wurde:

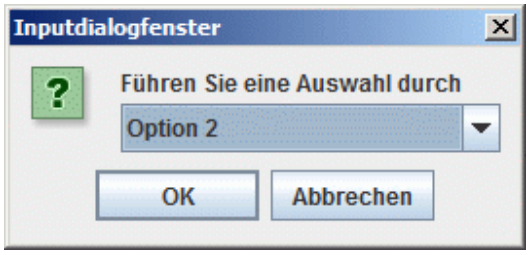


Die Rückgabe der Benutzereingabe erfolgt als Objekt der Klasse String. Beim Schließen des Fensters über den Schließen-Button im Fensterkopf ist der Rückgabewert wieder gleich dem Null-Pointer, was bei der Codierung des Meldungstextes in obigem Beispiel berücksichtigt wurde. Wird ein Text eingegeben, so wird dieser im nachfolgenden Meldungsdialogfenster angezeigt:

- **Eingabe-Dialog mit Vorgaben:**

```
Beispiel-Programmcode aus bsp13006.Verarbeitung:
      String[] optionen = {"Option 1","Option 2","Option 3","Option 4"};
      Object eingabe = JOptionPane.showInputDialog(
          mp,                           // Referenz auf die zu sperrende Komponente
          "Führen Sie eine Auswahl durch",
                                        // Meldungstext
          "Inputdialogfenster",         // Fenstertitel
          JOptionPane.QUESTION_MESSAGE,
                                        // Meldungstyp, beeinflusst das Symbol
          null,                         // Standard-Symbole verwenden
          optionen,                     // String-Array mit den anzuzeigenden Optionen
          optionen[1]                   // vorausgewählt einzustellende Option
      );
// Auswerten der Eingabe
      String ausgabe = "";
      if (eingabe == null)
        ausgabe = "Es ist keine Auswahl erfolgt!";
      else
        ausgabe = "Ausgewählt wurde \""+eingabe.toString()+"\"";
      JOptionPane.showMessageDialog(
          mp,                           // Referenz auf die zu sperrende Komponente
          ausgabe,                      // Meldungstext
          "Anzeige der Auswahl",        // Fenstertitel
          JOptionPane.INFORMATION_MESSAGE
                                        // Meldungstyp, beeinflusst das Symbol
      );
```

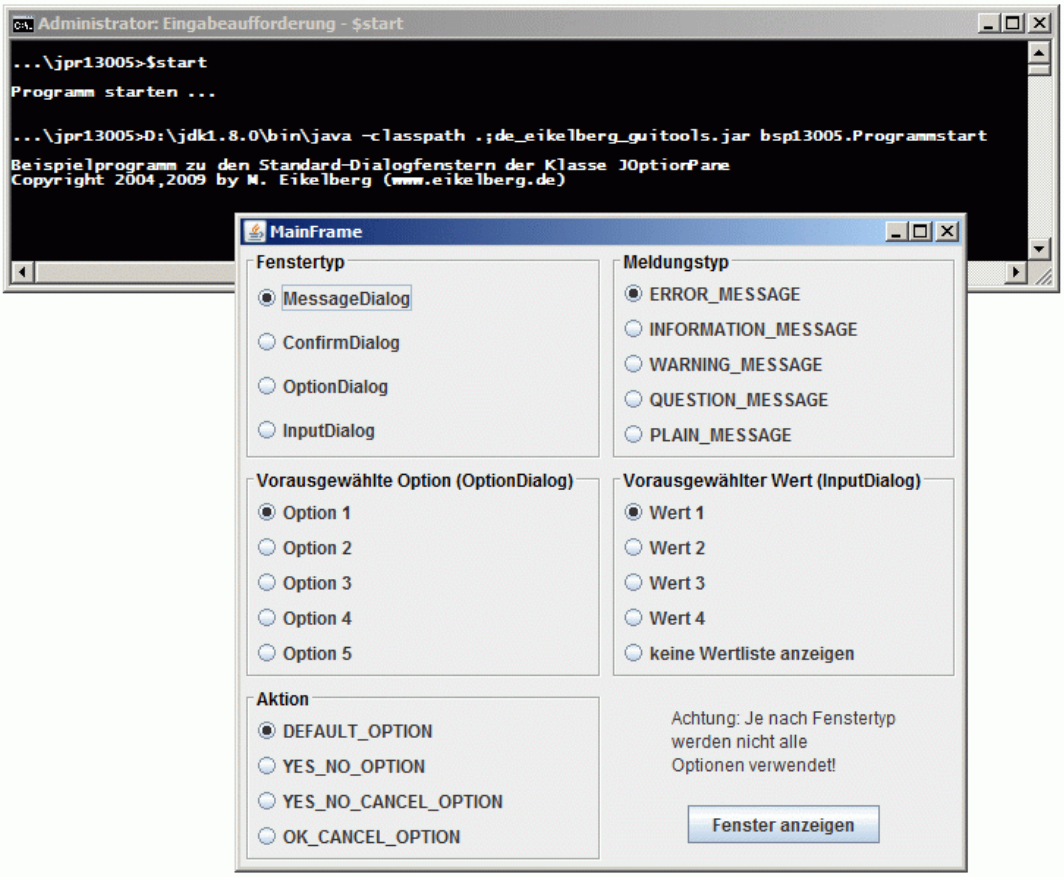
Die im String-Array optionen gespeicherten Werte werden in einer Combobox zur Auswahl angezeigt. Voreingestellt ist die Option mit Index 1, also die Option 2:



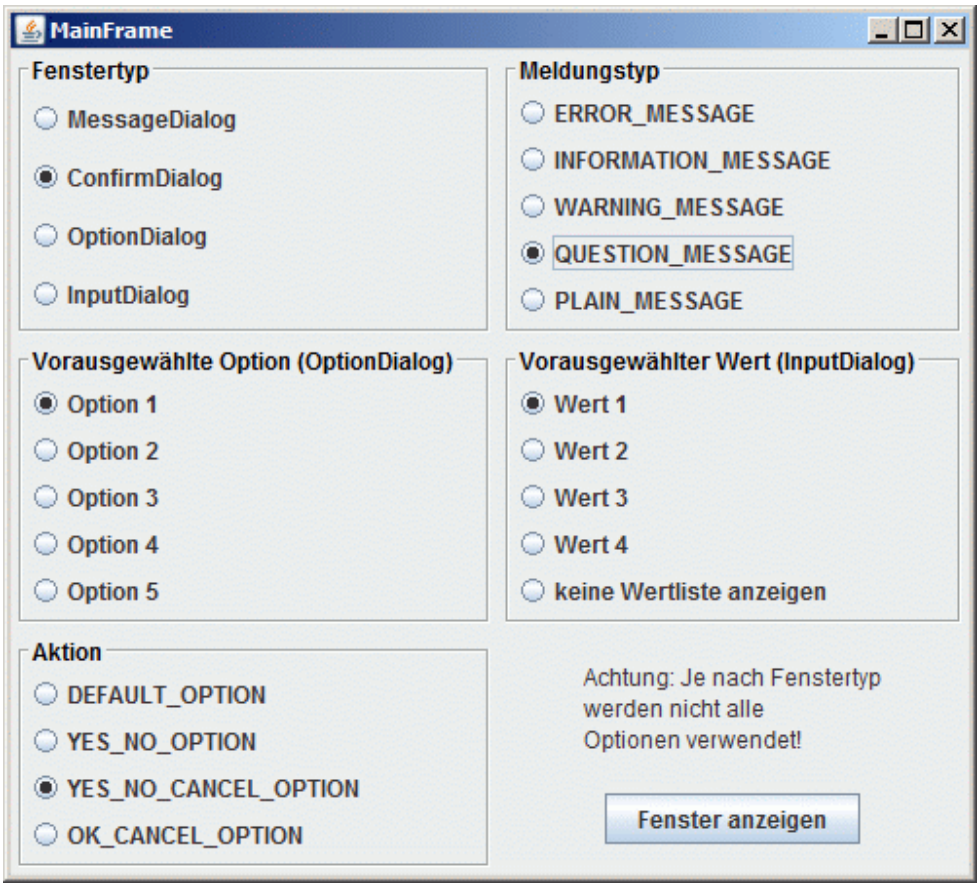
Die Rückgabe der Benutzereingabe erfolgt als Objekt der Klasse Object. Beim Schließen des Fensters über den Schließen-Button im Fensterkopf ist der Rückgabewert wieder gleich dem Null-Pointer, was bei der Codierung des Meldungstextes in obigem Beispiel berücksichtigt wurde. Wird eine Auswahl vorgenommen, so wird diese Answahl im nachfolgenden Meldungsdialogfenster angezeigt:

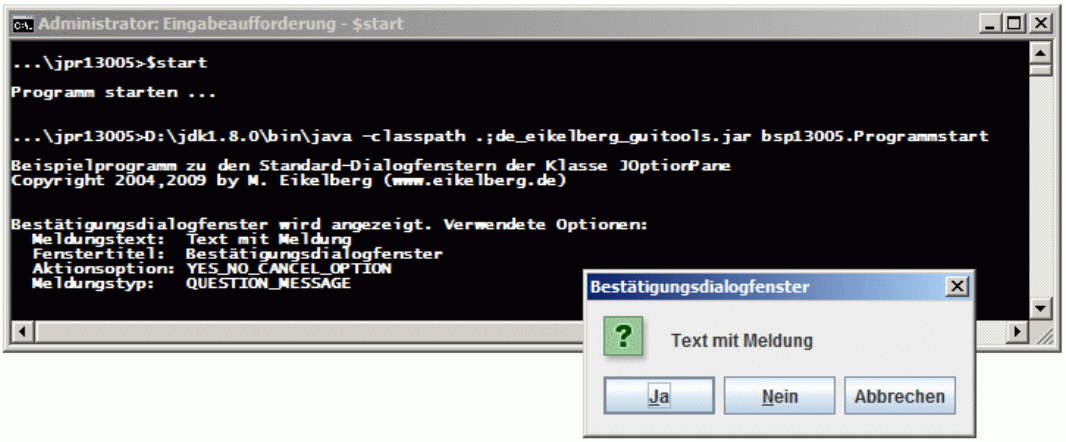
Hinweis: Die verschiedenen Typen von JOptionPane-Dialogfenstern können mit Hilfe des Programms im Projekt jpr13005 erkundet werden. Die GUI zu dem Programm im Projekt jpr13005 ist nicht unter Verwendung der in diesem Kapitel erklärten Methoden programmiert worden, sondern mit den in Kapitel 3 erklärten Techniken. Das Beispiel jpr13005 dient lediglich dazu, verschiedene Arten von JOptionPane Dialogfenstern zu erzeugen und die jeweils erforderlichen Parameter (in der Konsolbox) anzuzeigen.

Die Wirkung der Optionen der Klasse JOptionDialog und das Aussehen der jeweils erzeugen Fenster kann mit Hilfe des Beispielprojektes `jpr13005` getestet werden. Das Programm in diesem Projekt wird wie üblich gestartet über die Batch-Datei $start.bat und die Klasse bsp13005.Programmstart. Nach dem Programmstart erhält man das folgende Fenster:



In diesem Fenster können die bei der Anzeige des Dialogfensters zu verwendenden Optionen gewählt werden. Nach Klick auf den Button "Fenster anzeigen" wird ein entsprechendes Dialogfenster angezeigt. Ferner wird in der Konsolbox eine Meldung ausgegeben, welche der eingestellten Optionen für die Konfiguration des Dialogfensters verwendet wurden:


MainFrame
Fenstertyp
MessageDialog
ConfirmDialog
OptionDialog
InputDialog
Meldungstyp
ERROR_MESSAGE
INFORMATION_MESSAGE
WARNING_MESSAGE
QUESTION_MESSAGE
PLAIN_MESSAGE
Vorausgewählte Option (OptionDialog)
Option 1
Option 2
Option 3
Option 4
Option 5
Vorausgewählter Wert (InputDialog)
Wert 1
Wert 2
Wert 3
Wert 4
keine Wertliste anzeigen
Aktion
DEFAULT_OPTION
YES_NO_OPTION
YES_NO_CANCEL_OPTION
OK_CANCEL_OPTION
Achtung: Je nach Fenstertyp werden nicht alle Optionen verwendet!
Fenster anzeigen


Administrator: Eingabeaufforderung - $start
...\jpr13005>$start
Programm starten ...
...\jpr13005>D:\jdk1.8.0\bin\java -classpath .;de_eikelberg_guitools.jar bsp13005.Programmstart
Beispielprogramm zu den Standard-Dialogfenstern der Klasse JOptionPane
Copyright 2004,2009 by M. Eikelberg (www.eikelberg.de)
Bestätigungsdialogfenster wird angezeigt. Verwendete Optionen:
Meldungstext: Text mit Meldung
Fenstertitel: Bestätigungsdialogfenster
Aktionsoption: YES_NO_CANCEL_OPTION
Meldungstyp: QUESTION_MESSAGE
Bestätigungsdialogfenster
Text mit Meldung
Ja
Nein
Abbrechen

13.3.3 JFileChooser Dateidialogfenster

Die Klasse JFileChooser stellt in einfacher Weise Dialogfenster für die Dateiauswahl bereit. Zunächst erzeugt man ein Objekt der Klasse JFileChooser, wobei ein Pfad angegeben werden kann, der beim Öffnen des Dateidialogfensters eingestellt wird. Anschließend können die Dialogfenster mit Hilfe der folgenden Instanzmethoden als **modale** Fenster erzeugt werden:

- **int showOpenDialog(Component parent)** - öffnet Dateidialogfenster mit Beschriftungen zum Öffnen einer Datei
- **int showSaveDialog(Component parent)** - öffnet Dateidialogfenster mit Beschriftungen zum Schließen einer Datei
- **int showDialog(Component parent,String vorgangsbeschreibung)** - öffnet Dateidialogfenster mit Beschriftung vorgangsbeschreibung

Die Komponente parent gibt dabei das Panel (bzw. allgemeiner die Komponente) an, dessen Fenster als **Besitzerfenster** des erzeugten Dialogfensters modal gesetzt werden soll. Der Rückgabewert der Methoden ist vom Typ int und gibt an, ob eine Auswahl vorgenommen wurde. Dieser int-Wert kann mit folgenden Konstanten der Klasse JFileChooser verglichen werden:

- **JFileChooser.APPROVE_OPTION** - Datei wurde ausgewählt (Wert gleich 0).
- **JFileChooser.CANCEL_OPTION** - Vorgang wurde durch Click auf Cancel-Button abgebrochen (Wert gleich 1).
- **-1** - Dialogfenster wurde durch Klick auf Schließen-Button in der Kopfzeile des Fensters geschlossen.

Vor dem Erzeugen des Dialogfensters kann dieses in mehrfacher Weise mit Hilfe folgender Instanzmethoden bzw. Konstanten konfiguriert werden (die dabei verwendeten Klassen File und FileFilter werden weiter unten erklärt):

- **setCurrentDirectory(File file)** - positioniert die Dateiauswahl auf den im File-Objekt file (s.u.) angegebenen Pfad.
- **setSelectedFile(File file)** - markiert die im File-Objekt angegebene Datei als vorausgewählt.
- **setMultiSelectionEnabled(boolean wahr)** - erlaubt die gleichzeitige Auswahl mehrerer Dateien. Standardmäßig ist nur die Auswahl einer einzigen Datei erlaubt.
- **setFileFilter(FileFilter fFilter)** - ermöglicht die Angabe einer oder mehrerer Gruppen von Dateitypen. Der im FileFilter-Objekt fFilter (s.u.) angegebene Name zur Beschreibung des Filters wird im Dialogfenster neben dem Label "Files of Type" in einer JComboBox angezeigt. Bei Auswahl eines solchen Filters werden nur die Dateien im Dialogfenster angezeigt, die im FileFilter-Objekt zugelassen wurden.
- **setFileSelectionMode(int auswahltyp)** - konfiguriert, ob Dateien und/oder Verzeichnisse ausgewählt werden können. Als int-Zahl auswahltyp können die folgenden Konstanten eingesetzt werden:
  - **JFileChooser.FILES_ONLY** - ermöglicht nur die Auswahl von Dateien.
  - **JFileChooser.FILES_DIRECTORIES** - ermöglicht nur die Auswahl von Verzeichnissen.
  - **JFileChooser.FILES_AND_DIRECTORIES** - ermöglicht nur die Auswahl von Dateien und Verzeichnissen.
- **File getSelectedFile()** - gibt die Auswahl zurück bei Auswahl *einer einzigen* Datei bzw. *eines einzigen* Verzeichnisses. Rückgabewert ist ein Objekt der Klasse File.
- **File[] getSelectedFile()** - gibt die Auswahl zurück bei Auswahl *mehrerer* Dateien bzw. Verzeichnisse. Rückgabewert ist ein Array von Objekten der Klasse File.

Es folgen die notwendigen Informationen zur Klasse **java.io.File**:

- Mit Hilfe der Klasse **java.io.File** werden Dateien verwaltet.
- Ein File-Objekt wird wie folgt instanziiert:

  ```
  File file = new File(String pfad)
  ```

  Dabei kann die Variable Pfad einen absoluten oder auch einen relativen Pfad enthalten. Enthält das File-Objekt einen in einem JFileChooser-Objekt ausgewählten Pfad, so ist dieser Pfad absolut.
- Den im File-Objekt gespeicherten **Pfad** erhält man wie folgt:

  ```
  file.getPath()
  ```

- Den im File-Objekt gespeicherten **Dateinamen** erhält man wie folgt:

  ```
  file.getName()
  ```

Die zweite bei der Verwendung von JFileChooser benötigte Klasse ist die Klasse **javax.swing.filechooser.FileFilter**. Diese wird wie in folgendem Beispiel codiert:

```
package bsp13004;

import java.io.File;
// nicht das Interface java.io.FileFilter verwenden, sondern:
import javax.swing.filechooser.FileFilter;

/**
```

```
 * Title:        Beispielprogramm zu Dateiauswahlfenstern
 * Description:  Dateiauswahlfilter
 ...
 */

public class MyFileFilter1 extends FileFilter
{
// Überschreiben der Methode accept - es werden die Dateitypen angegeben,
// die angezeigt werden sollen
  public boolean accept(File f)
  {
    boolean anzeigeOK;
// html-Dateien sollen angezeigt werden
    anzeigeOK = f.getName().toLowerCase().endsWith(".html");
// xml-Dateien sollen angezeigt werden
    anzeigeOK = anzeigeOK || f.getName().toLowerCase().endsWith(".xml");
// java-Dateien sollen angezeigt werden
    anzeigeOK = anzeigeOK || f.getName().toLowerCase().endsWith(".java");
// Verzeichnisse sollen angezeigt werden
    anzeigeOK = anzeigeOK || f.isDirectory();
// Wahrheitswert zurückgeben
    return anzeigeOK;
  }

// Festlegen des Namens für die im FileFilter definierte Gruppe zugelassener
// Dateitypen - wird im Dateidialogfenster angezeigt
  public String getDescription()
  {
    return "Web-Dateien";
  }
}
```

Im Beispiel jpr13004 wird dieser sowie ein weiterer analog programmierter FileFilter mit Namen bsp13004.MyFileFilter2 verwendet. Der Aufbau des Dateidialogfensters sowie die Auswertung der Benutzereingaben erfolgt in der Klasse bsp13004.Verarbeitung in der folgenden statischen Methode

```
public static void zeigeDateiauswahlFensterAn(MainPanel mp).
```

Das angezeigte Dateidialogfenster wird bezüglich des Panels bsp13004.MainPanel modal gesetzt.

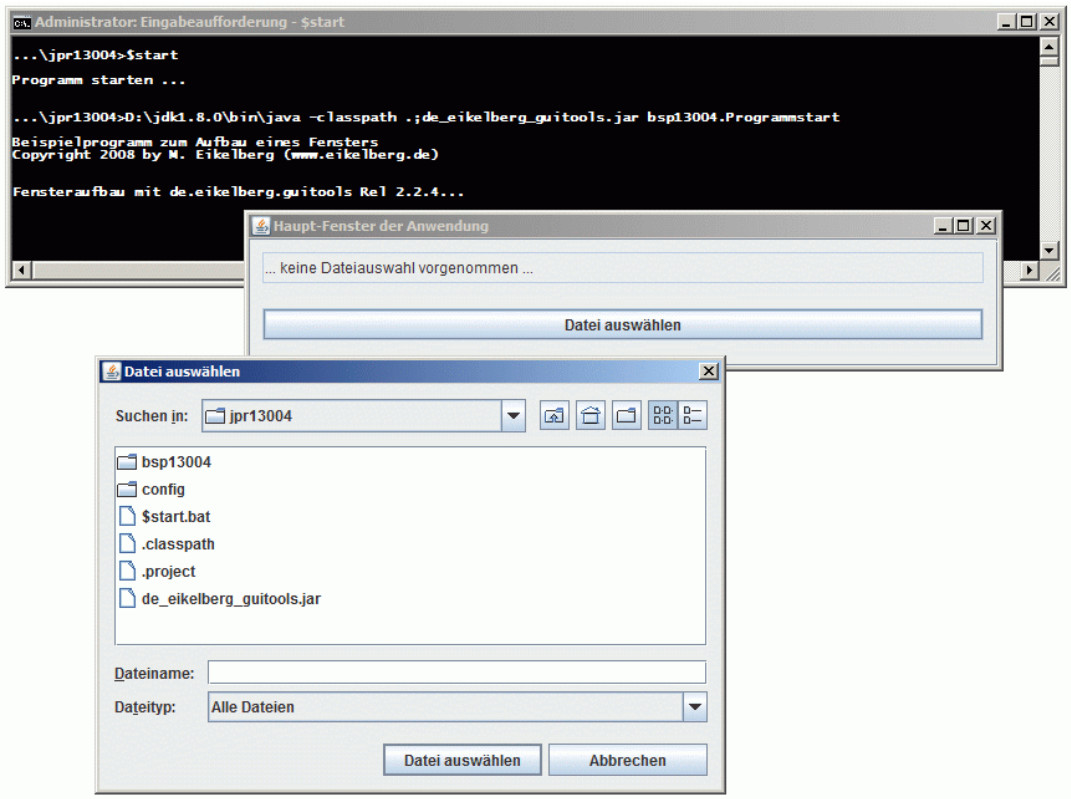
In der Klasse bsp13004.Verarbeitung ist die für das Dateiauswahlfenster verwendete Pfadangabe in einer statischen Variablen mit dem Namen `pfad` gespeichert. Diese Variable wird mit der Pfadangabe "." initialisiert und zeigt damit auf das Java-Basisverzeichnis jpr13004. Als statische Variable der Klasse bsp13004.Verarbeitung ist die Pfadangabe somit eine Eigenschaft des gesamten Programms. In der Methode `zeigeDateiauswahlFensterAn(MainPanel mp)` wird der Inhalt dieser Variablen gleich der Benutzerauswahl gesetzt, die im Dateidialogfenster durchgeführt wurde:

```
package bsp13004;

import javax.swing.JFileChooser;
...

public class Verarbeitung
{
// Deaktivieren des parameterlosen Konstruktors
  private Verarbeitung() { }

// Klassenvariable für die Verarbeitung
  private static String pfad = ".";

//   Methode zur Ereignisbehandlung
  public static void zeigeDateiauswahlFensterAn(MainPanel mp)
  {
//  Verarbeitung
//  JFileChooser-Objekt instanziieren, wobei der Focus auf den Pfad in der
//  Variablen Pfad eingestellt wird.
     JFileChooser myJFC = new JFileChooser(pfad);
//  Zwei Dateiauswahlfilter zuordnen. Die Reihenfolge im Code bestimmt die Reihen-
//  folge bei der Anzeige
     myJFC.addChoosableFileFilter(new MyFileFilter2());
     myJFC.addChoosableFileFilter(new MyFileFilter1());  // der letzte ist voreingestellt
//  Nur die Auswahl von Dateien zulassen
     myJFC.setFileSelectionMode(JFileChooser.FILES_ONLY);
```

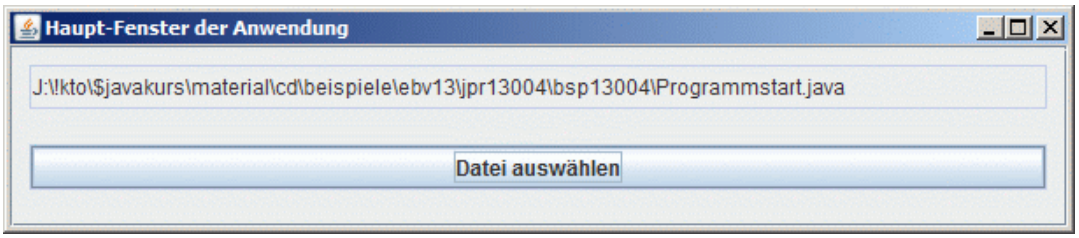
```
//  Dialogfeld zum Öffnen von Dateien öffnen als modales Fenster, wobei
//  das aktuelle Fenster das übergeordnete Fenster sei.
     int result = myJFC.showDialog(mp,"Datei auswählen");
//  Nach erfolgter Auswahl ausgewählte Datei feststellen. Dateidialogfenster sind
//  modal, also ist die Benutzereingabe nach Beendigung von showOpenDialog erfolgt.
     if (result==JFileChooser.APPROVE_OPTION)
     {
       pfad = myJFC.getSelectedFile().getPath();
//     Ausgabe in der GUI
       mp.getTfEingabe().setText(pfad);
     }
     else if (result==JFileChooser.CANCEL_OPTION)
     {
       pfad = ".";
//     Ausgabe in der GUI
       mp.getTfEingabe().setText("");
// Fehlermeldung
       ...
     }
     else
     {
       pfad = ".";
//     Ausgabe in der GUI
       mp.getTfEingabe().setText("");
//     Fehlermeldung
       ...
     }
   }
}
```

Da in der obigen Methode die Dateiauswahl mit `getSelectedFile().getPath()` ausgewertet wurde, wird der Dateiname incl. Pfad zurückgegeben.

Startet man das Beispielprogramm bsp13004, so erhält man nach Klick auf den Button "Datei auswählen" folgende Anzeige:



Anzeige des Haupt-Fensters der Anwendung jpr13004 nach einer erfolgreichen Dateiauswahl:


Haupt-Fenster der Anwendung
J:\!kto\$javakurs\material\cd\beispiele\ebv13\jpr13004\bsp13004\Programmstart.java
Datei auswählen

13.4 Programmierung von GUI Komponenten mit dem WindowBuilder Editor

13.4.1 Struktur des vom WindowBuilder Editor erzeugten Quellcode

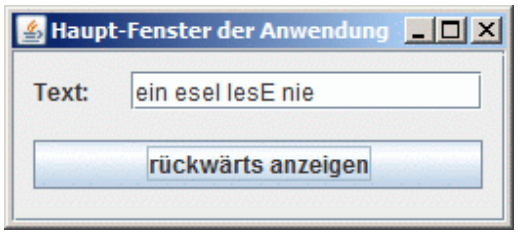
- In Kapitel 13.4 sind Hinweise zusammengestellt, wie nicht in Kapitel 13.1-13.3 genannte Komponenten der grafischen Benutzeroberfläche mit Hilfe des WindowBuilder Editors in die GUI eingefügt werden können.
- Für die weitergehende Programmierung von grafischen Benutzeroberflächen sowie die Programmierung komplexerer Komponenten der GUI ist es nützlich, die Struktur des vom WindowBuilder Editor erzeugten Quellcode zu kennen. Daher wird diese Struktur in Kapitel 13.4.1.1 erklärt.
- Einige Komponenten der GUI sollten in einem JPanel gruppiert werden (siehe Kapitel 13.4.1.2).
- Das Einfügen von separat programmierten JPanels in die GUI wird in Kapitel 13.4.1.3 erläutert.

13.4.1.1 Drei Grundbestandteile einer GUI-Komponente im WindowBuilder Quellcode

Wie bereits in Kapitel 13.1 und 13.2 erklärt besteht - abgesehen von einem JLabel - der WindowBuilder Quellcode zu einer GUI-Komponente aus folgenden drei Grundbestandteilen:

1. Deklaration der GUI-Komponente als **private Instanzvariable** des JPanels, in dem die Komponente angeordnet ist
2. **get-Methode** zur GUI-Komponente, in der die Komponente beim ersten Aufruf der Komponente instanziiert sowie durch weitere Anweisungen konfiguriert wird
3. Einfügung der Komponente durch Aufruf einer **add-Methode** in der Instanzmethode initialize() des JPanels.

Die grafische Benutzeroberfläche des bereits in Kapitel 13.1-2 behandelten Beispiels jpr13002 enthält ein JLabel, ein JTextField und einen JButton:



Für den in der GUI enthaltenen JButton finden sich die drei Grundbestandteile einer GUI-Komponente im WindowBuilder Quellcode wie folgt im Quellcode der von JPanel abgeleiteten Klasse bsp13002.MainPanel:

```
package bsp13002;

import javax.swing.JPanel;
import javax.swing.JButton;
...

public class MainPanel extends JPanel
{
  ...
// 1. Deklaration der GUI-Komponente als private Instanzvariable des JPanels
  private JButton butRueckwaerts;
  ...

  public MainPanel()
  {
    ...
    initialize();
  }

  private void initialize()
  {
    ...
// 3. Einfügung der Komponente durch Aufruf einer add-Methode der Klasse JPanel
    add(getButRueckwaerts());
    ...
  }
```

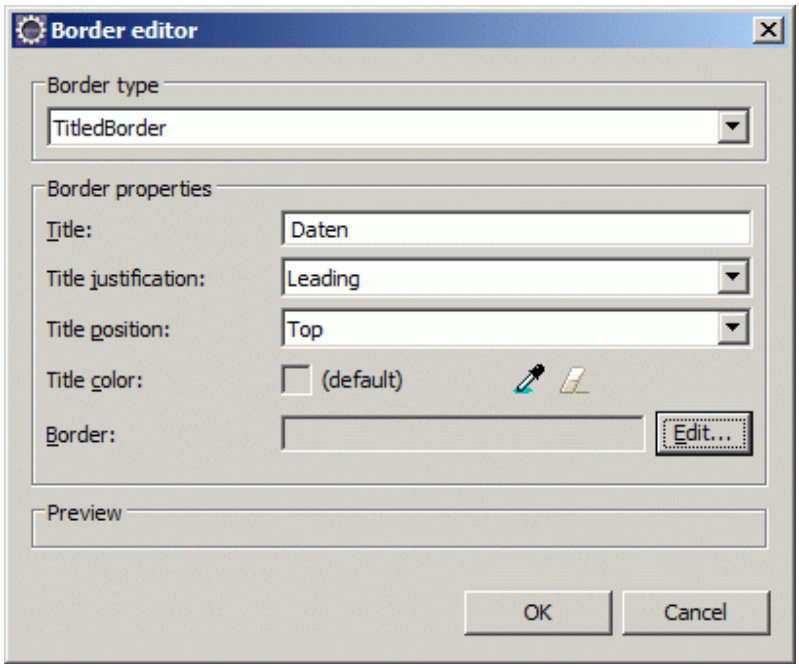
```
  ...

// 2. get-Methode zur GUI-Komponente
  protected JButton getButRueckwaerts()
  {
    if (butRueckwaerts == null)
    {
      butRueckwaerts = new JButton();
      butRueckwaerts = new JButton("r\u00FCckw\u00E4rts anzeigen");
      butRueckwaerts.setBounds(10, 45, 230, 25);
      butRueckwaerts.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseClicked(MouseEvent arg0)
        {
          // verarbeitende statische Methode aufrufen
          Verarbeitung.zeigeRueckwaertsAn(getMainPanel());
        }
      });
      butRueckwaerts.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyTyped(KeyEvent arg0)
        {
          // verarbeitende statische Methode aufrufen
          Verarbeitung.zeigeRueckwaertsAn(getMainPanel());
        }
      });
    }
    return butRueckwaerts;
  }
}
```

13.4.1.2 Gruppierung von GUI-Komponenten in untergeordneten Panels

Für die optische Gestaltung - z. Bsp. insbesondere bei Checkboxen und Radiobuttons - ist es oft sinnvoll, Komponenten der GUI in Panels zu gruppieren, die in das Panel mit der GUI des gesamten Programms - im Folgenden **Hauptpanel** genannt - als **untergeordnetes Panel** eingebettet werden. Für das Erzeugen eines solchen untergeordneten Panels mit dem WindowBuilder können Sie wie folgt vorgehen:

1. Öffnen Sie das Hauptpanel mit dem WindowBuilder Editor.
2. Öffnen Sie im WindowBuilder in der Palette den Abschnitt Containers.
3. Markieren Sie im Abschnitt Containers den Eintrag JPanel und fügen Sie das JPanel wie eine normale Komponente der Benutzeroberfläche in die GUI des Hauptpanels ein.
4. Benennen Sie das Panel mit einem Namen, der mit `pan` beginnt.
5. Markieren Sie das untergeordnete Panel, um Eigenschaften für dieses Panel einzustellen.
6. Konfigurieren Sie für das untergeordnete Panel geeignete Werte für die Variable Bounds.
7. Konfigurieren Sie zum untergeordneten Panel für die Eigenschaft layout den Wert null.
8. Konfigurieren Sie für das untergeordnete Panel bei Bedarf einen Wert für die Eigenschaft border:
    - Klicken Sie auf den Button "..." rechts neben der Eigenschaft border und öffnen Sie so den Border Editor von WindowBuilder.
    - Stellen Sie als Border Type den Typ TitledBorder ein und füllen Sie die Eigenschaften des betitelten Randes wie folgt:
    - Tragen Sie als Title einen geeigneten Text als Beschriftung des Randes ein.
    - Für die Title Justification können Sie den Wert Leading und für die Title Position den Wert Top eingetragen.
    - Für die Title color können Sie die Standardeinstellungen verwenden.
    - Im Feld Border sollten Sie keine Einstellung vornehmen.

    Insgesamt erhalten Sie folgende Konfiguration:



9. Fügen Sie wie gewohnt Komponenten in das untergeordnete Panel ein, indem Sie diese mit dem WindowBuilder im Bereich des untergeordneten Panels anordnen.

Bei Einfügung eines JLabels sowie eines JTextFields in das untergeordnete Panel entsteht Quelltext wie in folgendem Ausschnitt aus der Klasse bsp13007.MainPanel.

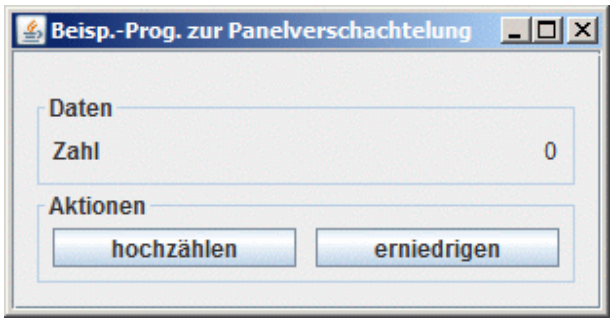
- Das untergeordnete JPanel wird vom WindowBuilder mit Quellcode wie zu einer einfachen Komponente erzeugt. Auch für das untergeordnete Panel sind die unter 13.4.1.1. genannten drei Grundbestandteile einer GUI-Komponente im WindowBuilder Quellcode vorhanden.
- Ferner sieht man, dass die im Bereich des untergeordneten Panels in der GUI angeordneten Komponenten nicht mehr in die Methode initialize() eingefügt wurden, sondern in analoger Weise statt in die Methode initialize() nunmehr in die get-Methode zum untergeordneten JPanel.
- Für das untergeordnete Panel muss die Layout-Eigenschaft erneut separat eingestellt werden - also wieder das null-Layout gesetzt werden.

Im nachfolgenden Ausschnitt aus dem Quellcode der Klasse bsp13007.MainPanel sind die oben genannten Punkte markiert:

```
package bsp13007;

import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
...

public class MainPanel extends JPanel
{

// 1. Deklaration des untergeordneten Panels als private Instanzvariable des Hauptpanels
  private JPanel panDaten;
  private JLabel lbZahl;
  private JTextField tfZahl;
  ...

  public MainPanel()
  {
    initialize();
  }

  private void initialize()
  {
    ...
// 3. Einfügung des untergeordneten Panels durch Aufruf einer add-Methode des Hauptpanels
    add(getPanDaten());
    ...
  }
```

```
// 2. get-Methode zum untergeordneten Panel
  private JPanel getPanDaten()
  {
    if (panDaten == null)
    {
      panDaten = new JPanel();
// Quellcode zur Konfiguration des untergeordneten Panels
      panDaten.setBorder(new TitledBorder(null, "Daten", TitledBorder.LEADING, TitledBorder.TOP,
null, null));
      panDaten.setBounds(10, 20, 280, 50);
      panDaten.setLayout(null);
// Enfügen der Komponenten in das untergeordnete Panel
      panDaten.add(getLbZahl());
      panDaten.add(getTfZahl());
    }
    return panDaten;
  }

  protected JLabel getLbZahl()
  {
    if (lbZahl == null)
    {
      lbZahl = new JLabel("Zahl");
      ...
    }
    return lbZahl;
  }

  protected JTextField getTfZahl()
  {
    if (tfZahl == null)
    {
      tfZahl = new JTextField();
      ...
    }
    return tfZahl;
  }

  ...
}
```

In der GUI des Programms jpr13007 findet man das untergeordnete Panel panDaten als eines der beiden untergeordneten Panels - man erkennt es an seinem Inhalt sowie an der Beschriftung "Daten":



13.4.1.3 Panelverschachtelung - Einfügen von separat programmierten Panels in ein Panel

13.4.1.3.1 Codierung eines untergeordneten Panels

Die in 13.4.1.2 eingeführten untergeordneten Panels können auch in einer eigenen Klasse codiert werden, was eine mehrfache Verwendung des Panels ermöglicht (z. Bsp. für eine Benutzeranmeldung innerhalb des Fensters - vgl. Kapitel 13.4.5.2). Ein solches in einer eigenen Klasse codiertes untergeordnetes Panel sollte wie folgt erzeugt werden:

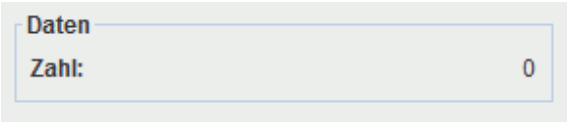
1. Anlegen des untergeordneten Panels als separates Panel in gleicher Weise, wie ein Hauptpanel angelegt wird.
2. Füllen des untergeordneten Panels mit GUI-Komponenten wie für ein Hauptpanel.
3. Codieren einer get-Methode, die die Referenz auf das untergeordnete Panel zurückgibt in der gleichen Weise wie eine entsprechende get-Methode auch in ein Hauptpanel eingefügt wird. Diese get-Methode wird wie beim Hauptpanel für die Codierung der Ereignisbehandlung zu Komponenten des untergeordneten Panels benötigt.

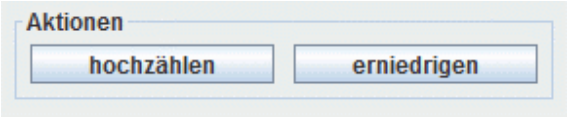
4. **Manuelles Einfügen von Quellcode für den Zugriff vom untergeordneten Panel aus auf die GUI außerhalb des untergeordneten Panels. Dieser Quellcode wird in das untergeordnete Panel eingefügt:**
    a. **Codieren einer private Instanzvariablen für die Speicherung der Referenz auf das übergeordnete Panel**
    b. **Speicherung der Referenz auf das übergeordnete Panel über eine set-Methode in der Instanzvariablen.**
    c. **Codierung einer get-Methode für den Zugriff auf diese Referenz.**
5. Codierung von Klassen für die Verarbeitung für jedes untergeordnete Panel analog zur Klasse Verarbeitung für das MainPanel.

Die Codierung eines untergeordneten Panels unterscheidet sich von der Codierung des obersten Panels (des Hauptpanels) gemäß Kapitel 13.1 bzw. 13.2 lediglich dadurch, dass der in obiger Auflistung genannte **Schritt 4** zusätzlich ausgeführt werden muss.

Hinweis: Es wäre sicher eine bessere Lösung zu Punkt 4, die Referenz auf das übergeordnete Panel über den Konstruktor an das untergeordnete Panel weiterzugeben. Dies ist allerdings bei Verwendung des WindowBuilder Editors (Rel. 1.8.0) nicht möglich, da dieser bei einem entsprechenden Versuch eine Exception wirft. Dieses Problem tritt im Übrigen auch bei anderen visuellen Editoren auf.

Beispiel jpr13008 enthält ein Programm, dessen GUI sich optisch nicht von der GUI von Beispiel jpr13007 unterscheidet, das jetzt aber zwei in separaten Klassen codierte untergeordnete Panels mit den Namen PanDaten (beschriftet mit dem Text "Daten") und PanAktionen (beschriftet mit dem Text "Aktionen") enthält:





13.4.1.3.2 Beispiel für ein untergeordnetes Panel

Das Panel **PanAktionen** aus obigem Bildschirmausdruck enthält zwei Buttons mit entsprechender Ereignisbehandlung. Es enthält u.a. Quellcode zu Schritt 4 aus der oben genannten Liste von Schritten zur Codierung eines untergeordneten Panels:

```
package bsp13008;

import ...

public class PanAktionen extends JPanel
{
  ...

  private JButton butHochzaehlen;
  private JButton butErniedrigen;

// Punkt 4: Drei manuelle Änderungen zur Verwaltung der Referenz auf das übergeordnete Panel
// 4a: Instanzvariable für die Referenz auf das übergeordnete Panel
  private MainPanel mp;
// 4b: get-Methode für die Referenz auf das übergeordnete Panel
  protected MainPanel getMainPanel()
  {
    return mp;
  }
// 4c: set-Methode für die Übergabe der Referenz auf das untergeordnete Panel
  protected void setMainPanel(MainPanel mp)
  {
    this.mp = mp;
  }

  public PanAktionen()
  {
    initialize();
  }

  private void initialize()
  {
    setBorder(new TitledBorder(null, "Aktionen", TitledBorder.LEADING, TitledBorder.TOP, null,
null));
    setPreferredSize(new Dimension(280, 50));
    setSize(new Dimension(280, 50));
    setLayout(null);
    add(getButHochzaehlen());
    add(getButErniedrigen());
  }
```

```
  private PanAktionen getPanAktionen()
  {
    return this;
  }

  protected JButton getButHochzaehlen()
  {
    if (butHochzaehlen == null)
    {
      butHochzaehlen = new JButton("hochz\u00E4hlen");
      butHochzaehlen.setBounds(10, 20, 125, 20);
      butHochzaehlen.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseClicked(MouseEvent arg0)
        {
          PanAktionenVerarbeitung.hochzaehlen(getPanAktionen());
        }
      });
      butHochzaehlen.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyTyped(KeyEvent arg0)
        {
          PanAktionenVerarbeitung.hochzaehlen(getPanAktionen());
        }
      });
    }
    return butHochzaehlen;
  }

  protected JButton getButErniedrigen()
  {
    if (butErniedrigen == null)
    {
      butErniedrigen = new JButton("erniedrigen");
      butErniedrigen.setBounds(145, 20, 125, 20);
      butErniedrigen.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseClicked(MouseEvent e)
        {
          PanAktionenVerarbeitung.erniedrigen(getPanAktionen());
        }
      });
      butErniedrigen.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyTyped(KeyEvent e)
        {
          PanAktionenVerarbeitung.erniedrigen(getPanAktionen());
        }
      });
    }
    return butErniedrigen;
  }
}
```

Man sieht, dass die Verarbeitung zum Panel PanAktionen in einer Klasse `PanAktionenVerarbeitung` angeordnet ist.

13.4.1.3.3 Herstellung der Bezüge zwischen den einzelnen Teilen der GUI bei der Codierung der Verarbeitung

Für die Bezüge zwischen den einzelnen Teilen der GUI wird somit folgendes Prinzip realisiert:

Bezüge in der Software von und in das untergeordnete Panel werden somit wie folgt aufgebaut:

- Das untergeordnete Panel nimmt wie folgt Bezug mit übergeordneten Teilen der GUI auf (vgl. 13.4.1.3.1):
  - Die Referenz auf das übergeordnete Panel wird über eine set-Methode an des untergeordnete Panel

weitergegeben.
  - Die Referenz wird in einer Instanzvariable des untergeordneten Panels gespeichert.
  - Für die Referenz auf das übergeordnete Panel wird eine passende get-Methode bereitgestellt, die den Zugriff auf das übergeordnete Panel ermöglicht.

- Das übergeordnete Panel nimmt Bezug auf mit Komponenten der untergeordneten Teile der GUI, indem es
  - zunächst auf das untergeordnete Panel zugreift und
  - anschließend die get-Methoden der GUI-Komponenten des untergeordneten Panels aufruft.

- Jedes Panel erhält eine eigene Klasse für die Ereignisbehandlung, damit eine flexible Verwendung der Panels möglich wird. Beim Aufruf der Ereignisbehandlung wird immer zunächst die Referenz auf das aktuelle Panel, also für die Ereignisbehandlung zum untergeordneten Panel die Referenz auf das untergeordnete Panel mitgegeben.

Die Ereignisbehandlung für die Ereignisse zu den Buttons im Panel PanAtionen erfolgt mit den statischen Methoden der Klasse **PanAktionenVerarbeitung**. Dort wird die Anwendung des obigen Prinzips für den Zugriff zwischen den Panels sichtbar:

```
package bsp13008;

public class PanAktionenVerarbeitung
{
  ...

  protected static void hochzaehlen(PanAktionen sp)
  {
    int zahl;
// Einlesen
    zahl = getZahl(sp);
// Verarbeitung
    zahl = zahl+1;
// Ausgabe
    setZahl(sp, zahl);
  }

  protected static void erniedrigen(PanAktionen sp)
  {
    int zahl;
// Einlesen
    zahl = getZahl(sp);
// Verarbeitung
    zahl = zahl-1;
// Ausgabe
    setZahl(sp, zahl);
  }

  private static int getZahl(PanAktionen sp)
  {
    int zahl=0;
// Auf die Referenz des übergeordneten Panels zugreifen
    MainPanel mp = sp.getMainPanel();
// Auf die Referenz des anderen untergeordneten Panels zugreifen
    PanDaten pd = mp.getPanDaten();
// Aus dem anderen untergeordneten Panel lesen
    String ein = pd.getTfZahl().getText();
    try
    {
      zahl = Integer.parseInt(ein);
    }
    catch (NumberFormatException e)
    {
// es kann nichts passieren, da keine Benutzereingabe erfolgt
    }
    return zahl;
  }

  private static void setZahl(PanAktionen sp, int zahl)
  {
// Auf die Referenz des übergeordneten Panels zugreifen
    MainPanel mp = sp.getMainPanel();
// Auf die Referenz des anderen untergeordneten Panels zugreifen
    PanDaten pd = mp.getPanDaten();
```

```
// In das andere untergeordnete Panel schreiben
    pd.getTfZahl().setText(String.valueOf(zahl));
  }
}
```

13.4.1.3.4 Einfügen von untergeordneten Panels in ein übergeordnetes Panel

Das Einfügen der untergeordneten Panels in das übergeordnete Panel mit dem WindowBuilder Editor geschieht wie folgt:

- Öffnen Sie in der Palette des WindowBuilder Editors den Abschnitt System und und klicken Sie auf den Eintrag "Choose component". Es öffnet sich das Fenster "Open type".
- Tragen Sie im Fenster "Open type" im Feld "Select a type to add...) den Namen des einzufügenden untergeordneten Panels ein. Der Paketname wird automatisch ergänzt, es reicht also den Namen der Klasse einzugeben.
- Das einzufügende Panel wird angezeigt und kann wie eine standard Swing-Komponente in die GUI eingefügt werden.
- Markieren Sie das untergeordnete Panel und wechseln sie in den Bereich Properties. Tragen Sie wie für eine standard Swing-Komponente die gewünschten Koordinaten für das Einfügen des untergeordneten Panels in der Variablen Bounds ein. Dabei sollten Sie als Werte für width und height die gleichen Werte eintragen wie für sie für width und height in den Variablen size und preferredSize des untergeordneten Panels eingesetzt wurden.
- Vergeben Sie wie für eine standard Swing-Komponente einen Namen für das untergeordnete Panel, der mit den Buchstaben **pan** beginnen sollte.
- Im übergeordneten Panel wird vom WindowBuilder Editor für das untergeordnete Panel Quellcode wie zu einer standard Swing-Komponente erzeugt. *Dieser Quellcode muss an einer Stelle angepasst werden: In der get-Methode für das untergeordnete Panel muss die set-Methode des untergeordneten Panels aufgerufen werden, mit der gemäß Kapitel 13.4.1.3.1 die Referenz auf das übergeordnete Panel im untergeordneten Panel gespeichert wird.* Im vorliegenden Beispiel muss also z. Bsp. im übergeordneten (Haupt)Panel MainPanel in der get-Methode zum untergeordneten Panel PanAktionen der im folgenden Quellcode markierte Aufruf der set-Methode setMainPanel(MainPanel) eingefügt werden:

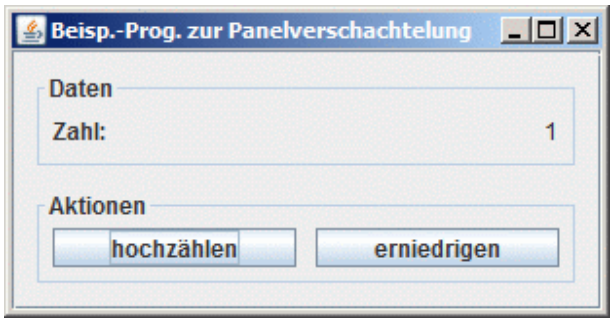
```
...

public class MainPanel extends JPanel
{
  ...

  protected PanAktionen getPanAktionen()
  {
    if (panAktionen == null)
    {
      panAktionen = new PanAktionen();
      panAktionen.setBounds(10, 70, 280, 50);
// manuell eingefügt:
      panAktionen.setMainPanel(this);
    }
    return panAktionen;
  }

  ...
}
```

Im übergeordneten Panel entsteht daraufhin insgesamt der folgende Quellcode, der sich für jedes einzufügende untergeordnete Panel jeweils nur an einer Stelle von Quellcode mit standard Swing-Komponenten in der GUI unterscheidet:

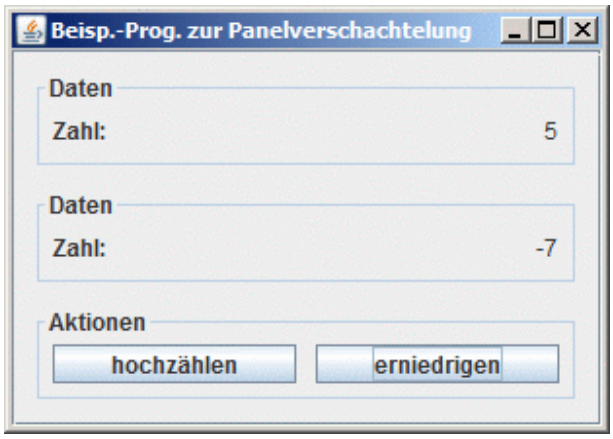
```
package bsp13008;

import ...

public class MainPanel extends JPanel
{
  ...

  private PanAktionen panAktionen;
  private PanDaten panDaten;

  public MainPanel()
  {
    initialize();
  }

  private void initialize()
  {
    ...
```

```
    add(getPanAktionen());
    add(getPanDaten());
    ...
  }

// manuell eingefügt
  private MainPanel getMainPanel()
  {
    return this;
  }

  protected PanAktionen getPanAktionen()
  {
    if (panAktionen == null)
    {
      panAktionen = new PanAktionen();
      panAktionen.setBounds(10, 70, 280, 50);
// Übergabe der Referenz auf das MainPanel von Hand eingefügt:
      panAktionen.setMainPanel(this);
    }
    return panAktionen;
  }

  protected PanDaten getPanDaten()
  {
    if (panDaten == null)
    {
      panDaten = new PanDaten();
      panDaten.setBounds(10, 10, 280, 50);
// Übergabe der Referenz auf das MainPanel von Hand eingefügt:
      panDaten.setMainPanel(this);
    }
    return panDaten;
  }
}
```

Insgesamt ergibt das Beispiel jpr13008 ein Programm mit der gleichen GUI wie das Beispiel jpr13007.



Allerdings sind jetzt im Quellcode die untergeordneten Panels in separaten Klassen codiert. Sie können entsprechend jetzt auch mehrfach verwendet werden, wie Beispiel jpr13009 zeigt - dort erfolgt das Hochzählen im oberen Daten-Panel und das Erniedrigen im unteren Daten-Panel:



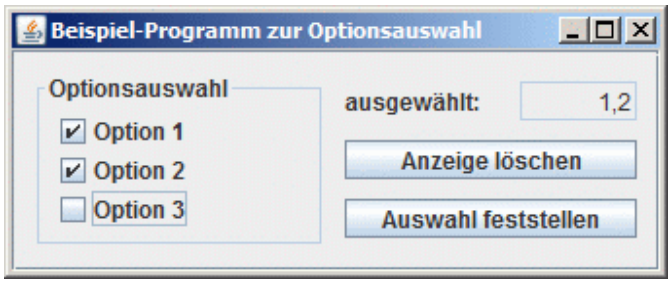
13.4.2 Beschriftungs- und Bedienelemente

Die zu den Beschriftungs- und Bedienelementen gehörenden Komponenten JLabel und JButton wurden schon in Kapitel 13.1 vorgestellt. Weitere Komponenten dieser Gruppe sind JCheckBox, JRadioButton, JSlider und JProgressbar.

- Alle in diesem Kapitel vorgestellten Komponenten werden mit dem WindowBuilder Editor wie ein JLabel oder ein JButton erzeugt. Es entsteht Quellcode wie in Kapitel 13.4.1.1 erklärt. Da die get-Methode für das Einfügen der Komponente in die GUI immer gleichförmig codiert wird, wird diese im Weiteren in der Regel nicht mehr erwähnt. Von dem generierten Quellcode wird somit nur die Deklaration der Komponente sowie die zugehörige get-Methode vorgestellt.
- Für einige Komponenten ist es sinnvoll, diese wie in Kapitel 13.4.1.2 erläutert in einem Panel zusammenzufassen.
- Für einige Komponenten sind manuell codierte Ergänzungen im Quellcode erforderlich. Diese sind jeweils kenntlich gemacht.

13.4.2.1 JCheckBox

Beispiel jpr13110 zeigt die Codierung von Komponenten vom Typ JCheckBox. Diese Komponente verwendet man, wenn voneinander unabhängige Optionen oder Einstellungen eingelesen werden sollen. Wenn diese Optionen inhaltlich zusammenhängen, sollte man diese Komponenten wie in vorliegendem Beispiel in einem Panel mit Beschriftung und Rahmen gruppiert anzeigen:



Der Quellcode besteht aus

- drei Komponenten vom Typ JCheckBox sowie
- einer Komponente vom Typ JPanel

Entsprechend gibt es im Quellcode des Hauptpanels bsp13110.MainPanel folgende vier Deklarationen mit jeweils einer zugehörigen get-Methode:

```
package bsp13110;

import javax.swing.JPanel;
import javax.swing.JCheckBox;
...

public class MainPanel extends JPanel
{
  ...

  private JPanel panCHB;
  private JCheckBox chbOption1;
  private JCheckBox chbOption2;
  private JCheckBox chbOption3;

  ...

  protected JPanel getPanCHB()
  {
    if (panCHB == null)
    {
      panCHB = new JPanel();
      panCHB.setBorder(
        new TitledBorder(
            null,
            "Optionsauswahl",
            TitledBorder.LEADING,
```

```
            TitledBorder.TOP,
            null,
            null));
      panCHB.setBounds(10, 10, 150, 90);
      panCHB.setLayout(null);
      panCHB.add(getChbOption1());
      panCHB.add(getChbOption2());
      panCHB.add(getChbOption3());
    }
    return panCHB;
  }

  protected JCheckBox getChbOption1()
  {
    if (chbOption1 == null)
    {
      chbOption1 = new JCheckBox("Option 1");
      chbOption1.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent arg0)
        {
          Verarbeitung.bestimmeOptionsAuswahl(getMainPanel());
        }
      });
      chbOption1.setSelected(true);
      chbOption1.setActionCommand("1");
      chbOption1.setBounds(10, 20, 130, 20);
    }
    return chbOption1;
  }

  protected JCheckBox getChbOption2()
  {
    ...
  }

  protected JCheckBox getChbOption3()
  {
    ...
  }

  ...
```

Alle Eigenschaften der JCheckBox wurden mit der Registerkarte Properties des WindowBuilder Editors eingestellt:

- Die Eigenschaft text legt die Beschriftung der JCheckBox fest
- Die Eigenschaft actionCommand legt eine Zeichenkette fest, die bei der Verarbeitung gelesen werden kann, damit die jeweilige JCheckBox einfacher identifiziert wird
- Die Eingeschaft selected legt fest, ob die JCheckBox bei erstmaliger Anzeige in der GUI als ausgewählt angezeigt werden soll oder nicht.

Das Einlesen der eingestellten Optionen erfolgt wie oben erkennbar in der statischen Methode bestimmeOptionsAuswahl(MainPanel mp) der Klasse bsp13110.Verarbeitung. Hier wird sichtbar, wie die JCheckBox-Komponenten in einem Array gruppiert werden können und wie das Einlesen der eingestellten Optionen erfolgt:

```
package bsp13110;

import javax.swing.JCheckBox;
import javax.swing.JOptionPane;

public class Verarbeitung
{
  public static void bestimmeOptionsAuswahl(MainPanel mp)
  {
// Variable für die Anzeige der Auswahl
    String chbAuswahl = "";
// Gruppierung der Komponenten in einem Array
    JCheckBox[] chbArr = new JCheckBox[3];
    chbArr[0] = mp.getChbOption1();
    chbArr[1] = mp.getChbOption2();
    chbArr[2] = mp.getChbOption3();
    for (int i=0;i<chbArr.length;i++)
    {
```

```
// Abfrage, ob JCheckBox angehakt wurde
      if (chbArr[i].isSelected())
      {
        String cmd = null;
        if (chbArr[i] != null)
// actionCommand-Text zur CheckBox holen
          cmd = chbArr[i].getActionCommand();
        if (cmd != null)
        {
          cmd = cmd.trim();
          if (cmd.length() > 0)
// Optionsnummer aus actionCommand-Text an Variable zur
// Anzeige in der GUI anhängen
            chbAuswahl += ","+cmd;
        }
      }
    }
// Komma am Ende von chbAuswahl entfernen
    if (chbAuswahl.length()>1)
      chbAuswahl = chbAuswahl.substring(1,chbAuswahl.length());
// Anzeige der Benutzereingabe
    if (chbAuswahl.length()==0)
    {
      mp.getTfAuswahl().setText("");
      JOptionPane.showMessageDialog(
          mp,
          "Es ist keine Option ausgewählt!",
          "Benutzerhinweis",
          JOptionPane.INFORMATION_MESSAGE
      );
    }
    else
    {
      mp.getTfAuswahl().setText(chbAuswahl);
    }
  }

  ...
}
```

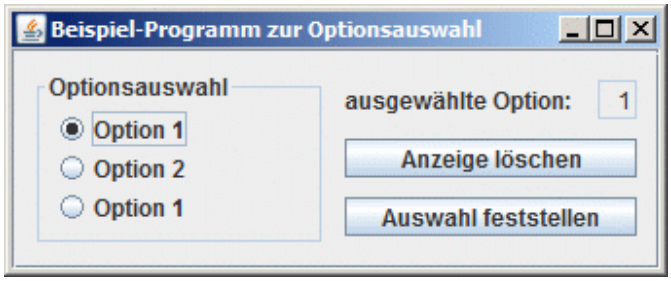
Das Anlegen einer Gruppe von JCheckBox-Komponenten kann insgesamt in folgendem Rezept zusammengefasst werden:

Rezept für das Anlegen von JCheckBox-Komponenten in einem Panel

- JPanel mit dem WindowBuilder Editor in die GUI einfügen.
- Für das Panel das Layout auf null setzen.
- JCheckBox-Komponenten im Bereich des Panels mit dem WindowBuilder Editor in die GUI einfügen.
- Für die Verarbeitung Code aus der Methode bestimmeOptionsAuswahl(..) aus bsp13110.Verarbeitung verwenden.

13.4.2.2 JRadioButton

Komponenten vom Typ JRadioButton werden grundsätzlich wie Komponenten vom Typ JCheckBox codiert. Da sie allerdings nur dann eingesetzt werden sollten, wenn *nur genau eine der angezeigten Optionen* ausgewählt werden sollen, müssen sie mit einer geeigneten Klasse so verbunden werden, dass bei Programmlauf auch nur eine Option ausgewählt werden kann, d.h. so dass die zuvor eingestellte Auswahl automatisch entfernt wird, wenn eine neue Auswahl vorgenommen wird. Entsprechend zeigt Beispielprogramm jpr13120 immer nur eine ausgewählte Auswahl an:



Der Quellcode zu den in einem Panel gruppierten JRadioButton-Komponenten besteht aus

- drei Komponenten vom Typ JRadioButton
- einem JPanel zur optischen Gruppierung der JRadioButton-Komponenten
- einem Objekt der Klasse ButtonGroup zur logischen Gruppierung der JRadioButton-Komponenten

Wir finden somit im Quellcode fünf Deklarationen von Objekten mit den zugehörigen get-Methoden:

```
package bsp13120;

import javax.swing.JPanel;
import javax.swing.ButtonGroup;
import javax.swing.JRadioButton;
...

public class MainPanel extends JPanel
{
  private JPanel panRB;
  private JRadioButton rbOption1;
  private JRadioButton rbOption2;
  private JRadioButton rbOption3;
  private ButtonGroup bgRadioButtonGruppe;

  ...

  private void initialize()
  {
    ...
    add(getPanRB());

// manuell hinzugefügt: ButtonGroup anlegen
    getBgRadioButtonGruppe();

    ...
  }

  protected JPanel getPanRB()
  {
    if (panRB == null)
    {
      panRB = new JPanel();
      panRB.setBorder(
        new TitledBorder(
            null,
            "Optionsauswahl",
            TitledBorder.LEADING,
            TitledBorder.TOP,
            null,
            null));
      panRB.setBounds(10, 10, 150, 90);
      panRB.setLayout(null);
      panRB.add(getRbOption1());
      panRB.add(getRbOption2());
      panRB.add(getRbOption3());
    }
    return panRB;
  }

  protected ButtonGroup getBgRadioButtonGruppe()
  {
    if (bgRadioButtonGruppe == null)
    {
      bgRadioButtonGruppe = new ButtonGroup();
// manuell hinzugefügt: RadioButton-Objekte zur ButtonGroup hinzufügen
      bgRadioButtonGruppe.add(getRbOption1());
      bgRadioButtonGruppe.add(getRbOption2());
      bgRadioButtonGruppe.add(getRbOption3());
    }
    return bgRadioButtonGruppe;
  }

  protected JRadioButton getRbOption1()
  {
    if (rbOption1 == null)
```

```
    {
      rbOption1 = new JRadioButton("Option 1");
      rbOption1.setSelected(true);
      rbOption1.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent arg0)
        {
          Verarbeitung.bestimmeOptionsAuswahl(getMainPanel());
        }
      });
      rbOption1.setActionCommand("1");
      rbOption1.setBounds(10, 20, 130, 20);
    }
    return rbOption1;
  }

  protected JRadioButton getRbOption2()
  {
    ...
  }

  protected JRadioButton getRbOption3()
  {
    ...
  }

  ...
```

Alle Eigenschaften der JRadioButton-Komponenten wurden mit der Registerkarte Properties des WindowBuilder Editors eingestellt:

- Die Eigenschaft text legt die Beschriftung des JRadioButtonfest
- Die Eigenschaft actionCommand legt eine Zeichenkette fest, die bei der Verarbeitung gelesen werden kann, damit der jeweilige JRadioButton einfacher identifiziert wird
- Die Eingeschaft selected legt fest, ob der JRadioButton bei erstmaliger Anzeige in der GUI als ausgewählt angezeigt werden soll oder nicht. Man achte darauf, diese Eigenschaft nur für einen einzigen JRadioButton einer ButtonGroup zu setzen.

In obigem Beispiel wurden die JRadioButton-Komponenten mit einer ButtonGroup logisch gruppiert. Wie man an dem Kommentar im obigen Quelltext sieht, wurde das ButtonGroup-Objekt mit der Palette des WindowBuilder Editors, Eintrag "Choose component" angelegt. Dabei wurde diese Komponenten außerhalb des JPanels angeordnet. Sie wurde unsichtbar angelegt, wie man auch an folgendem im Quelltext generierten Dokumentationskommentar erkennt:

```
/**
 * @wbp.nonvisual location=10,200
 */
```

Mit dem WindowBuilder Editor kann zwar auf diese Weise ein ButtonGroup-Objekt angelegt werden, allerdings wird kein Code für das Instanziieren des Objektes zur Laufzeit sowie für die Konfiguration des Objektes erzeugt. Die entsprechenden Anweisungen müssen manuell eingefügt werden:

1. Die Instanziierung des ButtonGroup-Objektes muss durch Einfügung des Aufrufs der get-Methode `getBgRadioButtonGruppe()` in die Methode initialize() angestoßen werden:

   ```
   getBgRadioButtonGruppe();
   ```

2. Das Gruppieren der RadionButton-Objekte in der ButtonGroup erfolgt durch manuelles Codieren von Aufrufen der Instanzmethode add() des ButtonGroup-Objektes:

   ```
   bgRadioButtonGruppe.add(getRbOption1());
   bgRadioButtonGruppe.add(getRbOption2());
   bgRadioButtonGruppe.add(getRbOption3());
   ```

Das Einlesen der eingestellten Optionen wie im Beispiel zur JCheckBox geschieht wie oben erkennbar in der statischen Methode bestimmeOptionsAuswahl(MainPanel mp) der Klasse bsp13120.Verarbeitung. Wieder werden die JRadioButton-Komponenten in einem Array gruppiert:

```
package bsp13120;

import javax.swing.JOptionPane;
import javax.swing.ButtonGroup;
import javax.swing.JRadioButton;
import java.util.Enumeration;

public class Verarbeitung
```

```
{
  private Verarbeitung() { }

  public static void loescheAnzeige(MainPanel mp)
  {
    mp.getTfAuswahl().setText("");
  }

  public static void bestimmeOptionsAuswahl(MainPanel mp)
  {
    JRadioButton rb;
    rb = getSelectedRadioButton(mp.getBgRadioButtonGruppe());
    char optAusw = ' ';
    String cmd = null;
    if (rb != null)
// 6. actionCommand-Text vom selektierten JRadioButton holen
      cmd = rb.getActionCommand();
    if (cmd != null)
    {
      cmd = cmd.trim();
      if (cmd.length() > 0)
        optAusw = cmd.charAt(0);
    }
    if (optAusw == ' ')
      JOptionPane.showMessageDialog(
          mp,
          "Fehler - es ist keine Option ausgewählt!",
          "Benutzerhinweis",
          JOptionPane.ERROR_MESSAGE
      );
    else
      mp.getTfAuswahl().setText(String.valueOf(optAusw));
  }

  private static JRadioButton getSelectedRadioButton(ButtonGroup bg)
  {
    JRadioButton rbSelected = null;
// 1. Referenzen der RadioButton-Objekte von der ButtonGroup holen
    Enumeration<?> e = bg.getElements();
    while (e.hasMoreElements() & rbSelected == null)
    {
// 2. Sukzessiv die einzelnen Referenzen aus der Enumeration holen
      Object o = e.nextElement();
      if (o instanceof JRadioButton)
      {
// 3. Cast nach Datentyp JRadioButton
        JRadioButton rb = (JRadioButton) o;
// 4. Referenzen der Daten-Modelle vergleichen. Ist die Referenz
//    der Modelle gleich, so sind auch die Radiobuttons identisch
        if (rb.getModel() == bg.getSelection())
        {
// 5. Referenz des gewählten JRadioButton speichern
          rbSelected = rb;
        }
      }
    }
    return rbSelected;
  }
}
```

Der Lesevorgang zur Feststellung der ausgewählten Option geschieht also in folgenden Schritten:

1. Vom ButtonGroup-Objekt wird mit der Methode getElements() ein Objekt der Klasse Enumeration geholt, das die Referenzen auf die einzelnen in der ButtonGroup zusammengefassten JRadioButtons enthält.
2. Mit der Methode nextElement() werden in einer Schleife die Referenzen auf die JRadioButton-Objekte geholt. Diese erhält man zunächst im Datentyp Object.
3. Für die JRadioButton-Objekte wird mit einem Cast der Datentyp nach JRadioButton geändert.
4. Wenn das Datenmodell des JRadioButton-Objektes gleich dem Objekt ist, das man von dem ButtonGroup-Objekt mit der Methode getSelection() erhält, so ist der ausgewählte JRadioButton gefunden.
5. Die Referenz dew gewählten JRadioButtons wird in einer Variablen vom Typ JRadioButton gespeichert.
6. Mit der Methode getActionCommand() der Klasse JRadioButton kann der actionCommand-Text vom ausgewählten JRadioButton geholt werden.

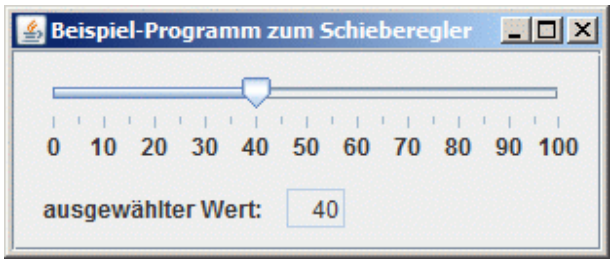
Das Anlegen einer Gruppe von JRadioButtons kann insgesamt in folgendem Rezept zusammengefasst werden:

Rezept für das Anlegen von JRadioButtons in einem Panel

- JPanel mit dem WindowBuilder Editor in die GUI einfügen.
- Für das Panel das Layout auf null setzen.
- JRadioButtons im Bereich des Panels mit dem WindowBuilder Editor in die GUI einfügen.
- Klasse ButtonGroup mit dem WindowBuilder Editor als Java Bean neben die GUI einfügen.
- In der Methode initialize() manuell den Aufruf der get-Methode für die Button Group codieren.
- Die RadioButtons mit manuell codierten add-Aufrufen zur ButtonGroup hinzufügen.
- Für die Verarbeitung Code aus der Methode bestimmeOptionsAuswahl(..) aus bsp13120.Verarbeitung verwenden. Ferner Methode getSelectedRadioButton(..) aus der gleichen Klasse verwenden.

13.4.2.3 JSlider

Für die Einstellung von Werten auf einer Skala mit fester Schrittweite lässt sich der Schieberegler JSlider sehr gut verwenden. Beispiel jpr13130 zeigt einen Schieberegler mit einer Skala von 0 bis 100, größeren Skaleneinteilungen alle 10 Einheiten und kleineren Skaleneinteilungen mittig zwischen den größeren Einteilungen:



Zu einem solchen Schieberegler gehört wieder eine Deklaration sowie eine get-Methode:

```
package bsp13130;

import javax.swing.JSlider;
...

public class MainPanel extends JPanel
{
  ...

  private JSlider slAuswahl;

  ...

  protected JSlider getSlAuswahl()
  {
      slAuswahl = new JSlider();
      slAuswahl.addChangeListener(new ChangeListener()
      {
        public void stateChanged(ChangeEvent arg0)
        {
          Verarbeitung.zeigeSliderWert(getMainPanel());
        }
      });
      slAuswahl.setMajorTickSpacing(10);
      slAuswahl.setValue(0);
      slAuswahl.setPaintTicks(true);
      slAuswahl.setPaintLabels(true);
      slAuswahl.setName("Prozentangabe");
      slAuswahl.setMinorTickSpacing(5);
      slAuswahl.setBounds(10, 10, 280, 50);
    }
    return slAuswahl;
  }
  ...
}
```

Wie man sieht ist die Codierung eines JSliders einfach. Die Komponente kann mit dem WindowBuilder Editor erzeugt werden und alle ihre

Eigenschaften können mit der Registerkarte Properties des WindowBuilder Editor gesetzt werden. Zusätzlich kann mit einem ChangeListener eine Ereignisbehandlung für ChangeEvent Ereignisse zugeordnet werden, die eine verarbeitende Methode aufruft. Folgende wichtigen Eigenschaften eines JSlider-Objektes sollten gesetzt werden:

- Eigenschaft paintLabels / Methode setPaintLabels(boolean) steuern, ob der JSlider eine Beschriftung mit Ziffern erhalten soll.
- Eigenschaft paintTicks / Methode setPaintTicks(boolean) steuern, ob die senkrechten Striche der Skaleneinteilung gezeichnet werden sollen.
- Eigenschaft majorTickSpacing / Methode setMajorTickSpacing(int) steuern, in welchem Abstand die größeren Skalen-Striche gezeichnet werden sollen.
- Eigenschaft minorTickSpacing / Methode setMinorTickSpacing(int) steuern, in welchem Abstand zusätzlich zu den größeren Skalen-Striche kleine Striche gezeichnet werden sollen.
- Eigenschaft value / Methode setValue(int) stellte den Startwert ein, mit dem der JSlider angezeigt werden soll.

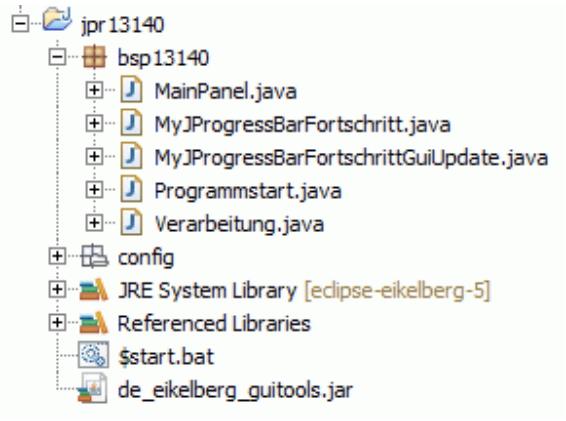
Im obigen Quellcode von Beispiel jpr13130 wird bei Zustandsänderungen des JSlider eine statische Methode zeigeSliderWert(MainPanel) der Klasse bsp13130.Verarbeitung aufgerufen. In dieser Methode erfolgt das Lesen des aktuell eingestellten JSlider-Wertes in einfacher Weise:

```
package bsp13130;

public class Verarbeitung
{
  private Verarbeitung() { }

  public static void zeigeSliderWert(MainPanel mp)
  {
    int wert = 0;
// Lesen des aktuell eingestellten JSlider-Wertes
    wert = mp.getSlAuswahl().getValue();
    mp.getTfAnzeige().setText(String.valueOf(wert));
  }
}
```

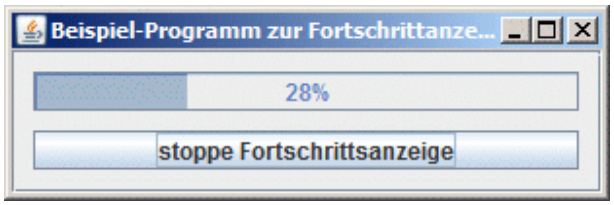
13.4.2.4 JProgressBar

Die Fortschrittsanzeige JProgressBar ist eine verhältnismäßig aufwändig zu codierende Komponente der GUI, da ihre Anzeige ständig aktualisiert wird, was die Programmierung eines Threads erfordert. Ein Thread ist ein relativ unabhängig vom eigentlichen Programm laufender Programmteil mit eigener Verarbeitung. Ein Thread darf in der Regel nicht direkt in die GUI schreiben, sondern muss für diesen Schreibvorgang einen weiteren Thread absetzen, der über die Ereignissteuerung (EventQueue) gelenkt wird, so dass der Schreibvorgang in die GUI mit der Ereignisbehandlung zur GUI synchronisiert wird. Entsprechend besteht ein Programm mit JProgressBar aus mindestens 5 Klassen:

- Programmstart - Klasse mit der main-Methode
- MainPanel - Klasse mit dem Haupt-Panel der GUI
- Verarbeitung - Klasse mit der Verarbeitung zur Ereignisbehandlung
- MyJProgressBarFortschritt - Klasse mit dem Thread zur Steuerung der Anzeige des JProgressBar
- MyJProgressBarFortschrittGuiUpdate - Klasse mit dem Thread, der die Anweisung zur Aktualisierung der GUI enthält.



Diese 5 Klassen ergeben insgesamt die Benutzeroberfläche von Beispiel jpr13140:



Diese Benutzeroberfläche mit dem JProgressBar ergibt sich zunächst aus folgendem Quellcode im Panel MainPanel:

```
package bsp13140;

import javax.swing.JProgressBar;
...

public class MainPanel extends JPanel
{

  ...
```

```
  private JProgressBar pbFortschrittAnzeige;

  ...

  protected JProgressBar getPbFortschrittAnzeige()
  {
    if (pbFortschrittAnzeige == null)
    {
      pbFortschrittAnzeige = new JProgressBar();
      pbFortschrittAnzeige.setStringPainted(true);
      pbFortschrittAnzeige.setFont(new Font("Dialog", Font.BOLD, 12));
      pbFortschrittAnzeige.setBounds(10, 10, 280, 20);
    }
    return pbFortschrittAnzeige;
  }
  ...
}
```

Die JProgressBar-Komponente sowie ihre Eigenschaften können wieder vollständig mit dem WindowBuilder Editor erzeugt werden:

- Eigenschaft font / Methode setFont(...) stellen den Font für die Beschriftung in dem JProgressBar ein.
- Eigenschaft stringPainted / Methode setStringPainted(boolean) stellen ein, ob die Beschriftung im JProgressBar erscheinen soll.

Die JProgressBar-Komponente erhält keine Ereignisverarbeitung und auch der Quellcode zur Veränderung der Anzeige wird nicht in der get-Methode zur Komponenten mit der Komponente verknüpft. Vielmehr erfolgt der Anstoß für die Veränderung der Anzeige der JProgressBar-Komponente in einer verarbeitenden Methode - hier der Methode schalteProgressBarAnzeige(MainPanel):

```
package bsp13140;

public class Verarbeitung
{
  ...

  private static MyJProgressBarFortschritt myJPBF = null;

  public static void schalteProgressBarAnzeige(MainPanel mp)
  {
    if (myJPBF == null)
    {
// Thread neu erzeugen und starten
      myJPBF = new MyJProgressBarFortschritt(mp.getPbFortschrittAnzeige());
      myJPBF.start();
      ...
    }
    else
    {
// Thread auf nicht laufend setzen so dass er endet und Referenz auf Thread löschen
      myJPBF.beenden();
      myJPBF = null;
      ...
    }
  }
}
```

Die Verarbeitung startet mit einem Null Pointer in der Variablen MyJProgressBarFortschritt. Dies entspricht der Tatsache, dass kein Thread existiert der die Anzeige des JProgressBar ändert. Die Methode schalteProgressBarAnzeige(MainPanel) schaltet dann zwischen laufender und nicht laufender Anzeige hin und her,

- indem entweder ein Objekt der Klasse MyJProgressBarFortschritt - also ein Thread zur Veränderung der Anzeige - instanziiert und gestartet wird oder
- indem der laufende Thread angehalten wird, wodurch dieser etwas zeitversetzt endet.

Die eigentliche Verarbeitung zur Veränderung der Anzeige der JProgressBar-Komponente geschieht in der run-Methode der Klasse MyJProgressBarFortschritt, denn diese Klasse erbt von der Klasse Thread, was zur Folge hat

- dass die Verarbeitung des Threads in einer Methode public void run() codiert werden muss,
- dass die Verarbeitung des Threads mit der geerbten Methode start() - die ihrerseits run() aufruft - gestartet wird und
- dass die Verarbeitung des Threads am besten gestoppt wird mit einer booleschen Variablen run, die bewirkt, dass der Thread nur dann weiterläuft, wenn diese Variable den Wert true hat:

```
package bsp13140;

import javax.swing.*;
```

```
import java.awt.EventQueue;

// Die Klasse MyJProgressBarFortschritt ist von der Klasse Thread abgeleitet
public class MyJProgressBarFortschritt extends Thread
{

  private JProgressBar pBar;

// sleepvalue gibt an, wieviele ms jeweils gewartet wird, bevor der Thread fortsetzt.
// Setzt man sleepvalue gleich 600 und hat man eine Fortschrittsanzeige mit
// 100 Schritten, so läuft die Anzeige 100*600 msec = 60 sec = 1 min.
  final int sleepvalue = 600;

// Variable zur Steuerung, ob der Thread weiter laufen soll
  private boolean run = true;

// Dieser Konstruktor übergibt das JProgressBar-Objekt an die Klasse
  public MyJProgressBarFortschritt(JProgressBar pBar)
  {
    this.pBar = pBar;
    this.run = true;
// Thread enthält Verarbeitung zum Programm und muss daher als
// user-Thread - nicht als Daemon-Thread - konfiguriert werden
    this.setDaemon(false);
  }

  public void beenden()
  {
    this.run = false;
  }
// Das Überschreiben der geerbten Methode run() ist erforderlich. Nach Instanziieren
// eines Objektes, das von der Klasse Thread abgeleitet ist, und nach Start der
// Methode start() wird die Methode run() gestartet. Im Programmcode der Methode
// run() wird festgelegt, was ein Objekt während seiner Existenz ("Laufzeit")
// durchführen soll.
  public void run()
  {
// Der JProgressBar wurde in MyFrame mit Werten zwischen 0 und 100 mit
// Inkrement 1 eingerichtet, also werden 100 Schritte erzeugt.
    for (int i = 0; i <= 100 & run; i++)
    {
// Thread sleepvalue viele Millisekunden warten lassen
      try
      {
        Thread.sleep(sleepvalue);
      }
      catch (InterruptedException e) {}

// Aktualisierung der GUI über einen Thread, dessen run-Methode die Anweisungen
// zur Aktualisierung der GUI enthält:
// 1. Thread zur Aktualisierung der GUI deklarieren
      MyJProgressBarFortschrittGuiUpdate mJPB;
// 2. Das zu aktualisierende Feld und den neuen Wert für den Fortschritt des JProgressBar
//    über den Konstruktor an den Thread zur Aktualisierung der GUI weitergeben
      mJPB = new MyJProgressBarFortschrittGuiUpdate(this.pBar,i);
// 3. Ausführung der Implementierung der Methode run() des Threads als Ereignis
//    in die EventQueue (also zur Ausführung im Thread zur Ereignisverarbeitung)
//    einstellen. Die Ausführung wird ggf. leicht verzögert ausgeführt.
      EventQueue.invokeLater(mJPB);
    }
  }
}
```

Der obige Quellcode ist ausführlich dokumentiert. Er zeigt,

- dass die Klasse MyJProgressBarFortschritt von der Klasse Thread abgeleitet ist,
- dass der Thread in der Methode run() eine Verarbeitung enthält,
- dass die Verarbeitung des Threads maximal 100 mal - und sofern die Variable run den Wert true hat - durchgeführt wird,
- dass die Verarbeitung des Threads beinhaltet, dass zunächst 600 msec gewartet wird und anschließend eine Aktualisierung der GUI erfolgt, und
- dass die Aktualisierung der GUI mit einem weiteren Thread MyJProgressBarFortschrittGuiUpdate erfolgt, der jedoch nicht mit der Methode start() gestartet wird sondern mit der Methode invokeLater(Thread) zur späteren Ausführung in die Warteschlange der

Ereignisbehandlung (EventQueue) eingestellt wird.

Der eigentliche Veränderungsvorgang für die Anzeige der JProgressBar-Komponente befindet sich also im Thread MyJProgressBarFortschritt:

```
package bsp13140;

import javax.swing.*;

public class MyJProgressBarFortschrittGuiUpdate extends Thread
{

  private JProgressBar pBar;
  private int i;

// Dieser Konstruktor übergibt das JProgressBar-Objekt an die Klasse
  public MyJProgressBarFortschrittGuiUpdate(JProgressBar pBar, int i)
  {
    this.pBar = pBar;
    this.i = i;
//  Thread enthält Verarbeitung zum Programm und muss daher als
//  user-Thread - nicht als Daemon-Thread - konfiguriert werden
     this.setDaemon(false);
  }

// Implementierung der Methode run()
  public void run()
  {
// Hier erfolgt der Zugriff auf die GUI - Anzeige des JProgressbar
// aktualisieren
     this.pBar.setValue(this.i);
  }
}
```

Der Aufbau der Klasse MyJProgressBarFortschrittGuiUpdate ähnelt dem Aufbau der Klasse MyJProgressBarFortschritt, beide sind von der Klasse Thread abgeleitet. Die Klasse MyJProgressBarFortschrittGuiUpdate führt jedoch eine einmalige Verarbeitung durch, nach der der Thread endet. Entsprechend ist keine Variable und auch keine Methode für das Beenden des Threads erforderlich. In der Methode run() steht jetzt die Anweisung zur Aktualisierung der GUI, nämlich der Aufruf der Instanzmethode setValue(int) der Klasse JProgressBar.

13.4.3 Programmierung von Modellen für Daten

13.4.3.1 Modelle für Daten gemäß dem Java Delegate Modell

Im Allgemeinen sieht das sogenannte Java Delegate Modell (vgl. 3.1.3) vor, dass in der GUI anzuzeigende Daten in separaten Klassen getrennt von der GUI gespeichert werden sollen. Diese Klassen mit den Daten nennt man Modelle. Sie stellen die Daten bereit, die in der GUI (mit der View für die Anzeige und dem Controller für die Aufnahme von Ereignissen) angezeigt werden.

Insgesamt verteilt sich damit die Funktionalität einer Komponente der GUI gemäß einem Grundprinzip der Informatik also zunächst auf drei Bestandteile:

Model - View - Controller

Das **Model-View-Controller**-Muster stellt ursprünglich das Prinzip für den Aufbau der grafischen Oberfläche (**GUI** - graphical user interface) der Programmiersprache Smalltalk dar. Es kann wie folgt am Beispiel einer Tabelle erläutert werden:

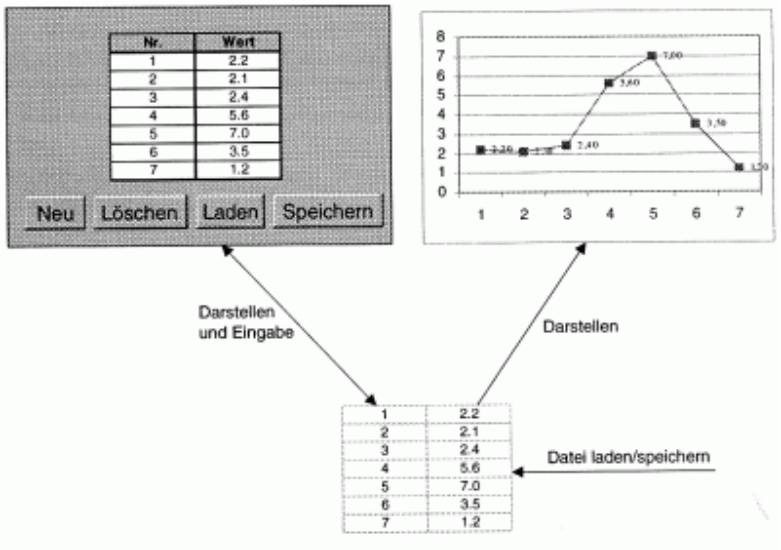


Abb. aus Goll, Weiß, Müller. Java als erste Programmiersprache S. 611

- Die Tabelle mit den Daten stellt das Model dar.
- Die Darstellung der Daten in der Grafik stellt eine Ansicht (View) der Daten des Models dar.
- Die Darstellung der Daten in einer grafischen Benutzeroberfläche stellt ebenfalls eine (andere) Ansicht (**View**) dar. Die Buttons der grafischen Benutzeroberfläche bilden einen **Controller** für das Model.

Die Darstellung und Bearbeitung von Daten mit dem Model-View-Controller-Muster ermöglicht, dass Model, View und Controller separat in der jeweiligen Programmiersprache realisiert werden können. Hierdurch wird es bei der Programmierung prinzipiell möglich, Model, View oder Controller jeweils durch andere Versionen auszutauschen - zum Beispiel zur Veränderung des Designs einer grafischen Benutzeroberfläche.

Model, View und Controller haben folgende Aufgaben:

- Das **Model** enthält die Daten sowie Methoden zum Lesen, Speichern und Bearbeiten der Daten. Ferner muss das Model den View und den Controller stets benachrichtigen, wenn Änderungen in den Daten des Models vorgenommen wurden.
- Die **View** ist verantwortlich für die Darstellung der Daten auf der Benutzeroberfläche des Programms. Hierzu muss der View beim Model registriert werden.
- Der **Controller** ist verantwortlich für die Aufnahme von Benutzereingaben und die Weitergabe dieser Eingaben an Model oder View. Auch der Controller muss sich beim Model registrieren. Ferner muss vom Controller sichergestellt werden, dass nach Benutzereingaben die Anzeige der View stets wieder die aktuellen Daten des Models enthält.

Bei der Realisierung des Model-View-Controller-Musters in der Programmiersprache Java wurden die View und der Controller zu dem sogenannten **Delegate** zusammengefasst. Man erhält das **Java Delegate-Modell**:

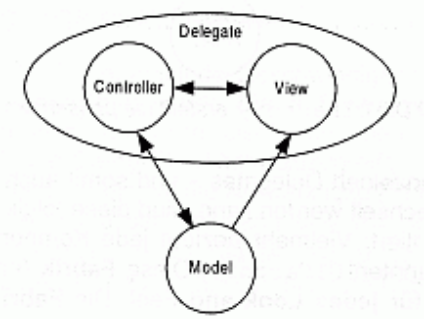


Abb. aus Goll, Weiß, Müller. Java als erste Programmiersprache S. 611

Der **Delegate** ist somit für folgende Aufgaben verantwortlich:

- die Darstellung der Daten auf der Benutzeroberfläche
- die Annahme der Benutzereingaben
- die Weitergabe von Benutzereingaben an das Model
- die Auffrischung der Anzeige nach Änderungen der Daten im Model

Für die in Kapitel 14.3.2 vorgestellten Komponenten in der GUI programmiert man - da man sie meistens nicht benötigt - in der Regel keine Modelle, obwohl dies auch möglich ist. Für die in Kapitel 14.3.4 vorzustellenden GUI-Elemente für Auswahlvorgänge ist es dagegen sehr oft zweckmäßig oder aber auch sogar notwendig, Datenmodelle zu codieren. Das grundsätzliche Vorgehen für die Codierung von Datenmodellen wird daher im aktuellen Kapitel anhand einer tabellarischen Anzeige von Daten mit der Swing-Komponente JTable mit zugehörigem Tabellenmodell erklärt.

13.4.3.2 Codierung einer tabellarischen Anzeige mit Delegate JTable und Tabellen-Modell AbstractTableModel

Ausgangspunkt einer tabellarischen Anzeige von Daten in der GUI ist die Swing-Komponente JTable. Sie ermöglicht es, Datenbestände tabellarisch angeordnet in der GUI anzuzeigen, die Anzeige zu sortieren und zu bearbeiten. Gemäß dem Java Delegate Modell sollte dabei die Anzeige des Datenbestandes (mit der Komponente JTable als Delegate) getrennt von der Speicherung des Datenbestandes (in einem Tabellen-Modell) codiert werden. Entsprechend sollte eine tabellarische Anzeige in der GUI mit Hilfe folgender Klassen codiert werden:

- `javax.swing.JTable` - Klasse mit der Komponente zur Anzeige des Datenbestandes in der GUI (Delegate)
- `javax.swing.table.AbstractTableModel` - abstrakte Klasse mit Methoden, die eine Klasse bereitstellen muss, in der der Datenbestand gespeichert wird. Zur Speicherung des Datenbestandes muss eine Klasse codiert werden - das sogenannte **Tabellenmodell** - die von der klasse AbstractTableModel abgeleitet ist und die die zu speichernden Daten aufnimmt (Model).

Das Beispiel jpr13180 zur tabellarischen Anzeige mit Tabellenmodell - es ist entsprechend den Ausführungen in Kapitel 13.1 - 13.2 sowie 13.4.1 aufgebaut - besteht entsprechend aus folgenden Klassen:

- `bsp13180.Programmstart` - Klasse mit der main-Methode
- `bsp13180.MainPanel` - Klasse mit dem Panel, also mit der gesamten GUI
- `bsp13180.MyTableModel` - Klasse mit dem Tabellenmodell (abgeleitet von javax.swing.table.AbstractTableModel)
- `bsp13180.Verarbeitung` - Klasse mit der Verarbeitung zum Panel MainPanel.

Die aus diesen Klasse aufgebaute Anwendung ergibt folgende Anzeige an der Benutzeroberfläche:

Sichtbar ist die JTable-Komponente, bestehend aus mehreren Spalten und Zeilen. Sie ist in eine JScrollPane eingebettet, die einen horizontalen und einen vertikalen Scrollbalken bereitstellt.

13.4.3.3 Generierung einer Komponente vom Typ JTable in einer JScrollPane mit dem WindowBuilder Editor

Eine Komponente vom Typ JTable sollte in der Regel in einen Bereich eingebettet werden, der Scrollbalken bereitstellt und so ermöglicht, dass die einzubettende Komponente mehr Fläche benötigt als von dem umgebenden Panel bereitgestellt wird. Daher gehe man zur Codierung einer JTable mit dem WindowBuilder Editor wie folgt vor:

1. Zunächst füge man in der gewohnten Weise mit dem WindowBuilder Editor eine Komponente vom Typ JScrollPane in das Panel mit der GUI ein. Man findet die Komponente JScrollPane in der Palette des WindowBuilder Editors im Abschnitt "Containers". Es wird emfohlen, der JScrollPane einen Namen zu geben, der mit dem Text **sp** beginnt.
2. In die JScrollPane füge man jetzt mit dem WindowBuilder Editor eine JTable ein. Diese Komponente findet man in der Palette im Abschnitt "Components". Beim Einfügevorgang ist es nicht erforderlich, den genauen Ort für die Komponente zu markieren. Vielmehr reicht es aus, einmal mit der linken Maustaste in in den Bereich der JScrollPane zu klicken, der beim Einfügen als Viewport markiert wird. Es wird emfohlen, der JTable einen Namen zu geben, der mit dem Text **tab** beginnt.
3. Zur Vereinfachung ist es in der Regel hilfreich, für die JTable das gleichzeitige Selektieren mehrerer Zeilen auszuschalten. Dies geschieht durch Einstellen der Eigenschaft der JTable mit dem Namen selectionMode auf den Wert SINGLE_SELECTION.

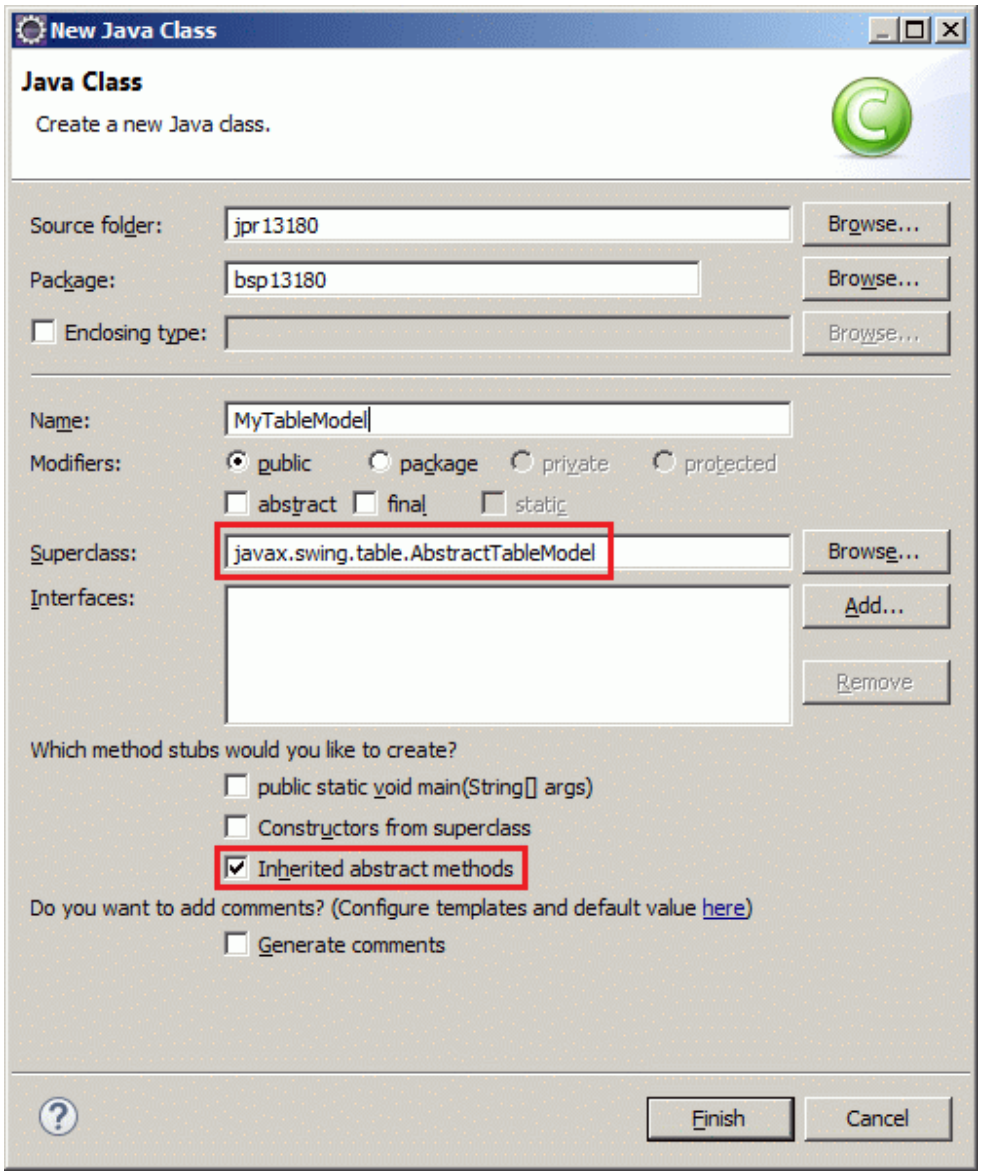
Nach Ausführung dieser Schritte erhält man in dem Panel mit der GUI (MainPanel) Quellcode wie in folgendem Ausschnitt aus der Klasse bsp13180.MainPanel:

```
package bsp13180;

import ...

public class MainPanel extends JPanel
{
  ...

  private JScrollPane spTabellenScrollPane;
  private JTable tabDaten;

  ...

  protected JScrollPane getSpTabellenScrollPane()
  {
    if (spTabellenScrollPane == null)
    {
      spTabellenScrollPane = new JScrollPane();
      spTabellenScrollPane.setBounds(5, 40, 215, 100);
      spTabellenScrollPane.setViewportView(getTabDaten());
    }
    return spTabellenScrollPane;
  }

  protected JTable getTabDaten()
  {
    if (tabDaten == null)
    {
      tabDaten = new JTable();
      ...
      tabDaten.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    }
    return tabDaten;
  }
  ...
}
```

13.4.3.4 Codierung des Tabellenmodells

Für die Codierung des Tabellenmodells zur Speicherung der Daten gehe man wie folgt vor:

Das Tabellenmodell muss von der Klasse **javax.swing.table.AbstractTableModel** abgeleitet sein. So wird sichergestellt, dass das Tabellenmodell alle erforderlichen Methoden zur Speicherung und Verwaltung der Daten enthält. Legen Sie über das Kontextmenü des Paketes, in das das Tabellenmodell eingefügt werden soll, eine neue Klasse an und geben Sie dabei an, dass die Klasse von der Klasse javax.swing.table.AbstractTableModel abgeleitet werden soll:



Es entsteht eine Klasse, in der bereits 3 Methoden codiert sind, wenn Sie - wie in obigem Bildschirmausdruck - die Option "Inherited abstract methods" ausgewählt haben:

```
package bsp13180;

import javax.swing.table.AbstractTableModel;

public class MyTableModel extends AbstractTableModel
{

  public int getColumnCount()
  {
    // TODO Auto-generated method stub
    return 0;
  }

  public int getRowCount()
  {
    // TODO Auto-generated method stub
    return 0;
  }

  public Object getValueAt(int arg0, int arg1)
  {
    // TODO Auto-generated method stub
    return null;
```

```
  }
}
```

Für diese Klasse kann man sich von Eclipse wie folgt anzeigen lassen, welche Methoden zusätzlich zu den obigen zwingend notwendigen Implementierungen von Methoden überschrieben werden können: Klicken Sie mit der rechten Maustaste im Bereich des Quellcode und wählen Sie in dem sich daraufhin öffnenden Kontextmenü die Auswahl Source / Override/Implement Methods... . Für das Beispiel jpr13180 wurde die Klasse bsp13180.MyTableModel mit dem Tabellenmodell entsprechend wie folgt ergänzt, um eine Speicherung und Verwaltung der Tabellendaten sowie die Aktualisiserung der GUI bei Änderung des Datenbestandes zu erreichen:

```
package bsp13180;

import javax.swing.table.AbstractTableModel;

public class MyTableModel extends AbstractTableModel
{
  ...

  private String[] spaltennamen =
     {"Spalte 1","Spalte 2","Spalte 3","Spalte 4","Spalte 5"};
  private String[][] eintraege = new String[10][5];

  public Class<?> getColumnClass(int j)
  {
    return String.class;
  }

  public int getColumnCount()
  {
    return this.spaltennamen.length;
  }

  public String getColumnName(int j)
  {
    return this.spaltennamen[j];
  }

  public int getRowCount()
  {
    return this.eintraege.length;
  }

  public Object getValueAt(int i, int j)
  {
    return this.eintraege[i][j];
  }

  public boolean isCellEditable(int i, int j)
  {
    return true;
  }

  public void setValueAt(Object obj, int i, int j)
  {
    this.eintraege[i][j] = obj.toString();
// threadsichere Methode zum Schreiben von Daten in die Tabelle:
    fireTableCellUpdated(i, j);
  }
}
```

Man beachte:

- Im String-Array `spaltennamen` werden die Namen der Spalten der Tabelle gespeichert. Die Methode
  `public String getColumnName(int j)`
  gibt den Namen der Spalte mit dem Spaltenindex j zurück.
- Im zweidimensionalen String-Array `eintraege` werden die Einträge der Tabelle gespeichert. Die Methode
  `public Object getValueAt(int i, int j)`
  gibt den Eintrag des String-Arrays `eintraege` mit dem Index (i,j) zurück.
- Umgekehrt nimmt die Methode
  `public void setValueAt(Object obj, int i, int j)`
  einen Eintrag zur Speicherung in das Tabellenmodell an und speichert diesen im String-Array `eintraege` im Feld mit dem Index (i,j). Zusätzlich informiert die Methode anschließend die GUI, dass die Anzeige der Daten im Delegate aktualisiert werden muss, denn das Modell ist stets verantwortlich dafür, die GUI über Datenänderungen zu informieren.

- Die Methode

  `getColumnClass(int j)`

  gibt den Datentyp der j-ten Spalte zurück. Alle Daten im Tabellenmodell sind vom Datentyp String, also gibt diese Methode im vorliegenden Beispiel den Wert String.class zurück. Der von dieser Methode zurückgegebene Datentyp beeinflusst die Anzeige der Daten in der JTable, weil bei dieser Anzeige Daten je nach Datentyp unterschiedlich formatiert angezeigt werden.
  Hinweis: Mit sogenannten CellRenderern kann diese Anzeige verändert werden.
- Die Methode

  `getColumnCount()`

  muss die Anzahl der Spalten der Tabelle zurückgeben.
- Die Methode

  `getRowCount()`

  muss die Anzahl der Zeilen der Tabelle zurückgeben.
- Mit der Methode

  `isCellEditable(int i, int j)`

  wird gesteuert, ob die Zelle der tabellarischen Anzeige der Größe im Tabellenmodell mit dem Index (i,j) editierbar angezeigt wird. Wird eine Editierbarkeit eingestellt, so ist es notwendig, Code für das Einlesen von Daten bereitzustellen.
  Hinweis: Mit sogenannten CellEditoren kann der Eingabevorgang komfortabel gestaltet werden.

Das Tabellenmodell MyTableModel kann jetzt wie folgt mit der JTable verbunden werden:

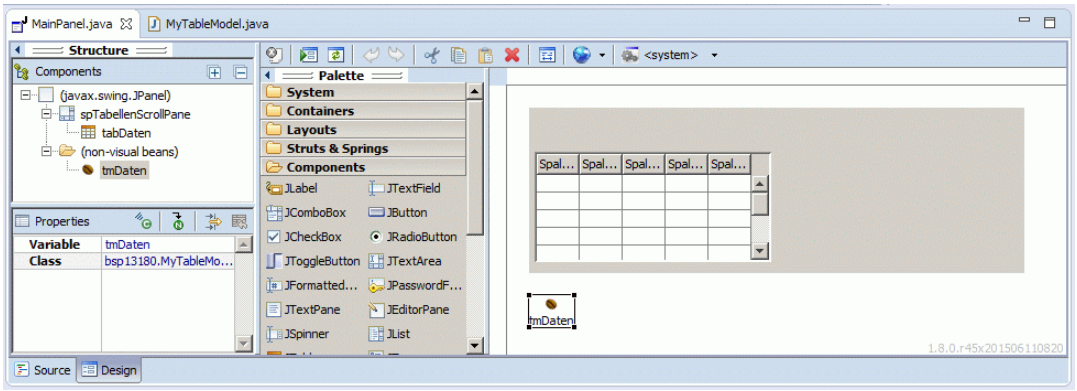
- Fügen Sie das Tabellenmodell über die Palette des WindowBuilder Editors, Abschnitt System, Link "Choose component" als unsichtbare Komponente in die GUI ein. Wählen Sie für diese Komponente einen Namen, der mit tm beginnt. Anders als bei der ButtonGroup (Beispiel 13120) wird das Tabellenmodell im WindowBuilder Editor (als JavaBean) im Designer angezeigt (s.u.).
- Ändern Sie die Sichtbarkeit der Methode getTmDaten() ab auf protected. Es entsteht folgender Quelltext:

```
/**
 * @wbp.nonvisual location=20,190
 */
protected MyTableModel getTmDaten()
{
  if (tmDaten == null)
  {
    tmDaten = new MyTableModel();
  }
  return tmDaten;
}
```

- Fügen Sie manuell in get-Methode für die JTable die im folgenden Quellcode markierte Zeile ein:

```
protected JTable getTabDaten()
{
  if (tabDaten == null)
  {
    tabDaten = new JTable();
    ...
    tabDaten.setModel(getTmDaten());
  }
  return tabDaten;
}
```

Das Tabellenmodell wird jetzt im WindowBuilder Editor zusätzlich als Java Bean (mit der Kaffeebohne als Symbol) angezeigt:



13.4.3.5 Zugriff auf die Tabelle in der GUI sowie auf die Tabellendaten aus der Verarbeitung

Aus der Verarbeitung wird wie folgt auf die Anzeige der Tabelle in der GUI sowie auf die Tabellendaten im Tabellenmodell zugegriffen:

Der **lesende Zugriff auf die Anzeige der Tabelle in der GUI** z. Bsp.

- um markiere Einträge festzustellen [Ereignisbehandlung zum Ereignis **mouseReleased(MouseEvent) zur JTable**] oder
- um neu in der GUI eingegebene Daten einzulesen [Ereignisbehandlung zum Ereignis **keyReleased(KeyEvent) zur JTable**]

erfolgt über die Komponente JTable.

Im Beispiel jpr13180 sind entsprechend

- beim Feststellen von in der Anzeige markierten Einträgen [mouseReleased(MouseEvent) ruft **zeigeAuswahlAn(MainPanel)** auf] sowie
- beim Lesen von neu in der GUI eingegebenen Daten aus der GUI [keyReleased(KeyEvent) ruft **speichere(MainPanel)** auf]

folgende Anweisungen codiert worden:

```
package bsp13180;

import ...

public class Verarbeitung
{
  ...
  public static void zeigeAuswahlAn(MainPanel mp)
  {
// Ausgewählte Option feststellen
    int i = mp.getTabDaten().getSelectedRow();
    int j = mp.getTabDaten().getSelectedColumn();
    String auswahl = "";
    if (i!=-1 & j!=-1) // ansonsten ist nicht ausgewählt
    {
      Object o = mp.getTabDaten().getValueAt(i,j);
      if (o != null)
        auswahl = o.toString();
//    GUI aktualisieren
      if (auswahl.equals(""))
        mp.getTfAuswahlAnzeige().setText(".. nichts ausgewählt ..");
      else
        mp.getTfAuswahlAnzeige().setText("Ausgewählt: "+auswahl);
    }
  }

  public static void speichere(MainPanel mp)
  {
// Ausgewählte Option feststellen
    int i = mp.getTabDaten().getSelectedRow();
    int j = mp.getTabDaten().getSelectedColumn();
    String auswahl = "";
    if (i!=-1 & j!=-1) // ansonsten ist nicht ausgewählt
    {
      Object o = mp.getTabDaten().getValueAt(i,j);
      if (o != null)
        auswahl = o.toString();
//    Daten im Modell speichern
      mp.getTmDaten().setValueAt(auswahl, i, j);
    }
  }
  ...
}
```

In obigem Listing der Methode speichere wird deutlich, dass beim Schreiben der Tabellendaten auf das Tabellenmodell zugegriffen wird. Dieses benachrichtigt dann die GUI und sorgt für die Aktualisierung der GUI:

Der **lesende und schreibende Zugriff auf die Tabellendaten** z. Bsp.

- beim Laden der Daten in das Tabellenmodell bzw.
- beim Speichern von neu in der GUI eingegebenen Daten in das Modell

erfolgt über das Tabellenmodell. Das Tabellenmodell muss dann für die Aktualisierung der GUI sorgen.

Im Beispiel jpr13180 wird entsprechend

- beim Speichern neu eingegebener Daten,

- beim Laden der Dateien von der Festplatte in das Tabellenmodell sowie
- beim Schreiben der Daten der Tabelle auf die Festplatte

auf das Tabellenmodell zugegriffen:

```
package bsp13180;

import ...

public class Verarbeitung
{
  ...
  public static void speichere(MainPanel mp)
  {
// Ausgewählte Option feststellen
    int i = mp.getTabDaten().getSelectedRow();
    int j = mp.getTabDaten().getSelectedColumn();
    String auswahl = "";
    if (i!=-1 & j!=-1) // ansonsten ist nicht ausgewählt
    {
      Object o = mp.getTabDaten().getValueAt(i,j);
      if (o != null)
        auswahl = o.toString();
//    Daten im Modell speichern
      mp.getTmDaten().setValueAt(auswahl, i, j);
    }
  }

  public static void ladeDaten(MainPanel mp)
  {
// Datei öffnen und lesen
    String einDateiName = "daten/tabelle.txt";
    ...
    dateiEin = new FileInputStream(einDateiName);
    ...
    InputStreamReader dateiLeser = null;
    BufferedReader dateiLesePuffer = null;
    ...
    dateiLeser = new InputStreamReader(dateiEin);
    dateiLesePuffer = new BufferedReader(dateiLeser);
    ...
    for (int i=0;i<mp.getTmDaten().getRowCount();i++)
    {
      String zeile = "";
      zeile = dateiLesePuffer.readLine();
      StringTokenizer st = new StringTokenizer(zeile,",");
      for (int j=0;j<mp.getTmDaten().getColumnCount();j++)
      {
        String eintrag = st.nextToken().trim();
        mp.getTmDaten().setValueAt(eintrag, i, j);
      }
    }
    ...
  }

  public static void speichereDaten(MainPanel mp)
  {
//  Datei öffnen und schreiben
    String ausDateiName = "daten/tabelle.txt";
    ...
    FileOutputStream dateiAus = null;
    ...
    dateiAus = new FileOutputStream(ausDateiName);
    ...
    OutputStreamWriter dateiSchreiber = null;
    BufferedWriter dateiSchreibPuffer = null;
    ...
    dateiSchreiber = new OutputStreamWriter(dateiAus);
    dateiSchreibPuffer = new BufferedWriter(dateiSchreiber);
    ...
    for (int i=0;i<mp.getTmDaten().getRowCount();i++)
    {
```

```
      for (int j=0;j<mp.getTmDaten().getColumnCount();j++)
      {
        Object o = mp.getTmDaten().getValueAt(i, j);
        String aus = " ";
        if (o!=null)
        {
          aus = o.toString();
          if (aus.equals(""))
             aus = " ";
        }
        dateiSchreibPuffer.write(aus);
        if (j<mp.getTmDaten().getColumnCount()-1)
             dateiSchreibPuffer.write(",");
      }
      dateiSchreibPuffer.newLine();
    }
  ...
  }
}
```

13.4.4 Elemente für Auswahlvorgänge

In diesem Kapitel wird zu den GUI-Elementen für Auswahlvorgänge der Code für

- die GUI (Delegate) sowie
- das Daten-Modell (Model)

zusammengestellt. Als Komponenten für Auswahlvorgänge stellt das Swing-Paket folgende Klassen bereit: JTable, JComboBox, JList und JTree.

13.4.4.1 JTable

Das Beispiel zur JTable mit Tabellenmodell AbstractTableModel ist ausführlich anhand von Beispiel jpr13180 in Kapitel 13.4.3 erklärt worden:

Für eine derartige tabellarische Anzeige benötigt man

- als Delegate eine JTable-Komponente, die zweckmäßigerweise in eine JScrollPane eingebettet sein sollte
- als Modell eine Klasse, die von der Klasse AbstractTableModel abgeleitet ist.

13.4.4.1.1 Codierung des Tabellenmodells

Das Tabellenmodell erhält man, indem man eine von der Klasse AbstractTableModel abgeleitete Klasse codiert. Eine solche Klasse erhält man mit Eclipse wie folgt:

- Markieren des Paketes, in das die Klasse eingefügt werden soll.
- Auswahl von New / Class im Kontextmenü des Paketes.
- Anlegen einer neuen Klasse im Fenster "New Java Class". Dabei Angeben der Klasse javax.swing.table.AbstractTableModel als Superclass.

Eine von AbstractTableModel abgeleitete Klasse muss folgende Methoden implementieren:

- Die Methode
  `getColumnCount()`
  muss die Anzahl der Spalten der Tabelle zurückgeben.
- Die Methode
  `getRowCount()`
  muss die Anzahl der Zeilen der Tabelle zurückgeben.
- Die Methode
  `public Object getValueAt(int i, int j)`
  muss den Eintrag im Tabellenmodell zurückgeben, der im Modell mit dem Index (i,j) gespeichert ist.

Ferner ist es oft zweckmäßig, über die oben genannten mindestens geforderten Methoden hinaus im Tabellenmodell folgende Methoden zu implementieren:

- Die Methode
  `public String getColumnName(int j)`
  sollte den Namen der Spalte mit dem Spaltenindex j zurückgeben.
- Die Methode
  `getColumnClass(int j)`
  sollte den Datentyp der j-ten Spalte zurückgeben. Der von dieser Methode zurückgegebene Datentyp beeinflusst die Anzeige der Daten in der JTable, weil bei dieser Anzeige Daten je nach Datentyp unterschiedlich formatiert angezeigt werden.
  Hinweis: Mit sogenannten CellRenderern kann diese Anzeige verändert werden.
- Mit der Methode

```
isCellEditable(int i, int j)
```

wird gesteuert, ob die Zelle der tabellarischen Anzeige der Größe im Tabellenmodell mit dem Index (i,j) editierbar angezeigt wird. Wird eine Editierbarkeit eingestellt, so ist es notwendig, Code für das Einlesen von Daten bereitzustellen.
Hinweis: Mit sogenannten CellEditoren kann der Eingabevorgang komfortabel gestaltet werden.

- Wenn eine Zelle editierbar konfiguriert wurde, so muss mit der Methode

```
public void setValueAt(Object obj, int i, int j)
```

der Eintrag zum Feld im Tabellenmodell mit dem Index (i,j) zur Speicherung in das Tabellenmodell angenommen werden. Zusätzlich muss die Methode anschließend die GUI informieren, dass die Anzeige der Daten im Delegate aktualisiert werden muss, denn das Modell ist stets verantwortlich dafür, die GUI über Datenänderungen zu informieren.

13.4.4.1.2 Codierung der tabellarischen Anzeige in der GUI

Die JTable, eingebettet in eine JScrollPane, erhält man mit dem WindowBuilder Editor wie folgt:

1. Zunächst füge man in der gewohnten Weise mit dem WindowBuilder Editor eine Komponente vom Typ JScrollPane in das Panel mit der GUI ein. Man findet die Komponente JScrollPane in der Palette des WindowBuilder Editors im Abschnitt "Containers". Es wird emfohlen, der JScrollPane einen Namen zu geben, der mit dem Text **sp** beginnt.
2. In die JScrollPane füge man jetzt mit dem WindowBuilder Editor eine JTable ein. Diese Komponente findet man in der Palette im Abschnitt "Components". Beim Einfügevorgang ist es nicht erforderlich, den genauen Ort für die Komponente zu markieren. Vielmehr reicht es aus, einmal mit der linken Maustaste in den Bereich der JScrollPane zu klicken, der beim Einfügen als Viewport markiert wird. Es wird emfohlen, der JTable einen Namen zu geben, der mit dem Text **tab** beginnt.
3. Zur Vereinfachung ist es in der Regel hilfreich, für die JTable das gleichzeitige Selektieren mehrerer Zeilen auszuschalten. Dies geschieht durch Einstellen der Eigenschaft der JTable mit dem Namen selectionMode auf den Wert SINGLE_SELECTION.
4. Konfigurieren Sie für die JTable für die Eigenschaft autoResizeMode den Wert AUTO_RESIZE_OFF.
5. Fügen Sie das Tabellenmodell über die Palette des WindowBuilder Editors, Abschnitt System, Link "Choose component" als unsichtbare Komponente in die GUI ein. Wählen Sie für diese Komponente einen Namen, der mit tm beginnt. Das Tabellenmodell wird daraufhin im WindowBuilder Editor im Designer als JavaBean angezeigt.
6. Ändern Sie die Sichtbarkeit der get-Methode für das Tabellenmodell ab auf protected.
7. Fügen Sie manuell in die get-Methode für die JTable die im folgenden Quellcode markierte Zeile ein:

```
/**
 * @wbp.nonvisual location=20,190
 */
protected MyTableModel getTmDaten()
{
  ...
}

protected JTable getTabDaten()
{
  if (tabDaten == null)
  {
    tabDaten = new JTable();
    ...
    tabDaten.setModel(getTmDaten());
  }
  return tabDaten;
}
```

Aufgrund der oben beschriebenen Aktionen sollte insgesamt folgender Quellcode entstanden sein:

```
private JScrollPane spTabellenScrollPane;
private JTable tabDaten;

...

protected JScrollPane getSpTabellenScrollPane()
{
  if (spTabellenScrollPane == null)
  {
    spTabellenScrollPane = new JScrollPane();
    spTabellenScrollPane.setBounds(5, 40, 215, 100);
    spTabellenScrollPane.setViewportView(getTabDaten());
  }
  return spTabellenScrollPane;
}

protected JTable getTabDaten()
{
  if (tabDaten == null)
```

```
    {
      tabDaten = new JTable();
      ...
      tabDaten.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
      tabDaten.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

// manuell eingefügt
      tabDaten.setModel(getTmDaten());
    }
    return tabDaten;
  }

 /**
   * @wbp.nonvisual location=20,190
   */
  protected MyTableModel getTmDaten()
  {
    if (tmDaten == null)
    {
      tmDaten = new MyTableModel();
    }
    return tmDaten;
  }
```

13.4.4.1.3 Codierung des lesenden Zugriffs auf die Anzeige in der GUI

Der lesende Zugriff auf die Anzeige der Daten in der GUI

- für das Feststellen markierter Einträge oder
- für das Einlesen von in der GUI neu eingegebenen Daten

erfolgt über den Delegate, also über die JTable. Die wichtigsten Methoden hierzu sind folgende **Instanzmethoden der Klasse JTable**:

- Die Methode
  `getSelectedRow()`
  gibt den Index der in der GUI markierten Zeile zurück.
- Die Methode
  `getSelectedColumn()`
  gibt den Index der in der GUI markierten Spalte zurück.
- Die Methode
  `getValue(i,j)`
  gibt den in der GUI angezeigten Wert zum Tabellenelement mit dem Index (i.j) zurück.

13.4.4.1.4 Codierung des verändernden Zugriffs auf die Tabellendaten im Modell

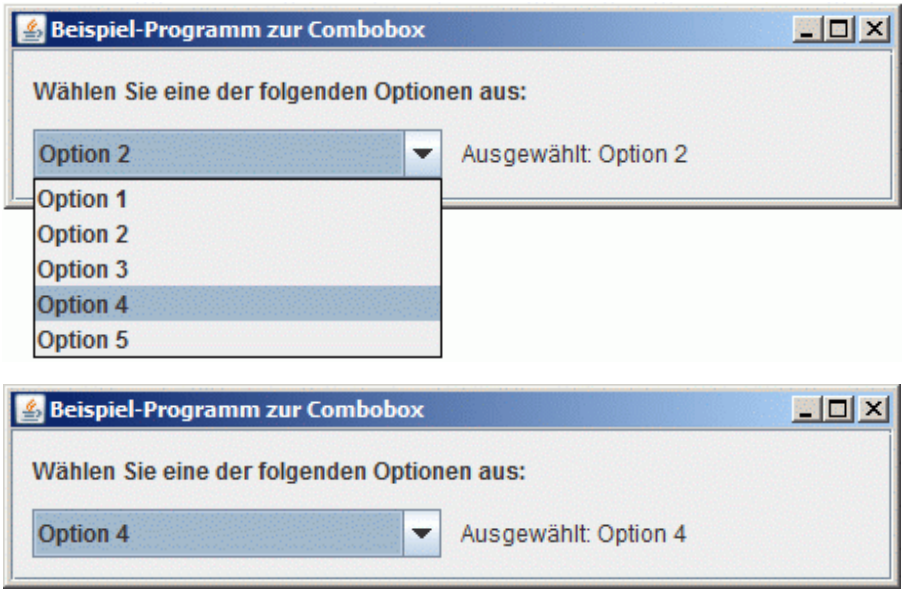
Der lesende bzw. schreibende Zugriff auf die Tabellendaten im Modell

- beim Speichern von neu in der GUI eingegebenen Daten,
- beim Laden von Daten in das Tabellenmodell (von der Festplatte oder vom Datenbankserver),
- beim Speichern der Daten des Tabellenmodells (auf der Festplatte oder im Datenbankserver)

erfolgt über die im Kapitel 13.4.4.1.1 genannten **Methoden des Tabellenmodells**.

13.4.4.2 JComboBox

Auswahlvorgänge, die die Auswahl einer Option bzw. eines Eintrags aus einer möglicherweise unterschiedlich großen Liste von Einträgen erfordern, können sehr gut mit der JCombobox programmiert werden:



Für die Programmierung einer JComboBox benötigt man folgende Komponenten:

- eine JComboBox-Komponente als Delegate
- als Modell eine Klasse, die von der Klasse DefaultComboBoxModel abgeleitet ist

13.4.4.2.1 Codierung des ComboBox-Modells

Eine als Datenmodell für eine JComboBox geeignete Klasse erhält man, indem man die jeweilige Klasse von der Klasse DefaultComboBoxModel ableitet. Die erbende Klasse erhält dann alle Methoden, die für ein ComboBox-Modell erforderlich sind. Von diesen Methoden müssen für die Anzeige einer festen Liste von Einträgen mindestens die beiden nachfolgenden Methoden überschrieben werden:

- Die Methode
  `public String getElementAt(int i)`
  sollte den i-ten Eintrag der Liste der in der JComboBox anzuzeigenden Elemente zurückgeben (mit Null geginnend indiziert).
- Die Methode
  `public int getSize()`
  sollte die Anzahl der in der JComboBox anzuzeigenden Elemente zurückgeben.

Beispiel jpr13150 zeigt ein minimales ComboBox-Modell, dass für eine JComboBox 5 Einträge mit der Beschriftung "Option 1" bis "Option 5" bereitstellt:

```
package bsp13150;

import javax.swing.DefaultComboBoxModel;

public class MyComboBoxModel extends DefaultComboBoxModel<String>
{
  ...

  private String[] optionen = {"Option 1","Option 2","Option 3","Option 4","Option 5"};

  public String getElementAt(int i)
  {
    return this.optionen[i];
  }

  public int getSize()
  {
    return this.optionen.length;
  }
}
```

Hinweis 1: Statt einer Vererbung von der Klasse DefaultComboBoxModel kann auch eine Implementierung des Interface ComboBoxModel codiert werden. Ein Beispiel hierfür ist das Projekt jpr13151.

Hinweis 2: Die Klasse DefaultComboBoxModel ist generisch. Es ist am einfachsten, für den nicht festgelegten Datentyp der Elemente des Models wie oben den Datentyp String zu wählen, also als Datentyp `DefaultComboBoxModel<String>` *zu codieren.*

13.4.4.2.2 Codierung der Anzeige der JComboBox in der GUI

Die Codierung der JComboBox erfolgt im Prinzip wieder wie die Codierung einer Komponente gemäß Kapitel 13.4.1.1, also nur mit Deklaration und get-Methode. Allerdings ist JComboBox eine generische Klasse, was vom WindowBuilder Editor noch nicht geeignet berücksichtigt wird. Daher muss der Quellcode manuell nachgearbeitet werden (s.u.). Ferner wird (analog zum Vorgehen in 13.4.3 für die JTable) ein Datenmodell (jetzt ein ComboBox-Model) als Java Bean mit dem WindowBuilder Editor erzeugt:

```
package bsp13150;

import javax.swing.JComboBox;
...

public class MainPanel extends JPanel
{
  ...
  private JComboBox<String> cmbDialogFensterAuswahl;
  private MyComboBoxModel cmmComboBoxDaten;

  ...

  protected JComboBox<String> getCmbDialogFensterAuswahl()
  {
    if (cmbDialogFensterAuswahl == null)
    {
      cmbDialogFensterAuswahl = new JComboBox<String>();

// manuell eingefügt, muss direkt nach der Instanziierung der JComboBox
// codiert werden:
      cmbDialogFensterAuswahl.setModel(getCmmComboBoxDaten());

      cmbDialogFensterAuswahl.addActionListener(new ActionListener()
      {
        public void actionPerformed(ActionEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });
      cmbDialogFensterAuswahl.setSelectedIndex(1);
      cmbDialogFensterAuswahl.setBounds(10, 40, 210, 25);

    }
    return cmbDialogFensterAuswahl;
  }

  /**
   * @wbp.nonvisual location=20,120
   */
  private MyComboBoxModel getCmmComboBoxDaten()
  {
    if (cmmComboBoxDaten == null)
    {
      cmmComboBoxDaten = new MyComboBoxModel();
    }
    return cmmComboBoxDaten;
  }
  ...
}
```

Die Codierung der obigen get-Methode für die JComboBox kann allerdings nicht vollständig mit dem WindowBuilder Editor erzeugt werden:

- Mit dem Eintrag Choose component im Abschnitt System der Palette des WindowBuilder Editor sollte zunächst ein Objekt der Klasse angelegt werden, die das ComboBox-Modell enthält.
- Mit dem Eintrag JComboBox im Abschnitt Components der Palette des WindowBuilder Editor kann eine JComboBox-Komponente erzeugt und in der GUI angeordnet werden.
- Für den zur JComboBox generierten Quellcode müssen folgende Änderungen vorgenommen werden. Der Datentyp JComboBox muss genauer als JComboBox<String> spezifiziert werden an folgenden Stellen:
    - Deklaration der JComboBox
    - Rückgabert der get-Methode für die JComboBox
    - Konstruktor-Aufruf für die JComboBox
- Durch manuelles Codieren der Anweisung

```
cmbDialogFensterAuswahl.setModel(getCmmComboBoxDaten());
```

in der get-Methode zur JComboBox-Komponente wird die Komponente mit dem Model verbunden. Es ist notwendig, dass dieser Methodenaufruf vor allen Anweisungen eingefügt wird, die die JComboBox konfigurieren.

- Über die Registerkarte Properties des WindowBuilder Editor können Eigenschaften der JComboBox konfiguriert werden - z. Bsp. der vorausgewählte Eintrag festgelegt werden. Achten Sie darauf, zum Abschluss dieses Schritts noch einmal sichergestellt wird, dass der Aufruf der Methode setModel(ComboBoxModel) vor allen Anweisungen - auch vor allen generierten Anweisungen - steht, die die JComboBox konfigurieren.
- Über das Kontextmenü zur JComboBox-Komponente im WindowBuilder Editor kann dann der Komponente eine Ereignisbehandlung zugewiesen werden. Verwenden Sie für die Ereignisbehandlung einen ActionListener!

13.4.4.2.3 Codierung des lesenden Zugriffs auf die Anzeige in der GUI

Wie in Kapitel 13.4.4.2.2 sichtbar erfolgt die Ereignisbehandlung für ActionEvent-Ereignisse zur JComboBox in einer statischen Methode zeigeAuswahlAn(MainePanel) der Klasse bsp13150.Verarbeitung:

```
package bsp13150;

public class Verarbeitung
{
  ...

  public static void zeigeAuswahlAn(MainPanel mp)
  {
// Ausgewählte Option feststellen
    String auswahl = mp.getCmbDialogFensterAuswahl().getSelectedItem().toString();
// GUI aktualisieren
    mp.getTfAuswahlAnzeige().setText("Ausgewählt: "+auswahl);
  }
}
```

Man sieht, dass der lesende Zugriff auf das in der JComboBox ausgewählte Element denkbar einfach ist und dass dieser Zugriff mit der Methode

```
public Object getSelectedItem()
```

erfolgt. Zusätzlich dazu gibt es noch die Methode

```
public int getSelectedIndex()
```

die den Index des selektierten Eintrags zurückgibt. Wird die JComboBox als editierbar konfiguriert und wird ein neuer Eintrag in die JComboBox eingegeben, so kann man diese Tatsache daran erkennen, dass in diesem Fall die Methode getSelectedIndex() den Wert -1 zurückgibt (s.u.).

Auch für die JComboBox gilt somit wieder:

- Der lesende Zugriff auf die in der GUI angezeigten Einträge sowie
- der lesende Zugriff auf neu in der GUi eingegebene Einträge (vgl. 13.4.4.2.4)

erfolgt **über den Delegate, also über die JComboBox**.

13.4.4.2.4 Codierung des verändernden Zugriffs auf die Daten im ComboBox-Modell

Für die Erklärung des schreibenden Zugriffs auf die Daten zur JComboBox - also für die Erklärung des schreibenden Zugriffs auf das ComboBox-Modell - wird das obige Beispiel zur JComboBox zunächst wie folgt zu Beispiel jpr13152 erweitert:

- Konfiguration der JComboBox mit dem WindowBuilder Editor als editierbar.
- Erweiterung des Modells für Veränderungen der Einträge in der JComboBox duch Einfügung einer Methode insertElementAt(String o, int i) (sofern das ComboboxModel mit dem Zusatz <String> erzeugt wurde).

Damit zur Laufzeit Einträge in die von einer JComboBox dargestellte Liste eingefügt werden können, muss das JComboBox-Modell um eine Methode

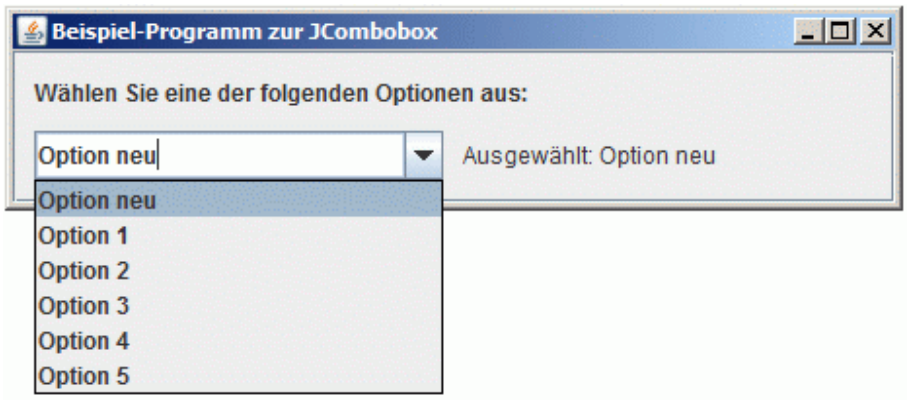
```
public void insertElementAt(String o, int i)
```

ergänzt werden. In dieser Methode müssen mindestens folgenden beiden Schritte durchgeführt werden:

- Einfügung des neuen Elementes, das als Parameter String o an die Methode übergeben wurde, in die Liste der Einträge der JComboBox im Modell an Index i.
- Benachrichtigung der GUI, dass die Anzeige der JComboBox (wegen der Veränderung der Daten im Modell) aktualisiert werden muss.

Beispiel jpr13152 hat folgende Benutzeroberfläche:

Die JComboBox ist nun editierbar. Nach Eingabe eines neuen Eintrags wird dieser in die Liste der anzuzeigenden Elemente an erster Stelle eingefügt:



Um diese Funktionalität zu erreichen, wurde die JComboBox in der Registerkarte Properties des WindowBuilder Editor als editierbar konfiguriert, was im Quellcode zu folgender Ergänzung geführt hat:

```
  protected JComboBox<String> getCmbDialogFensterAuswahl()
  {
    if (cmbDialogFensterAuswahl == null)
    {
      cmbDialogFensterAuswahl = new JComboBox<String>();
      ...
      cmbDialogFensterAuswahl.setEditable(true);
      ...
    }
    return cmbDialogFensterAuswahl;
  }
```

Im Modell wurde folgende Methode insertElementAt(String o, int i) eingefügt:

```
package bsp13152;

...

public class MyComboBoxModel extends DefaultComboBoxModel<String>
{
  ...

  private String[] optionen = {"Option 1","Option 2","Option 3","Option 4","Option 5"};

  ...

  public void insertElementAt(String o, int i)
  {
// neuen Eintrag im Modell speichern
    String[] optionenNeu = new String[optionen.length+1];
    for (int k=0;k<i;k++)
      optionenNeu[k] = optionen[k];
    optionenNeu[i] = o;
```

```
    for (int k=i+1;k<optionenNeu.length;k++)
      optionenNeu[k] = optionen[k-1];
    this.optionen = optionenNeu;
// JComboBox informieren, dass die Anzeige aktualisiert werden muss
    this.fireContentsChanged(this, 0, this.optionen.length);
  }
}
```

Nach diesen Vorbereitungen kann jetzt die Speicherung neuer Einträge im Modell in der Methode zeigeAuswahlAn(MainPanel) der Ereignisbehandlung von ActionEreignissen ergänzt werden. Hierzu wird die gleichnamige Methode aus Beispiel jpr13150 um eine Anweisung erweitert, die den neuen - aus der JComboBox eingelesenen - Eintrag am Anfang der Liste der Einträge im Modell einfügt:

```
package bsp13152;

public class Verarbeitung
{
  ...

  public static void zeigeAuswahlAn(MainPanel mp)
  {
// Ausgewählte Option feststellen
    String auswahl = mp.getCmbDialogFensterAuswahl().getSelectedItem().toString();
// GUI aktualisieren
    mp.getTfAuswahlAnzeige().setText("Ausgewählt: "+auswahl);
// Wenn ein neuer Eintrag existiert, muss dieser im Tabellenmodell gespeichert werden
    if (mp.getCmbDialogFensterAuswahl().getSelectedIndex()==-1)  // neuer Eintrag
    {
// Zugriff auf das Model über die JComboBox
      MyComboBoxModel myCBM = (MyComboBoxModel)mp.getCmbDialogFensterAuswahl().getModel();
// Neues Element als erster Eintrag in die JComboBox einfügen
      myCBM.insertElementAt(auswahl, 0);
    }
  }
}
```

Auch für die JComboBox und das zugehörige ComboBoxModel gilt somit wieder:

- Der schreibende Zugriff auf die Daten im Datenmodell,
- auch beim Speichern von neu in der GUI eingegebenen Daten im Datenmodell

erfolgt **über die Methoden des Modells**.

Das Vorgehen zum Anlegen einer JComboBox kann in folgendem Rezept zusammengefasst werden:

Rezept für das Anlegen einer JComboBox

- (editierbare oder auch nichteditierbare) JComboBox mit dem WindowBuilder Editor in die GUI einfügen
- ComboBox Modell aus jpr13150 (nicht editierbare Combobox) oder jpr13152 (editierbare Combobox) mit dem WindowBuilder Editor als Java Bean neben der GUI einfügen.
- Im Model Einträge für die Combobox korrigieren.
- Model zur Combobox hinzufügen mit der Methode setModel(..) (manuell codierter Code).
- Code für den Zugriff auf die JComboBox bzw. das Datenmodell aus bsp13150.Verarbeitung bzw. bsp13152.Verarbeitung entnehmen.

13.4.4.3 JList

Eine JList ermöglicht wie eine JComboBox die Auswahl aus einer Liste von Einträgen. Anders als bei einer JComboBox kann eine JList jedoch so konfiguriert werden, dass mehrere Einträge gleichzeitig ausgewählt werden können. Beispiel jpr13160 zeigt eine JList mit einer kurzen unveränderlichen Liste von Einträgen, aus denen ein Eintrag ausgewählt werden soll. Eine solche JList kann ohne umgebende JScrollPane codiert werden:

Für die Programmierung einer JList mit Datenmodell benötigt man folgende Komponenten:

- eine JList-Komponente als Delegate
- als Modell eine Klasse, die von der Klasse AbstractListModel abgeleitet ist.

13.4.4.3.1 Codierung des Listenmodells

Eine Klasse, die als Modell für eine JList verwendet werden kann, erhält man durch Implementieren der Klasse AbstractListModel. In der erbenden Klasse müssen mindestens die folgenden beiden Methoden codiert werden:

- Die Methode
  `public String getElementAt(int i)`
  muss den i-ten Eintrag im Datenmodell zurückgeben.
- Die Methode
  `public int getSize()`
  muss die Anzahl der Einträge im Datenmodell zurückgeben.

Beispielprojekt jpr13160 zeigt ein Modell, das diese beiden Methoden bereitstellt. Man beachte, dass die Klasse AbstractListModel eine generische Klasse ist, weshalb man am einfachsten den in der generischen Klasse unbestimmten Datentyp als String wählt:

```
package bsp13160;

import javax.swing.AbstractListModel;

public class MyListModel extends AbstractListModel<String>
{
  ...

  String[] eintraege = {"Option 1","Option 2","Option 3","Option 4"};

  public String getElementAt(int i)
  {
    String el = null;
    if (0<=i && i < eintraege.length)
      el = eintraege[i];
    return el;
  }

  public int getSize()
  {
    return eintraege.length;
  }
}
```

13.4.4.3.2 Codierung der Anzeige der JList in der GUI

Zur Codierung der Anzeige der JList in der GUI erzeugt man wie bei den anderen Komponenten dieses Kapitels, jedoch mit etwas veränderter Vorgehensweise zunächst

- eine Komponente vom Typ JList, für die
- über die Registerkarte Properties der selectionMode auf SINGLE gesetzt wird und für die
- eine Ereignisbehandlung mit KeyListener und MouserListener zugeordnet wird. Dabei implementiere man die Ereignisse keyReleased und mouseReleased, da es bei der Verwendung einer JList darauf ankommt, auf welchem Eintrag die Taste bzw. die Maustaste *losgelassen* wird.
- Da JList ein generische Klasse ist, füge man in der Deklaration der JList-Variablen, im Rückgabewert der get-Methode für die JList sowie im Konstruktor-Aufruf für die JList den Ausdruck **<String>** hinzu, um den in der JList variabel gehaltenen Datentyp als Datentyp String

festzulegen;

```
private JList<String> lstOptionsAuswahl;

...

protected JList<String> getLstOptionsAuswahl()
{
  if (lstOptionsAuswahl == null)
  {
    lstOptionsAuswahl = new JList<String>();
    ...
  }
  return lstOptionsAuswahl;
}
```

- Über die Palette, Abschnitt System, Eintrag Choose component muss ein Objekt der Klasse MyListModel als unsichtbare Java Bean gemäß 13.4.3.4 eingefügt werden. Diese Bean wird anschließend in der GUI des WindowBuilder Editor angezeigt. Als Variablennamen für das ListModel wähle man einen Namen, der mit lsm beginnt.
- Man füge folgende Anweisung manuell in die get-Methode für die JList ein, um die JList mit dem ListModel zu verbinden:

```
// manuell eingefügt
      lstOptionsAuswahl.setModel(getLsmMyListModel());
```

Auf diese Weise erhält man für die GUI folgenden Quellcode:

```
package bsp13160;

import javax.swing.JList;
...

public class MainPanel extends JPanel
{

  ...
  private JList<String> lstOptionsAuswahl;

  ...

  protected JList<String> getLstOptionsAuswahl()
  {
    if (lstOptionsAuswahl == null)
    {
      lstOptionsAuswahl = new JList<String>();
      lstOptionsAuswahl.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseReleased(MouseEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });
      lstOptionsAuswahl.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyReleased(KeyEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });
      lstOptionsAuswahl.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
      lstOptionsAuswahl.setBounds(10, 40, 205, 75);

// manuell eingefügt
      lstOptionsAuswahl.setModel(getLsmMyListModel());
    }
    return lstOptionsAuswahl;
  }
```

13.4.4.3.3 Codierung des lesenden Zugriffs auf die Anzeige in der GUI

Gemäß dem allgemeinen Prinzip der Aufgabenverteilung im Java Delegate Model geschieht der lesende Zugriff auf den in der GUI angezeigten

markierten Eintrag bzw. auf markierte Einträge über den Delegate, also die JList. Hierzu stellt die Klasse JList folgende Methoden bereit (sofern das ComboboxModel mit dem Zusatz <String> erzeugt wurde):

- Die Methoden
  ```
  public String getSelectedValue()
  public String[] getSelectedValues()
  ```
  geben den bzw. die markierten Einträge zurück.
- Die Methoden
  ```
  public int getSelectedIndex
  public[] int getSelectedIndeces
  ```
  geben den Index bzw. die Indizes der markierten Einträge zurück.

Entsprechend wird in der Methode Verarbeitung.zeigeAuswahlAn(MainPanel) wie folgt der vom Benutzer markierte Index festgestellt - für die JList im Beispiel jpr13160 ist ja oben eine SINGLE Selektion eingestellt worden:

```
package bsp13160;

public class Verarbeitung
{
  ...

  public static void zeigeAuswahlAn(MainPanel mp)
  {
// Ausgewählte Option feststellen
    String auswahl = mp.getLstOptionsAuswahl().getSelectedValue();
    if (auswahl != null)
    {
      mp.getTfAuswahlAnzeige().setText("Ausgewählt: "+auswahl);
    }
  }
}
```

13.4.4.3.4 Codierung des verändernden Zugriffs auf die Daten im Listen-Modell

Entsprechend dem Konzept des Java Delegate Model erfolgt der schreibende Zugriff auf die Daten der JList mit einer Methode,

- die das einzufügende Element in die Liste im Datenmodell einfügt und
- die anschließend die JList-Komponente informiert, dass die Anzeige der Daten aktualisiert werden muss.

Als Beispiel für den schreibenden Zugriff auf die Daten einer JList wird das Beispiel jpr13160 wie folgt zu einem Beispiel jpr13161 erweitert:

- Einfügung der JList-Komponente in die GUI nicht direkt, sondern mit umgebender JScrollPane, damit auch nach Erweiterung der JList eine geeignete Anzeige der JList in der GUI erfolgt.
- Einfügung einer Methode in das Listen-Modell,
  - die die Erweiterung der Liste im Modell durchführt und
  - die die Benachrichtigung der JList-Komponente über die Veränderung der Daten im Modell durchführt.

Beispiel jpr13161 hat folgende Benutzeroberfläche:

In einem Textfeld dieser GUI kann ein neuer Eintrag eingegeben werden:

Nach Klick auf den Button "Eintrag zur Liste hinzufügen" wird dieser Eintrag zu Beginn der Liste eingefügt, die in der JList angezeigt wird:

Wie man an obigen Bildschirmausdrucken sieht, ist die JList jetzt von einer JScrollPane umgeben. Dies wurde mit dem WindowBuilder Editor codiert, der folgenden Quellcode erstellt hat:

```
package bsp13161;

import javax.swing.JList;
import javax.swing.JScrollPane;
...

public class MainPanel extends JPanel
{
  ...
  private JList<String> lstOptionsAuswahl;
  private JScrollPane spListScrollPane;

  ...

  protected JList<String> getLstOptionsAuswahl()
  {
    if (lstOptionsAuswahl == null)
    {
      lstOptionsAuswahl = new JList<String>();
      lstOptionsAuswahl.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseReleased(MouseEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });
      lstOptionsAuswahl.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyReleased(KeyEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });
      lstOptionsAuswahl.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

// manuell eingefügt
      lstOptionsAuswahl.setModel(getLsmMyListModel());
    }
    return lstOptionsAuswahl;
  }
```

```
  private JScrollPane getSpListScrollPane()
  {
    if (spListScrollPane == null)
    {
      spListScrollPane = new JScrollPane();
      spListScrollPane.setBounds(new Rectangle(10, 40, 205, 75));
      spListScrollPane.setViewportView(getLstOptionsAuswahl());
    }
    return spListScrollPane;
  }

  ...
}
```

Um den schreibenden Zugriff auf die Daten des Modells zu erreichen, wurde eine Methode public void erweitere(String o, int i) in das Modell eingefügt. Diese Methode führt das Einfügen der Daten in das Modell und die Benachrichtigung des Delegate durch. Der Name der Methode ist frei wählbar, es gibt keine passende Methode, die überschrieben werden kann oder sollte:

```
package bsp13161;

...

public class MyListModel extends AbstractListModel<String>
{
  ...

  String[] eintraege = {"Option 1","Option 2","Option 3","Option 4","Option 5"};

  ...

  public void erweitere(String o, int i)
  {
// neuen Eintrag im Modell speichern
    String[] eintraegeNeu = new String[eintraege.length+1];
    for (int k=0;k<i;k++)
      eintraegeNeu[k] = eintraege[k];
    eintraegeNeu[i] = o;
    for (int k=i+1;k<eintraegeNeu.length;k++)
      eintraegeNeu[k] = eintraege[k-1];
    this.eintraege = eintraegeNeu;
// GUI informieren, dass die Anzeige aktualisiert werden muss
    this.fireContentsChanged(this, 0, eintraege.length);
  }
}
```

Wie man im Quellcode leicht abliest,

- fügt die Methode erweitere(String,int) das als Parameter übergebene Objekt, umgewandelt in String, an i-ter Stelle in die Liste des Datenmodells ein und
- benachrichtigt anschließend die GUI, dass der Datenbestand des Modells geändert worden ist.

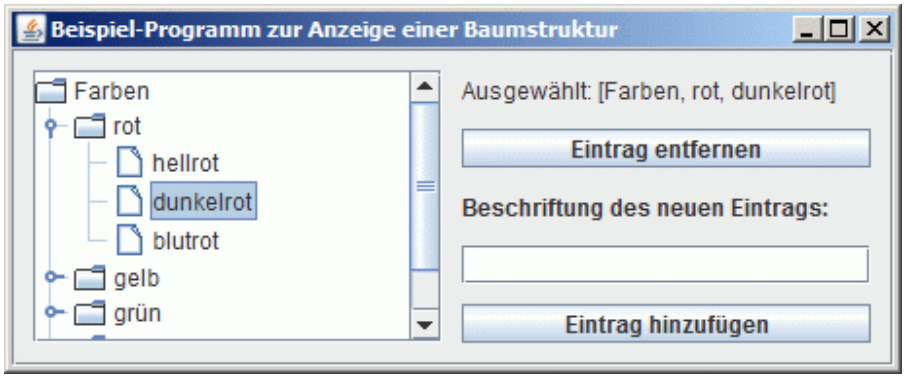
Die Methode erweitere(String,int) wird nun in der verarbeitenden Methode fuegeEintragHinzu(MainPanel) zum Button "Eintrag zur Liste hinzufügen" wie folgt aufgerufen:

```
package bsp13161;

public class Verarbeitung
{
  ...

  public static void fuegeEintragHinzu(MainPanel mp)
  {
// Neuen Eintrag lesen
    String ein = mp.getTfEintrag().getText().trim();
// Auf das Datenmodell zugreifen
    MyListModel myLM = (MyListModel)mp.getLstOptionsAuswahl().getModel();
// Eintrag hinzufügen
    if (!ein.equals(""))
      myLM.erweitere(ein, 0);
// Anzeige im Eingabefeld löschen
    mp.getTfEintrag().setText("");
  }
```

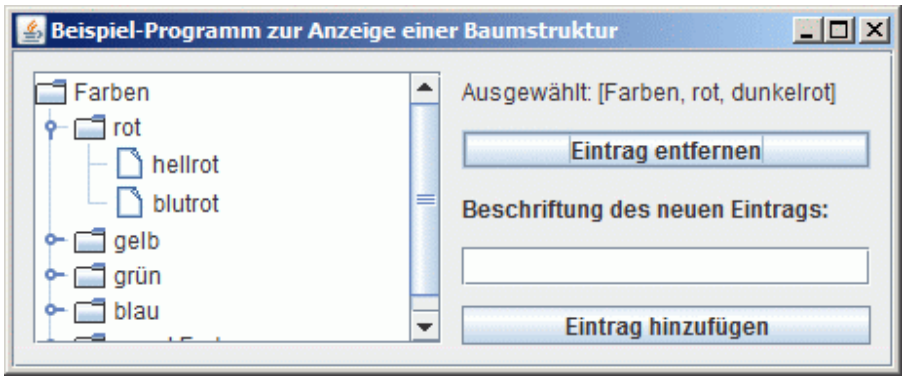
}

13.4.4.4 JTree

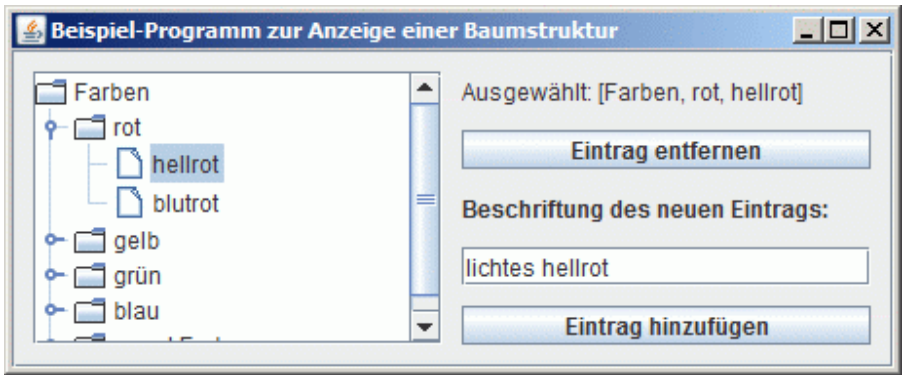
Im Beispiel jpr13170 zum JTree sind alle im Zusammenhang mit Delegate und Model interessanten Vorgänge auscodiert. Das Programm ermöglicht das Einfügen von Blättern bzw. Knoten in den Tree sowie das Löschen von Blättern und Knoten:



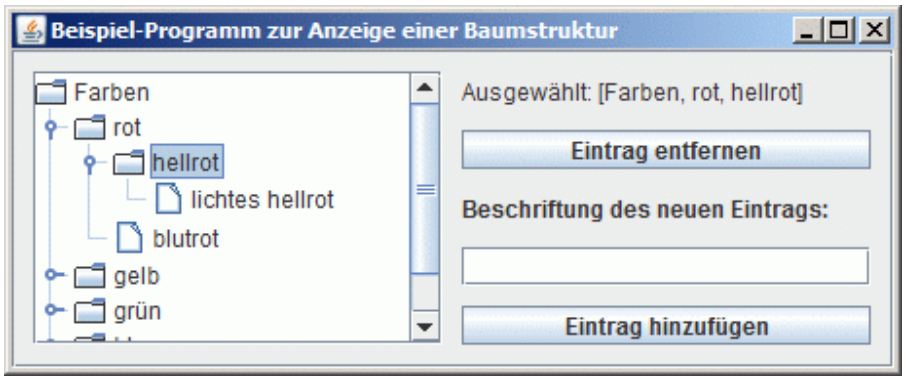
Klickt man auf den Button "Eintrag entfernen", so wird der im Baum markierte Eintrag - Blatt oder Knoten - entfernt:



Markiert man ein Blatt oder einen Knoten, so wird unter diesem Element ein neues Blatt eingefügt, das den in der GUI im Feld "Beschriftung des neuen Eintrags" angegenenen Namen erhält:



War der markierte Eintrag ein Blatt, wird dieser dabei zu einem Knoten umgewandelt:



13.4.4.4.1 Codierung des Tree-Modells

Die Codierung eines Modells für einen JTree ist denkbar einfach:

- Man codiere eine Klasse, die von der Klasse javax.swing.tree.DefaultTreeModel erbt und
- in deren Konstruktor der Teil des Baums in das Tree-Modell eingefügt wird, der bei der erstmaligen Anzeige des Baums angezeigt werden soll.

Die als Modell zu codierende Klasse hat im Beispiel jpr13170 somit folgende Form:

```
package bsp13170;

import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.DefaultMutableTreeNode;

public class MyTreeModel extends DefaultTreeModel
{
  private static final long serialVersionUID = 710L;

  public MyTreeModel()
  {
// 1. Tree mit dem Root Knoten "Farben" instanziieren
    super(new DefaultMutableTreeNode("Farben"));

// 2. String-Array für die obersten Elemente anlegen
   String spalte1[] = {"rot","gelb","grün","blau","sonst.Farben"};
// 3. String-Array für die zweitobersten Elemente anlegen
   String spalte2[][] =
   {
     {"hellrot","dunkelrot","blutrot"},
     {"hellgelb","goldgelb","gelbgrün"},
     {"hellgrün","grasgrün","dunkelgrün","blaugrün"},
     {"hellblau","mittelblau","himmelblau","stahlblau","dunkelblau"},
     {"hellorange","violett","gold"}
   };

// 4. Array-Struktur in die Knoten (nodes) des Baums kopieren
//    Als root Knoten wurde oben ein DefaultMutableTreeNode angegeben.
   DefaultMutableTreeNode root = (DefaultMutableTreeNode)this.getRoot();
   for (int i=0;i<spalte1.length;i++)
   {
     DefaultMutableTreeNode knoten1 = new DefaultMutableTreeNode(spalte1[i]);
     root.add(knoten1);
     for (int j=0;j<spalte2[i].length;j++)
     {
       DefaultMutableTreeNode knoten2 = new DefaultMutableTreeNode(spalte2[i][j]);
       knoten1.add(knoten2);
     }
   }
  }
}
```

13.4.4.4.2 Codierung der Anzeige des JTree in der GUI

Da der von einem JTree benötigte Platz beim Erweitern von Knoten stark zunehmen kann, ist es immer zweckmäßig, den JTree in einer JScrollPane anzuordnen. Dieser Vorgang kann nicht vollständig mit dem WindowBuilder Editor codiert werden:

- Zunächst wird in der Palette des WindowBuilder Editors, Abschnitt Containers die Komponente JScrollPane ausgewählt und in der GUI mit drag-and-drop angeordnet.
- Anschließend wird in der Palette des WindowBuilder Editors, Abschnitt Components die Komponenten JTree ausgewählt und durch Klick mit der linken Maustaste in den Bereich der JScrollPane eingefügt, der im WindowBuilder Editor als Viewport angezeigt wird.
- Einfügen des Tree-Modells bspß170.MyTreeModel mit dem WindowBuilder Editor, Abschnitt System, Eintrag "Choose component". Es ensteht folgender Quellcode:

```
  private MyTreeModel myTreeModel;

  ...

  /**
   * @wbp.nonvisual location=25,200
   */
// Methode manuell auf protected gesetzt
  protected MyTreeModel getMyTreeModel()
  {
    if (myTreeModel == null)
    {
      myTreeModel = new MyTreeModel();
    }
    return myTreeModel;
  }
```

- Der vom WindowBuilder Editor erzeugte Quellcode zum JTree wird so bearbeitet, dass nach dem Instanziieren des JTree diesem als Model ein Objekt der Klasse MyTreeModell zugewiesen wird:

```
// 1. Dem JTree das Model zuweisen
      treeBaumAnzeige.setModel(getMyTreeModel());
```

- Durch manuelles Einfügen des nachfolgenden Quellcode wird der Wurzeleintrag des JTree als vorselektiert festgelegt - dies vermeidet unnötige NullPointer-Exceptions:

```
// 2. Wurzeleintrag des Tree als selected festlegen
      treeBaumAnzeige.setSelectionPath(
        new TreePath(
            getMyTreeModel().getRoot()));
```

- Durch manuelles Einfügen des nachfolgenden Quellcode wird für das Beispiel jpr13170 konfiguriert, dass immer nur ein Knoten oder ein Blatt des JTree markiert werden kann:

```
// 3. Einstellen, dass nur eine einfache Auswahl möglich ist
      DefaultTreeSelectionModel dTSM = new DefaultTreeSelectionModel();
      dTSM.setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
      treeBaumAnzeige.setSelectionModel(dTSM);
```

- Über den WindowBuilder Editor wird dem Tree eine Ereignisbehandlung für Tastatur- und Mausereignisse zugewiesen. Dabei muss darauf geachtet werden, dass die Ereignisse keyReleased und mouseReleased behandelt werden.

Insgesamt erhält man für die GUI folgenden Quellcode:

```
package bsp13170;

import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.tree.DefaultTreeSelectionModel;
...

public class MainPanel extends JPanel
{

  private JScrollPane spTreePane;
  private MyTreeModel myTreeModel;
  private JTree treeBaumAnzeige;
  ...

  private JScrollPane getSpTreePane()
  {
    if (spTreePane == null)
```

```
    {
      spTreePane = new JScrollPane();
      spTreePane.setBounds(10, 10, 210, 140);
      spTreePane.setViewportView(getTreeBaumAnzeige());
    }
    return spTreePane;
  }

  /**
   * @wbp.nonvisual location=25,200
   */
// Methode manuell auf protected gesetzt
  protected MyTreeModel getMyTreeModel()
  {
    if (myTreeModel == null)
    {
      myTreeModel = new MyTreeModel();
    }
    return myTreeModel;
  }

  protected JTree getTreeBaumAnzeige()
  {
    if (treeBaumAnzeige == null)
    {
      treeBaumAnzeige = new JTree();
      treeBaumAnzeige.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseReleased(MouseEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });
      treeBaumAnzeige.addKeyListener(new KeyAdapter()
      {
        @Override
        public void keyReleased(KeyEvent arg0)
        {
          Verarbeitung.zeigeAuswahlAn(getMainPanel());
        }
      });

// Manuelle Ergänzungen des Quelltextes:
// 1. Dem JTree das Model zuweisen
      treeBaumAnzeige.setModel(getMyTreeModel());
// 2. Wurzeleintrag des Tree als selected festlegen
      treeBaumAnzeige.setSelectionPath(
        new TreePath(
            getMyTreeModel().getRoot()));
// 3. Einstellen, dass nur eine einfache Auswahl möglich ist
      DefaultTreeSelectionModel dTSM = new DefaultTreeSelectionModel();
      dTSM.setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
      treeBaumAnzeige.setSelectionModel(dTSM);
    }
    return treeBaumAnzeige;
  }

  ...
}
```

Man sieht, dass bei der Codierung der GUI für einen JTree der vom WindowBuilder Editor erzeugte Quellcode noch wie oben erklärt nachbearbeitet werden muss.

13.4.4.4.3 Codierung des lesenden Zugriffs auf die Anzeige in der GUI

Für den lesenden Zugriff auf die Anzeige des Trees in der GUI, also für die Feststellung des bzw. der markierten Elemente des Trees ist gemäß dem Java Delegate Modell ein Zugriff auf den Delegate erforderlich, also auf den JTree. Die Klasse JTree stellt hierzu folgende Methoden bereit:

- Die Methoden

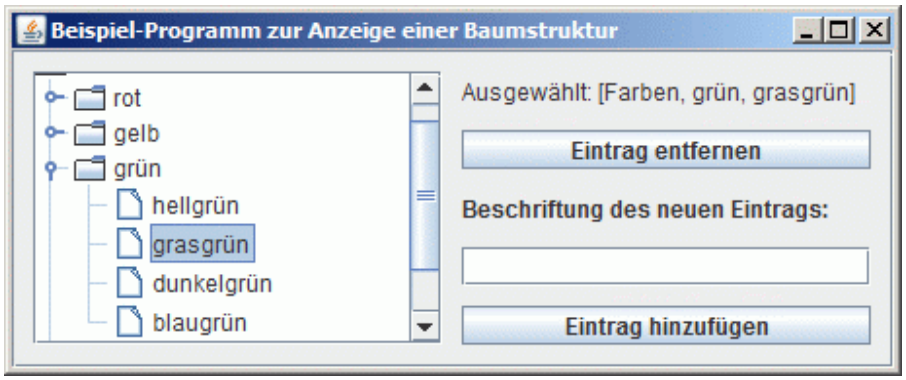
```
public TreePath getSelectionPath
```

```
public TreePath[] getSelectionPaths
```

liefern den bzw. die markierten Einträge des Trees als Objekte der Klasse TreePath. Der Inhalt dieser Objekte kann mit der Instanzmethode toString() der Klasse TreePath in einen String umgewandelt werden, der folgende Form hat:

```
[Wurzelknoten,Knoten,...,Knoten,Blatt ]
```

Mit den üblichen Methoden der Klasse StringTokenizer und String können aus diesem String die Namen der Knoten bzw. des Blatts am Ende des Pfades extrahiert werden. Im Beispielprogramm wird einfach der gesamte in String umgewandelte TreePath als "Ausgewählt" angezeigt:



Die Feststellung des markierten Eintrags - für Beispiel jpr13170 ist SINGLE_TREE_SELECTION eingestellt - geschieht also wie folgt in der verarbeitenden Methode zu den Maus- und Tastaturereignissen im Bereich des JTree:

```
package bsp13170;

import javax.swing.tree.TreePath;
...

public class Verarbeitung
{
  ...

  public static void zeigeAuswahlAn(MainPanel mp)
  {
// Ausgewählte Option feststellen
    String auswahl = "";
// Beim Erweitern des Baums mit einer Maus wird null zurückgegeben, wenn
// kein Eintrag selektiert ist!
    TreePath tp = mp.getTreeBaumAnzeige().getSelectionPath();
    if (tp == null)
      auswahl = "Keine Auswahl vorgenommen";
    else
      auswahl = "Ausgewählt: "+tp.toString();
// GUI aktualisieren
    mp.getTfAuswahlAnzeige().setText(auswahl);
  }
  ...
}
```

13.4.4.4.4 Codierung des verändernden Zugriffs auf die Daten im Tree-Modell

Der schreibende Zugriff auf den Baum erfolgt gemäß dem Java Delegate Modell durch Zugriff auf das Tree-Modell, das seinerseits dann die Aufgabe hat, die GUI über die Veränderung der Daten zu informieren. Die Besonderheit beim JTree ist hier, dass das oben in einfache Weise aus der Klasse DefaultTreeModel abgeleitete Tree-Modell schon alle Methoden geerbt hat, die für einen schreibenden Zugriff notwendig sind. *Entsprechend ist es für das Tree-Modell nicht notwendig, eine Methode für das Einfügen von Blättern in den Tree zu codieren. Alle entsprechenden Methoden sind bereits in der Klasse DefaultTreeModel vorhanden und werden von dieser Klasse geerbt.*

Der zu erstellende Quellcode für das Einfügen von Daten in das Tree-Modell besteht also lediglich im Codieren von geeigneten Verfahren für das Einfügen oder Löschen von Blättern. Diese Verfahren verwenden die entsprechenden Methoden des Tree-Modells und sind im Beispiel jpr13170 in folgenden verarbeitenden Methoden implementiert:

```
package bsp13170;

import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;
import javax.swing.JOptionPane;
import javax.swing.JTree;
```

```
public class Verarbeitung
{
  ...
  public static void fuegeEintragHinzu(MainPanel mp)
  {
//  Neuen Eintrag aus der GUI lesen
    String neuEintrag;
    neuEintrag = ...

//  Markierten Knoten identifizieren
    JTree myT = mp.getTreeBaumAnzeige();
    DefaultMutableTreeNode dmtn =
      (DefaultMutableTreeNode)myT.getLastSelectedPathComponent();
    if (!neuEintrag.equals(""))
    {
      if (dmtn != null)
      {
//  Blatt in Baum einfügen
        mp.getMyTreeModel().insertNodeInto(
            new DefaultMutableTreeNode(neuEintrag),
            dmtn,
            dmtn.getChildCount()
        );
        ...
      }
      ...
    }
  }

  public static void loescheEintrag(MainPanel mp)
  {
// Markierten Knoten identifizieren
    JTree myT = mp.getTreeBaumAnzeige();
    DefaultMutableTreeNode dmtn =
      (DefaultMutableTreeNode)myT.getLastSelectedPathComponent();
    if (dmtn != null)
    {
// Der root Knoten darf nicht entfernt werden
      if (dmtn != mp.getMyTreeModel().getRoot())
      {
// Blatt oder Knoten aus dem Baum entfernen
        mp.getMyTreeModel().removeNodeFromParent(dmtn);
      }
      ...
    }
    ...
  }
}
```

Man sieht, dass das Einfügen bzw. Löschen von Blättern bzw. Knoten mit den folgenden beiden Methoden der Klasse DefaultTreeModel erfolgt:

- `insertIntoNode(MutableTreeNode,MutableTreeNode,int)` - fügt den als erster Parameter angegebenen TreeNode als Blatt an den TreeNode an, der als zweiter Parameter angegeben wird. Der dritte Parameter legt fest, an welcher Stelle der Reihenfolge der bereits vorhandenen Blätter eingefügt werden soll - nach dem Blatt mit dem als dritter Parameter angegebenen Index.
- `removeNodeFromParent(MutableTreeNode)` - entfernt den angegebenen TreeNode aus dem Tree-Model

Neben diesen beiden Methoden für das Einfügen und Löschen von Blättern und Knoten im Tree zeigt der obige Quellcode weitere Methoden, die das Identifizieren des markierten Blatts bzw. Knoten ermöglichen bzw. die die Identifzierung des Wurzelknotens (root node) - also des tiefsten Knotens im Baum - ermöglichen.

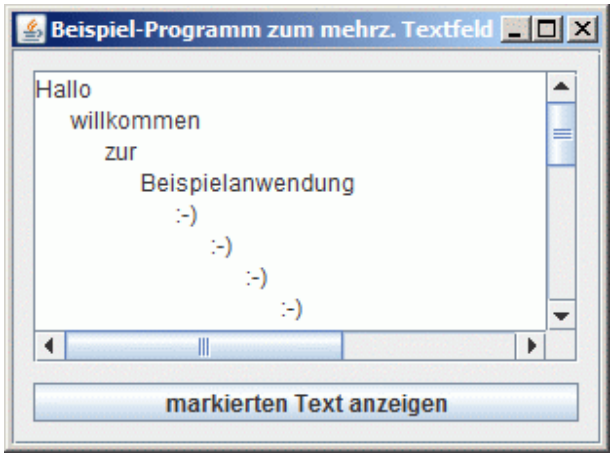
13.4.5 Editierbare Oberflächenelemente

Die zu der Gruppe der editierbaren Oberflächenelemente gehörende Komponente JTextField wurde schon in Kapitel 13.1 eingeführt. Die Gruppe der editierbaren Oberflächenelemente besteht ferner aus den Komponenten JTextArea, JPasswordField, JEditorPane und JTextPane. Diese vier Komponenten werden im aktuellen Kapitel vorgestellt.

Hinweis: Für alle Komponenten dieses Kapitels kann problemlos darauf verzichtet werden, ein Datenmodell zu codieren.

13.4.5.1 JTextArea

Die JTextArea unterscheidet sich in Bezug auf die Codierung nur unwesentlich von dem JTextField, sie ist eine mehrzeilige Variante des JTextField:



13.4.5.1.1 Codierung der JTextArea

Da eine JTextArea mehrzeilig ist,

- da sich zur Laufzeit die Anzahl der Zeilen erhöhen oder verringern kann,
- da sich ferner auch die Länge der Zeilen zur Laufzeit erhöhen und verringern kann,

ist es zweckmäßig, eine JTextArea in eine JScrollPane einzubetten. Die Codierung der JTextArea mit dem WindowBuilder Editor geschieht somit wie folgt:

- Zunächst sollte mit dem WindowBuilder Editor die JScrollPane in die GUI eingebettet werden (WindowBuilder Editor / Palette / Containers / JScrollPane).
- Im zweiten Schritt sollte dann die JTextArea aus der Palette des WindowBuilder Editors, Abschnitt Components ausgewählt und durch Klick mit der linken Maustaste in den Bereich der JScrollPane eingebettet werden, der im WindowBuilder Editor als Viewport angezeigt wird.

Es entsteht auf diese Weise ein Quellcode entsprechend folgendem Abschnitt aus Beispiel jpr13190:

```
package bsp13190;

import javax.swing.JScrollPane;
import javax.swing.JTextArea;
...

public class MainPanel extends JPanel
{
  ...

  private JScrollPane spTextBereich;
  private JTextArea taTextBereich;

  ...

  private JScrollPane getSpTextBereich()
  {
    if (spTextBereich == null)
```

```
    {
      spTextBereich = new JScrollPane();
      spTextBereich.setBounds(new Rectangle(10, 10, 280, 150));
      spTextBereich.setViewportView(getTaTextBereich());
    }
    return spTextBereich;
  }

  protected JTextArea getTaTextBereich()
  {
    if (taTextBereich == null)
    {
      taTextBereich = new JTextArea();
// manuell eingefügt, damit die JTextArea gefüllt wird
      Verarbeitung.initialisiereAnzeige(getMainPanel());
    }
    return taTextBereich;
  }
}
```

Grundsätzlich entspricht dieser Quellcode dem in Kapitel 13.4.1 erklärten Prinzip. In die get-Methode zur JTextArea wurde anschließend jedoch manuell eine verarbeitende Methode eingefügt, die die JTextArea mit einem Text füllt. Auf diese Methode wird jetzt eingegangen:

13.4.5.1.2 Lesender und verändernder Zugriff auf den in der JTextArea angezeigten Text

Alle in diesem Kapitel 13.4.5 erklärten Komponenten können problemlos ohne ein vom Programmierer separat codiertes Modell verwendet werden. Wenn kein separat codiertes Modell bereitgestellt wird, werden die im JDK enthaltenen Standard-Modelle verwendet, die bereits mit den hier vorgestellten Komponenten verbunden sind. Die Funktionalität dieser Standard-Modelle ist mehr als ausreichend, so dass der Aufwand der Codierung von individuellen Modellen entfallen kann:

- Gemäß dem Java Delegate-Modell geschieht der lesende Zugriff auf die Anzeige in der GUI über Methoden der Komponente. Dies ist auch für die in Kapitel 13.4.5 vorgestellten Komponenten der Fall.
- Gemäß dem Java Delegate-Modell geschieht der verändernde Zugriff auf die Daten über das Modell. Für die Komponenten dieses Kapitels wird jedoch kein Modell bereitgestellt. Vielmehr gibt es für diese Komponenten Methoden, die "thread-sicher" sind und eine direkte Veränderung der angezeigten Daten durch Zugriff auf die Komponente ermöglichen.

Man erkennt die "thread-sichere" Methoden daran, dass in der JavaTM 2 Platform Standard Edition API Specification der jeweiligen Methode folgender Satz vorhanden ist:

This method is thread safe, although most Swing methods are not.

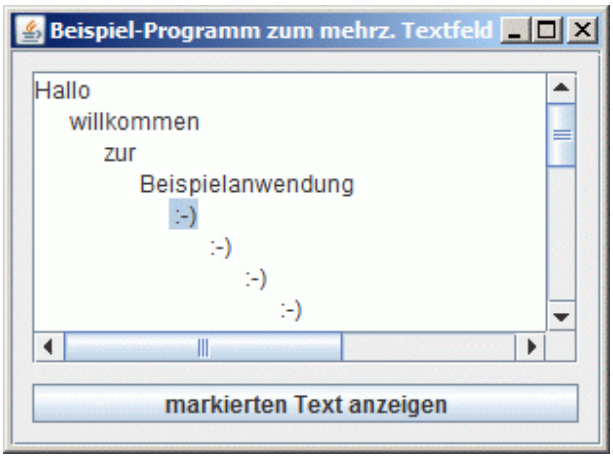
Nur in diesem Fall ist es erlaubt, direkt verändernd auf eine Komponente der GUI zuzugreifen. Für die in Kapitel 14.3.5 vorgestellten Methoden zur Veränderung der Text-Anzeige in der jeweiligen Komponente ist dies jedoch sichergestellt.

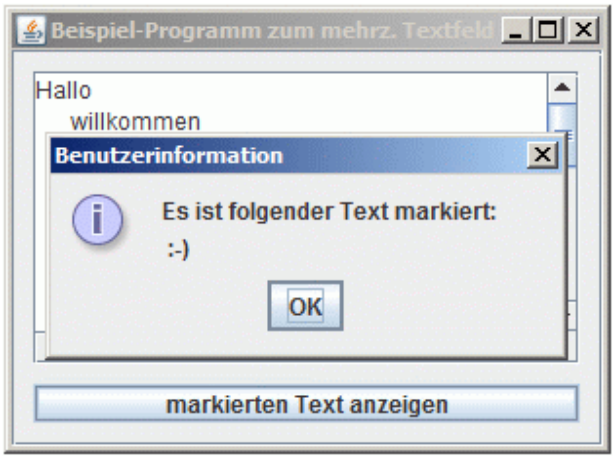
In Beispiel jpr13190 gibt es folgende beiden Methoden mit Zugriff auf den in der JTextArea angezeigten Text:

```
package bsp13190;

...

public class Verarbeitung
{
  ...

  public static void initialisiereAnzeige(MainPanel mp)
  {
    ...
    String text = ...
    ...
    mp.getTaTextBereich().setText(text);
  }

  public static void zeigeSelektiertenText(MainPanel mp)
  {
    String markierung = null;
    markierung = mp.getTaTextBereich().getSelectedText();
    ...
  }
}
```

- Mit der von der Klasse javax.swing.text.JTextComponent geerbten Methode
  `setText(String)`
  der Klasse JTextArea kann der in der JTextArea anzuzeigende Text gelöscht und neu gesetzt werden.
- Mit der von der Klasse javax.swing.text.JTextComponent geerbten Methode
  `getSelectedText()`
  der Klasse JTextArea kann der in der JTextArea markierte Text gelesen werden. Diese Methode ist nicht thread-sicher, was aber auch nicht notwendig ist, da sie einen lesenden Vorgang durchführt.

Die Verwendung der beiden Methoden in den verarbeitenden Methoden von Beispiel jpr13190 ermöglicht damit ein Füllen der JTextArea mit einem Text bei der ersten Anzeige des Programms (s.o.) sowie das Lesen des in der GUI vom Anwender markierten Textes:





Hinweis: Die Klasse JTextArea hat eine umfangreiche API, die viele weitere nützliche Methoden bereitstellt. Viele dieser Methoden sind von übergeordneten Klassen geerbt und sind daher auch für andere Komponenten dieses Kapitels verfügbar.

13.4.5.2 JPasswordField

Das JPasswordField wird für die verdeckte Eingabe - also zum Beispiel für die Eingabe von Kennwörtern - eingesetzt.

13.4.5.2.1 Codierung einer JPasswordField-Komponente

Eine Komponente vom Typ JPasswordField wird in gleicher Weise generiert wie ein JTextField. Hierzu wähle man in der Palette des WindowBuilder Editors die Komponente JPasswordField aus und bette diese Komponente wie gewohnt in das Panel ein. Es wird empfohlen, den Namen des ensprechenden Objektes mit dem Text **pf** beginnen zu lassen. Es wird dann ein Quellcode wie in folgendem Beispiel (Beispiel jpr13200, Panel de.eikelberg.guitools.panels.LogonPanel) erzeugt:

```
package de.eikelberg.guitools.panels;

import javax.swing.JPasswordField;
...

public class LogonPanel extends JPanel
{
  ...

  private JPasswordField pfKennwort;

  ...

  protected JPasswordField getPfKennwort()
  {
    if (pfKennwort == null)
    {
      pfKennwort = new JPasswordField();
      pfKennwort.setBounds(new Rectangle(110, 40, 160, 20));
    }
    return pfKennwort;
  }
  ...
}
```

13.4.5.2.2 Lesender und verändernder Zugriff auf ein JPasswordField

Für den Zugriff auf ein JPasswordField können neben vielen weiteren Methoden die folgenden Methoden verwendet werden:

- `public void setText(String t)` - von der Klasse javax.swing.text.JTextComponent geerbte thread-sichere Methode (vgl. Kap. 13.4.5.1.2 zur JTextArea), mit der der im JPasswordField eingetragene Text geändert werden kann. Da diese Methode threadsicher ist, kann der im JPasswordField eingetragene Text mit dieser Methode zur Laufzeit geändert werden.
- `public char[] getPassword()` - gibt den eingegebenen Text in Form eines char-Arrays zurück. Entsprechend muss der gelesene Text gegebenenfalls in das String-Format umgewandelt werden. Diese Methode ist nicht thread-sicher, was aber auch nicht notwendig ist da ein lesender Vorgang vorliegt.
- `public void setEditable(boolean b)` - von der Klasse javax.swing.text.JTextComponent geerbte Methode, mit der ein JPasswordField editierbar oder nicht editierbar gesetzt werden kann.
- `public boolean isEditable()` - von der Klasse javax.swing.text.JTextComponent geerbte Methode, mit der abgefragt werden kann, ob ein JPasswordField editierbar oder nicht editierbar gesetzt ist.
- `public void setBackground(Color bg)` - von der Klasse javax.swing.text.JTextComponent geerbte Methode, mit der die Hintergrundfarbe der Komponente gesetzt werden kann.

Hinweis: Verwenden Sie nicht die Methode getText(), denn diese ist deprecated (nicht mehr empfohlen und zukünfig möglicherweise nicht mehr vorhanden)!

Das Lesen des eingegebenen Kennwortes geschieht somit mit der Methode getPassword(). Anschließend wird von den meisten Programmierern eine Umwandlung des eingelesenen Kennwortes in den Datentyp String vorgenommen wie zum Beispiel auch in der folgenden Methode Beispiel jpr13200, Klasse de.eikelberg.guitools.panels.LogonVerarbeitung:

```
protected static void logOnOff(LogonPanel lp)
{
  ...
  char[] pwa = lp.getPfKennwort().getPassword();
  String pws = String.valueOf(pwa).trim();
  ..
}
```

Das Setzen der Hintergrundfarbe einer Komponente kann mit Hilfe von Standard-Java-Farben oder mit RGB-Werten erfolgen wie in folgenden Anweisungen aus Beispiel jpr13200, Klasse de.eikelberg.guitools.panels.LogonPanel:

```
protected static void logOnOff(LogonPanel lp)
{
  ...
  lp.getPfKennwort().setBackground(Color.WHITE);
  ...
  lp.getPfKennwort().setBackground(new Color(232, 232, 232));
  ..
}
```

Die erste Anweisung setzt eine weiße Hintergrundfarbe (codiert für den Fall, dass das JPasswordField editierbar gesetzt ist) und die zweite Anweisung setzt eine hellgraue Hintergrundfarbe (codiert für den Fall, dass das JPasswordField nicht editierbar gesetzt ist). Die drei angegebenen Werte sind die RGB-Werte der einzustellenden Farbe.

13.4.5.2.3 Beispiel de.eikelberg.guitools.panels.LogonPanel

Die Verwendung einer JPasswordField-Komponente wird im Beispiel jpr13200 gezeigt.

- Die Anzeige der Komponente geschieht in einem Panel **de.eikelberg.guitools.panels.LogonPanel** mit einer GUI für die Kennworteingabe und
- die zugehörige Verarbeitung geschieht in statischen Methoden der Klasse **de.eikelberg.guitools.panels.LogonVerarbeitung**.

Hinweis: Das Panel LogonPanel mit der zugehörigen verarbeitenden Klasse LogonVerarbeitung wird im Beispiel jpr13200 sowie in nachfolgenden Beispielen zu Kapitel 13.5 verwendet, weshalb beide Klassen in ein Paket de.eikelberg.guitools.panels ausgelagert wurden.

- In Beispiel jpr13200 wird das LogonPanel in ein Panel **bsp13200.MainPanel** eingebettet.
- Die zum übergeordneten Panel bsp13200.MainPanel gehörende Verarbeitung ist in der Klasse **bsp13200.Verarbeitung** codiert.

Das Einbetten des Panels LogonPanel in das Panel bsp13200.MainPanel erfolgte auf die in Kapitel 13.4.1.3 erklärte Art und Weise. Das untergeordnete Panel **de.eikelberg.guitools.panels.LogonPanel** enthält die Komponente JPasswordField sowie Instanzvariablen, get- und set-Methoden zur Verwaltung von Benutzername, Kennwort und Logon-Status:

```
package de.eikelberg.guitools.panels;

import ...
```

```
public class LogonPanel extends JPanel
{

  ...

  private JTextField tfBenutzername;
  private JPasswordField pfKennwort;
  private JButton butLogon;

  ...

// Manuell eingefügte Variablen, get- und set-Methoden
  private boolean logon = false;

  public boolean isLogon()
  {
    return logon;
  }

  protected void setLogon(boolean logon)
  {
    this.logon = logon;
  }

  private String benutzername = null;

  public String getBenutzername()
  {
    return benutzername;
  }

  protected void setBenutzername(String benutzername)
  {
    this.benutzername = benutzername;
  }

  private String kennwort = null;

  public String getKennwort()
  {
    return kennwort;
  }

  protected void setKennwort(String kennwort)
  {
    this.kennwort = kennwort;
  }

  protected LogonPanel getLogonPanel()
  {
    return this;
  }

// vom WindowBuilder Editor generierter Quellcode
  /**
   * Create the panel.
   */
  public LogonPanel()
  {
    initialize();
  }

  ...

  protected JTextField getTfBenutzername()
  {
    ...
  }

  protected JPasswordField getPfKennwort()
  {
    if (pfKennwort == null)
```

```
    {
      pfKennwort = new JPasswordField();
      pfKennwort.setBounds(new Rectangle(110, 40, 160, 20));
    }
    return pfKennwort;
  }

  protected JButton getButLogon()
  {
    ...
  }
}
```

Insgesamt entsteht folgende GUI - hier angezeigt vom WindowBuilder Editor:

Die Verarbeitung zu diesem Panel erfolgt nach Klick auf den Button "logon", dessen Beschriftung während des Programmlaufs zwischen "logon" und "logoff" wechselt. Die Verarbeitung ist codiert in der statischen Methode logOnOff(LogonPanel lp) der Klasse **de.eikelberg.guitools.panels.LogonVerarbeitung**. Diese Methode enthält die oben angeführten Methodenaufrufe für den Zugriff auf das JPasswordField:

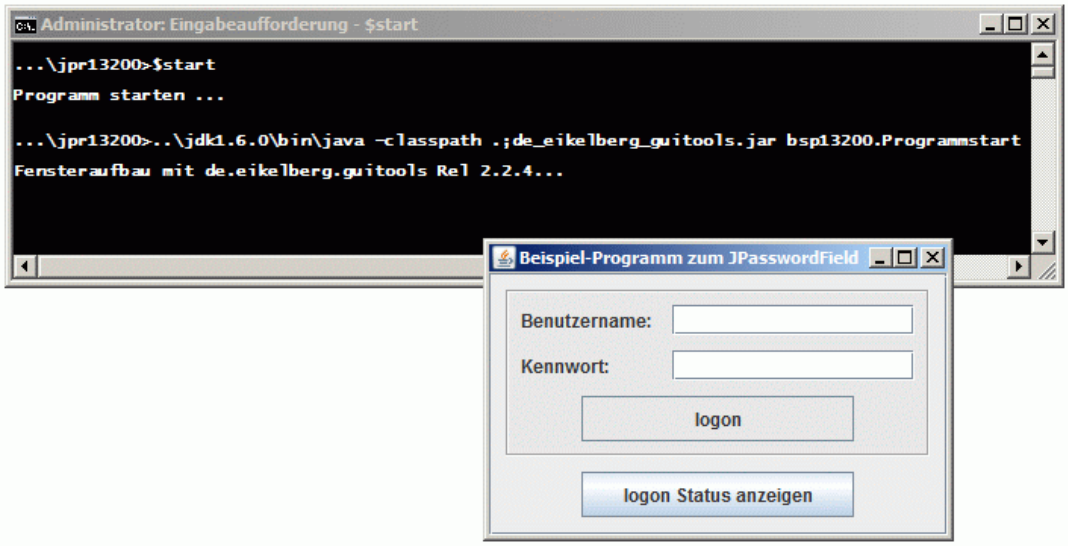
```
package de.eikelberg.guitools.panels;

import ...

public class LogonVerarbeitung
{
  ...

  protected static void logOnOff(LogonPanel lp)
  {
    boolean aktion = false;
// Beschriftung des Buttons ändern, Kennwort und Benutzername setzen
    if (lp.isLogon())
    {
      lp.setBenutzername(null);
      lp.getTfBenutzername().setText("");
      lp.getTfBenutzername().setBackground(Color.WHITE);
      lp.setKennwort(null);
      lp.getPfKennwort().setText("");
      lp.getPfKennwort().setBackground(Color.WHITE);
      lp.getButLogon().setText("logon");
      aktion = true;
    }
    else if(!lp.getTfBenutzername().getText().trim().equals(""))
    {
      lp.setBenutzername(lp.getTfBenutzername().getText().trim());
      lp.getTfBenutzername().setBackground(new Color(232, 232, 232));
      char[] pwa = lp.getPfKennwort().getPassword();
      String pws = String.valueOf(pwa).trim();
      lp.setKennwort(pws);
// Anzeige im Passwordfeld überschreiben, damit die Länge des
// Kennwortes nicht mehr erkennbar ist
      lp.getPfKennwort().setText("************************************");
      System.out.println("\nLogon von Benutzer mit Benutzername: "+lp.getBenutzername());
      lp.getPfKennwort().setBackground(new Color(232, 232, 232));
      lp.getButLogon().setText("logoff");
      aktion = true;
    }
    else
    {
      JOptionPane.showMessageDialog(
        lp,
```

```
        "Fehler - es wurde beim Logon\n"+
        "kein Benutzerkonto eingegeben!",
        "Fehler beim Logon",
        JOptionPane.ERROR_MESSAGE
      );
      aktion = false;
    }
    if (aktion)
    {
// Status editierbar umstellen
    lp.getTfBenutzername().setEditable(!lp.getTfBenutzername().isEditable());
    lp.getPfKennwort().setEditable(!lp.getPfKennwort().isEditable());
// Status aktiviert umstellen
    lp.getTfBenutzername().setEnabled(!lp.getTfBenutzername().isEnabled());
    lp.getPfKennwort().setEnabled(!lp.getPfKennwort().isEnabled());
// Logon-Status umstellen
    lp.setLogon(!lp.isLogon());
    }
  }
}
```

Im Panel **bsp13200.MainPanel** wurde das Panel LogonPanel wie in Kapitel 13.4.1.3 beschrieben mit dem WindowBuilder Editor eingefügt, wodurch im Panel bsp13200.MainPanel der folgende Quellcode enstand:

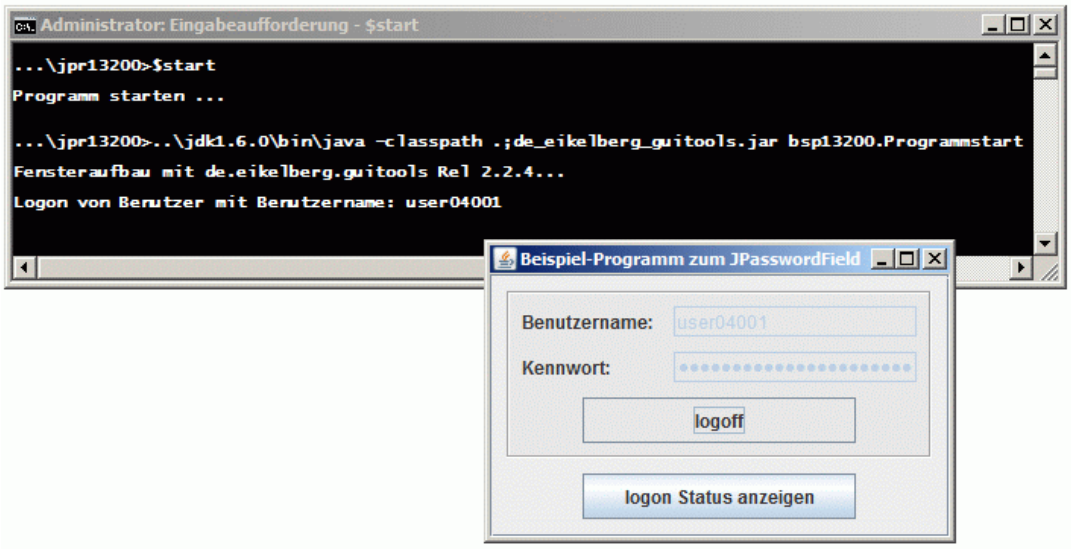
```
package bsp13200;

import ...

import de.eikelberg.guitools.panels.LogonPanel;

public class MainPanel extends JPanel
{
  ...

  private LogonPanel logonPanel;

  ...

  private void initialize()
  {
    ...
    add(getLogonPanel());
    ...
  }

  protected LogonPanel getLogonPanel()
  {
    if (logonPanel == null)
    {
      logonPanel = new LogonPanel();
      logonPanel.setBounds(10, 10, 280, 110);
    }
    return logonPanel;
  }
  ...
}
```

Nach Start des Beispielprogramms im Projekt jpr13200 wird aufgrund von Anweisungen zur Einstellung der Hintergrundfarbe im LogonPanel das in das MainPanel eingebettete LogonPanel mit einem etwas grauen Farbton (RGB-Farbe 232,232,232) als standardmäßig eingestellt angezeigt:



Nach Eingabe eines Benutzernamens und eines Kennwortes sowie Klick auf den Button logon

- wird die Beschriftung des Buttons von logon auf logoff umgestellt,
- die Eingabefelder für Benutzername und Kennwort werden für eine weitere Eingabe gesperrt und
- diese beiden Felder erhalten ebenfalls als Hintergrundfarbe die RGB-Farbe 232,232,232:



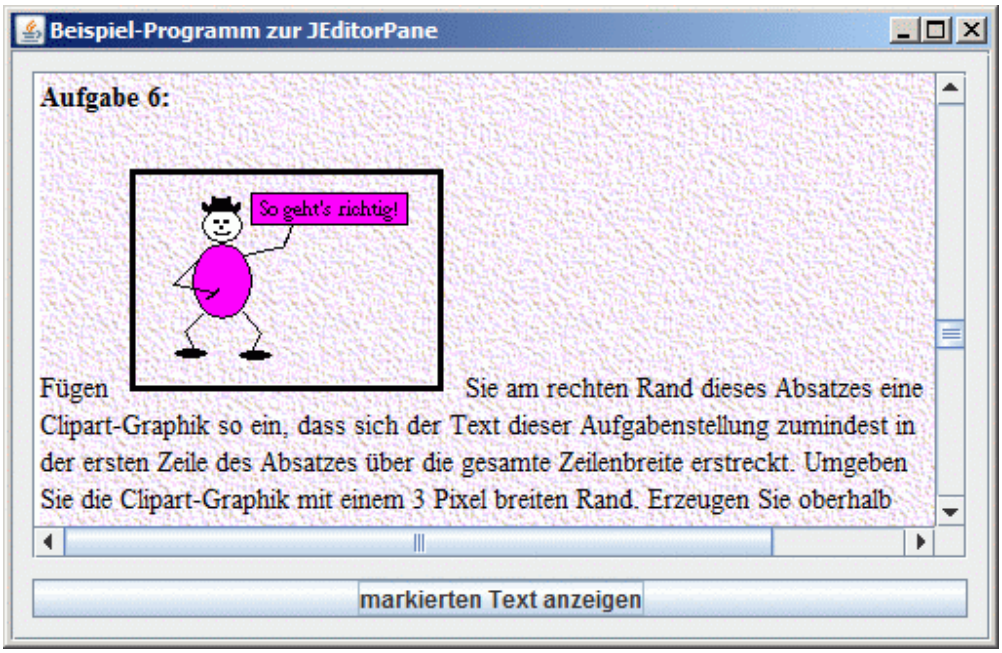
Das Logon-Panel wird in Kapitel 13.5 verwendet, um eine Benutzeranmeldung an einer Datenbank durchzuführen.

13.4.5.3 JEditorPane

Die Klasse JEditorPane ist wie die Klasse JTextArea von der Klasse javax.swing.text.JTextComponent abgeleitet, weshalb es deutliche Übereinstimmungen der Darstellung in diesem Kapitel mit der Darstellung in Kapitel 13.4.5.1 gibt. Im Unterschied zur JTextArea kann die JEditorPane jedoch neben Daten in reinem Text-Format auch andere Daten verarbeiten. Insgesamt können verarbeitet werden folgende Formate:

- **text/plain** - Daten in reinem Text-Format, wie sie z. bsp. von reinen Text-Editoren erzeugt werden.
- **text/html** - HTML-Daten (für das JDK 1.4.2 in Version HTML 3.2)
- **text/rtf** - RTF (rich text) Daten, wie sie zum Beispiel vom Windows-Programm WordPad erzeugt werden.

Beispiel jpr13210 zeigt eine GUI mit JEditorPane, in der eine Web-Seite mit Wallpaper und Images angezeigt wird:



13.4.5.3.1 Codierung einer JEditorPane-Komponente

Eine JEditorPane-Komponente wird wie eine JTextArea-Komponente im WindowBuilder Editor angelegt. Sie sollte ebenfalls in eine JScrollPane eingebettet werden:

- Zunächst sollte die JScrollPane in die GUI eingebettet werden.
- Im zweiten Schritt sollte dann die JEditorPane aus der Palette des WindowBuilder Editors, Abschnitt Components ausgewählt und durch Klick mit der linken Maustaste in dem Bereich der JScrollPane eingebettet werden, die im WindowBuilder Editor als Viewport markiert ist.

Es entsteht auf diese Weise ein Quellcode entsprechend folgendem Abschnitt aus Beispiel jpr13210:

```
package bsp13210;

import javax.swing.JEditorPane;
...

public class MainPanel extends JPanel
{
  ...

  private JScrollPane spTextBereich;
  private JEditorPane epHtmlAnzeige;

  private JScrollPane getSpTextBereich()
  {
    if (spTextBereich == null)
    {
      spTextBereich = new JScrollPane();
      spTextBereich.setBounds(10, 10, 480, 250);
      spTextBereich.setViewportView(getEpHtmlAnzeige());
    }
    return spTextBereich;
  }

  protected JEditorPane getEpHtmlAnzeige()
  {
    if (epHtmlAnzeige == null)
    {
      epHtmlAnzeige = new JEditorPane();
      epHtmlAnzeige.addMouseListener(new MouseAdapter()
      {
        @Override
        public void mouseClicked(MouseEvent e)
        {
```

```
          if (e.getClickCount() >= 2)
            Verarbeitung.zeigeSelektiertenText(getMainPanel());
        }
      });
      epHtmlAnzeige.setEditable(false);

// manuell eingefügt, um die Webseite beim Programmstart zu laden
      Verarbeitung.initialisiereAnzeige(getMainPanel());
    }
    return epHtmlAnzeige;
  }
}
```

Man sieht, dass der JEditorPane ein MouseListener hinzugefügt wurde, der bei Doppelklick eine Methode aufruft, die den beim Doppelklick markierten Text - in der Regel den Bereich den man markiert hat bzw. das Wort auf das man doppelt geklickt hat - in einem Meldefenster anzeigen soll. Ferner wurde manuell eine Methodenaufruf hinzugefügt, der die Anzeige der Web-Seite bei der erstmaligen Anzeige der JEditorPane durchführt. Beide Methoden zeigen den lesenden bzw. schreibenden Zugriff auf die JEditorPane-Komponente:

13.4.5.3.2 Lesender und verändernder Zugriff auf den in der JEditorPane angezeigten Text

In den beiden oben genannten statischen Methoden der Klasse bsp13210.Verarbeitung geschieht der Zugriff auf die JEditorPane wie folgt:

```
package bsp13210;

import java.net.URL;
import java.net.MalformedURLException;
import java.io.IOException;
...

public class Verarbeitung
{
  ...

  public static void initialisiereAnzeige(MainPanel mp)
  {
//  Zugriff auf Webseite, die in der JEditorPane angezeigt werden soll.
     URL url = null;
     try
     {
       url = new URL("file:webseite//aufg0002.html");
     }
     catch (MalformedURLException mue)
     {
       ...
     }
//  Wenn Webseite gefunden wurde wird diese angezeigt
     if (!(url==null))
     {
       try
       {
         mp.getEpHtmlAnzeige().setPage(url);
       }
       catch (IOException e)
       {
         ...
       }
     }
  }

  public static void zeigeSelektiertenText(MainPanel mp)
  {
    String markierung = null;
    markierung = mp.getEpHtmlAnzeige().getSelectedText();
    ...
  }
}
```

Beim Zugriff werden aus der umfangreichen API der Klasse JEditorPane folgende beiden Methoden verwendet:

- Mit der Methode

```
public void setPage(URL) throws IOException
```

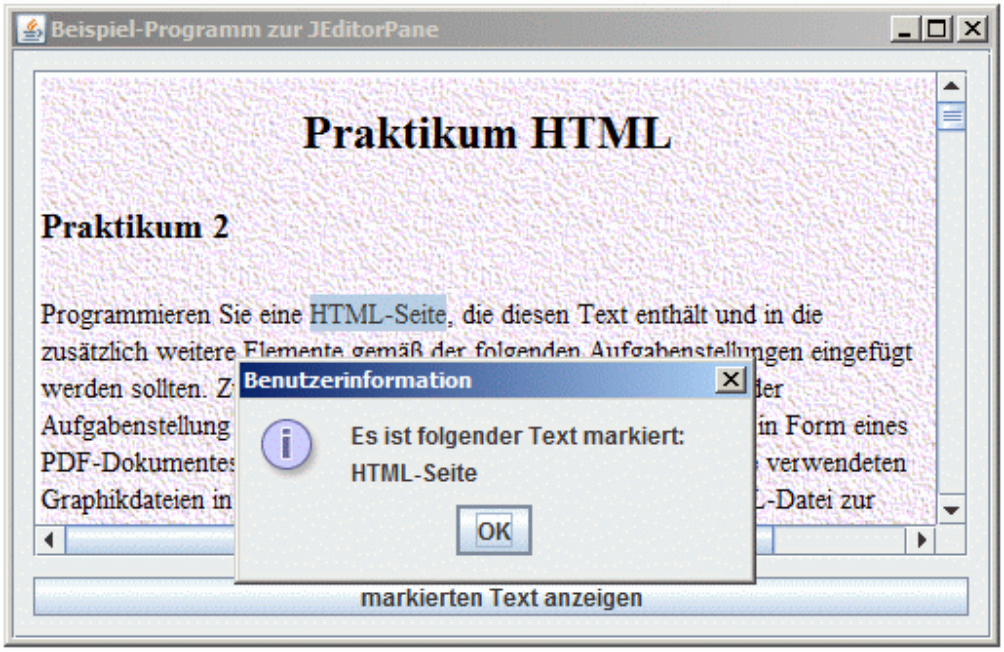
wird die Web-Seite, auf die das als Parameter angegebenen URL-Objekt zeigt, in der JEditorPane angezeigt. *Diese Methode ist nicht thread-sicher und darf daher nur in der get-Methode zur JEditorPane (und damit vor Anzeige der JEditorPane) oder in einer Methode codiert werden, die im Rahmen der Ereignisbehandlung aufgerufen wird!*

- Mit der von der Klasse javax.swing.text.JTextComponent geerbten Methode

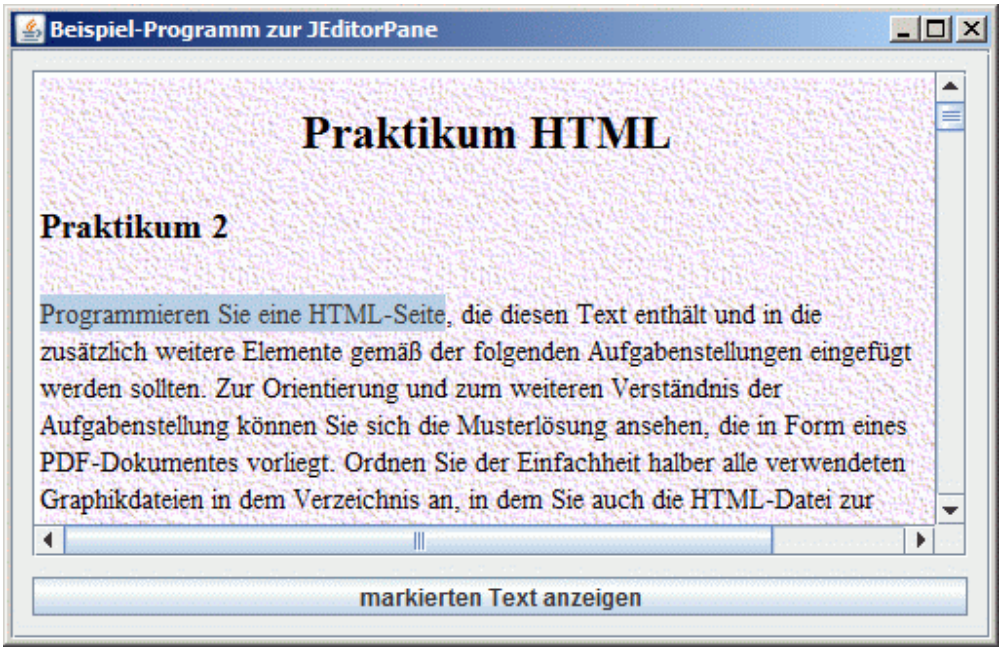
```
public String getSelectedText()
```

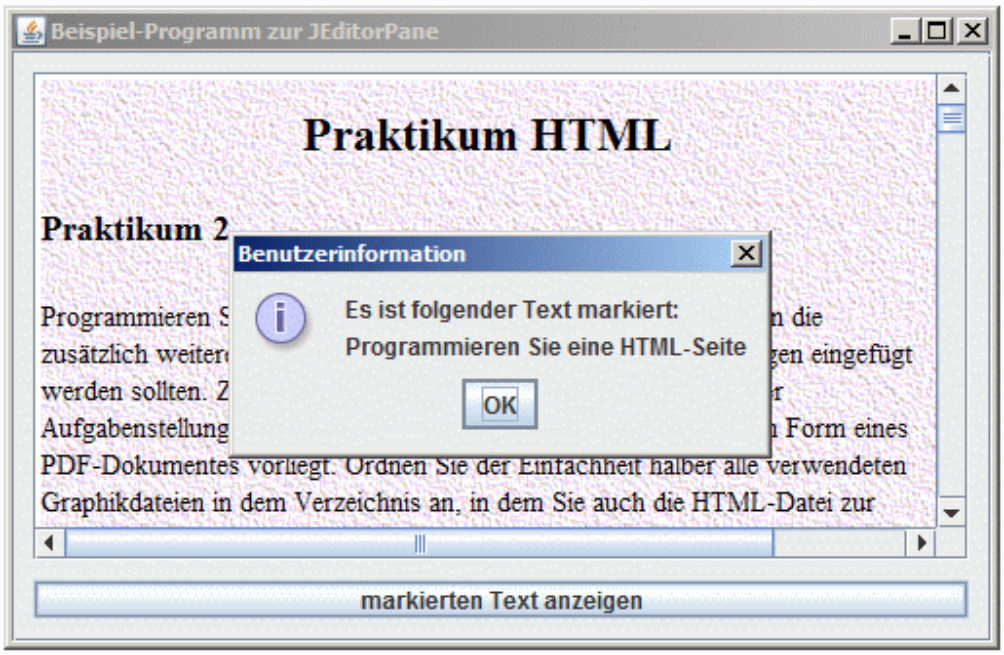
der Klasse JTextArea kann der in der JTextArea markierte Text gelesen werden. Diese Methode ist nicht thread-sicher, was aber auch nicht notwendig ist, da sie einen lesenden Vorgang durchführt.

Die Anwendung der Methode getSelectedText() ermöglicht es wieder wie bei der Komponente JTextArea, bei Doppelklick im Bereich der Web-Seite den dabei vom Benutzer in der JEditorPane beim Doppelklick markierten Text - dies ist in der Regel das Wort auf das geklickt wird - zu lesen:



Bei einem einfachen Klick auf den Button "markierten Text anzeigen" wird der vom Benutzer zuvor markierte Text angezeigt:



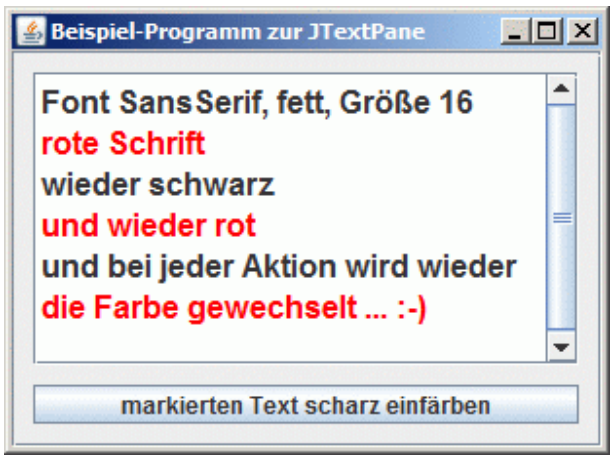


13.4.5.4 JTextPane

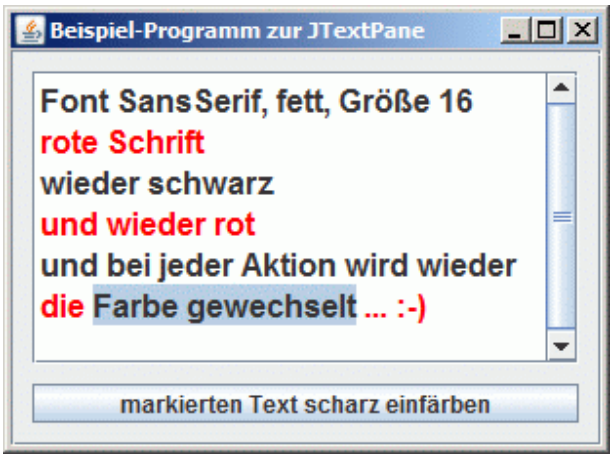
Eine JTextPane ist eine von der Klasse JEditorPane abgeleitete Klasse, die somit die Funktionalität der JEditorPane erweitert. Zusätzlich zum Funktionsumfang der JEditorPane bietet eine JTextPane folgende Möglichkeiten:

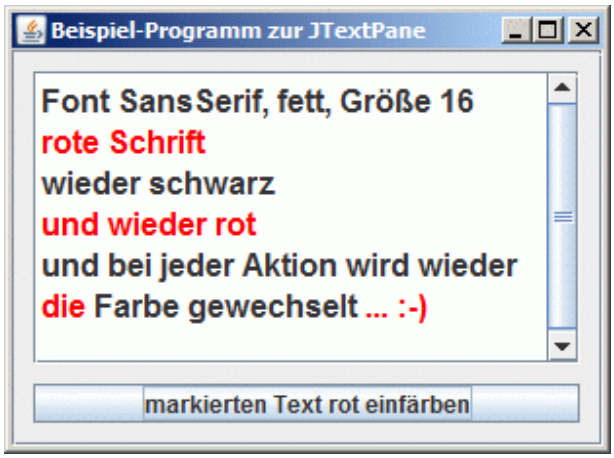
- In einer JTextPane können Absätze gebildet werden, die jeweils individuell formatiert sind.

Beispiel jpr13220 zeigt die Anzeige von unterschiedlich formatiertem Text in einer JTextPane.



Zusätzlich ist es möglich, Text zu markieren und für diesen markierten Text die Formatierung zu ändern (hier Wechsel zwischen roter und schwarzer Schrift nach Klick auf den Button "markierten Text ... einfärben"):





13.4.5.4.1 Codierung einer JTextPane-Komponente

Eine JTextPane-Komponente wird wie eine JEditorPane-Komponente im WindowBuilder Editor angelegt. Sie sollte ebenfalls in eine JScrollPane eingebettet werden:

- Zunächst sollte eine JScrollPane in die GUI eingebettet werden.
- Im zweiten Schritt sollte dann die JTextPane aus der Palette des WindowBuilder Editors, Abschnitt Components ausgewählt und durch Klick mit der linken Maustaste in dem Bereich der JScrollPane eingebettet werden, der im WindowBuilder Editor als Viewport angezeigt wird.

Es entsteht auf diese Weise ein Quellcode entsprechend folgendem Abschnitt aus Beispiel jpr13220:

```
package bsp13220;

import javax.swing.JTextPane;
...

public class MainPanel extends JPanel
{
  ...

  private JScrollPane spTextBereich;
  private JTextPane tpTextAnzeige;

  private JScrollPane getSpTextBereich()
  {
    if (spTextBereich == null)
    {
      spTextBereich = new JScrollPane();
      spTextBereich.setBounds(new Rectangle(10, 10, 280, 150));
      spTextBereich.setViewportView(getTpTextAnzeige());
    }
    return spTextBereich;
  }

  protected JTextPane getTpTextAnzeige()
  {
    if (tpTextAnzeige == null)
    {
      tpTextAnzeige = new JTextPane();
      tpTextAnzeige.setEditable(false);

// manuell eingefügt, um den anzuzeigenden Text beim Programmstart zu laden
      Verarbeitung.initialisiereAnzeige(getMainPanel());

    }
    return tpTextAnzeige;
  }
}
```

Grundsätzlich ist es möglich, für eine JTextPane Funktionalität wie für eine JEditorPane zu codieren, da die Klasse JTextPane von der Klasse JEditorPane abgeleitet ist. Im vorliegenden Fall wurde der JEditorPane jedoch keine Ereignisbehandlung zugefügt. Vielmehr wurde lediglich manuell eine Methodenaufruf hinzugefügt, der die Anzeige eines formatierten Textes bei der erstmaligen Anzeige der JEditorPane durchführt.

Nach Klick auf den Button "markierten Text ... einfärben", wird der markierte Text mit Hilfe einer weiteren Methode der Klasse Verarbeitung je nach Beschriftung des Buttons scharz bzw. rot eingefärbt. Beide Methoden zeigen den lesenden bzw. verändernden Zugriff auf die JTextPane-Komponente:

13.4.5.4.2 Lesender und verändernder Zugriff auf den in der JTextPane angezeigten Text

Da die JTextPane von der JEditorPane abgeleitet ist, können die bereits in Kapitel 13.4.5.3.2 genannten Methoden der JEditorPane verwendet werden sowie natürlich auch alle von der Klasse JTextArea an die Klasse JEditorPane vererbten Methoden. Damit verfügt die Klasse JTextPane über eine sehr umfanreiche API. In Beispiel jpr13220 werden aus dieser API die folgenden Methoden verwendet:

- Mit der Methode
  `public void setCharacterAttributes(AttributeSet attr, boolean replace)`
  legt die neue Formatierung eines Textes fest und entfernt die bisherige Formatierung, wenn als zweiter Parameter der Wert true angegeben wird. Ist zum Zeitpunkt des Methodenaufrufs in der JTextPane ein Text markiert, so wird die neue Formatierung auf diesen Text angewendet. Ist dagegen kein Text markiert, so gilt die neue Formatierung für anschließend neu einzufügenden Text. Die Methode setCharacterAttributes(AttributeSet,boolean) ist thread-sicher und kann daher zur Veränderung des bereits angezeigten Textes angewendet werden.
- Mit der von der Klasse javax.swing.text.JTextComponent geerbten Methode
  `public String getSelectedText()`
  der Klasse JTextPane kann der in der JTextPane markierte Text gelesen werden. Diese Methode ist nicht thread-sicher, was aber auch nicht notwendig ist, da sie einen lesenden Vorgang durchführt.
- Mit der von der Klasse javax.swing.text.JTextComponent geerbten Methode
  `public void setSelectionStart(int selectionEnd)`
  wird der Beginn eines in der GUI markierten Bereichs festgelegt.
- Mit der von der Klasse javax.swing.text.JTextComponent geerbten Methode
  `public void setSelectionEnd(int selectionEnd)`
  wird das Ende eines in der GUI markierten Bereichs festgelegt. Ist für den markierten Bereich der Beginn gleich dem Ende, so gilt der Bereich als nicht markiert, d.h. es gibt dann keine Markierung.

Beim Aufruf der Methode setCharacterAttributes(AttributeSet,boolean) muss ein Objekt einer Klasse angegeben werden, die das Interface java.swing.text.AttributeSet implementiert. Wie man ein solches Objekt erhält, zeigt der Quellcode in der Klasse bsp13220.Verarbeitung:

```
package bsp13220;

import javax.swing.JTextPane;
import javax.swing.text.AbstractDocument;
import javax.swing.text.StyledDocument;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.StyleConstants;
import java.awt.Color;
...

public class Verarbeitung
{
  ...

  public static void initialisiereAnzeige(MainPanel mp)
  {
    JTextPane jtp = mp.getTpTextAnzeige();
    StyledDocument styledDoc = jtp.getStyledDocument();
    AbstractDocument myDoc = null;
    if (styledDoc instanceof AbstractDocument)
    {
      myDoc = (AbstractDocument)styledDoc;
    }
    else
    {
      ...
    }
    if (myDoc != null)
    {
// Text für die JTextPane codieren
      String[] text =
      {
          "Font SansSerif, fett, Größe 16",
          "rote Schrift",
          "wieder schwarz",
          "und wieder rot",
          "und bei jeder Aktion wird wieder",
```

```
          "die Farbe gewechselt ... :-)"
      };
// JTextPane füllen
      try
      {
        for (int i = 0; i < text.length; i ++)
        {
          myDoc.insertString(
            myDoc.getLength(),
            text[i] + "\n",
            getAttributeSet()
          );
        }
      }
      catch (BadLocationException e)
      {
        ...
      }

    }
  }

  private static SimpleAttributeSet[] textStilArray =  null;
  private static boolean textStilStandard = true;
  private static SimpleAttributeSet getAttributeSet()
  {
//  sicherstellen, dass ein Array für die Formate existiert
    if (textStilArray == null)
    {
      textStilArray = new SimpleAttributeSet[2];

      textStilArray[0] = new SimpleAttributeSet();
      StyleConstants.setFontFamily(textStilArray[0], "SansSerif");
      StyleConstants.setBold(textStilArray[0], true);
      StyleConstants.setFontSize(textStilArray[0], 16);

      textStilArray[1] = new SimpleAttributeSet(textStilArray[0]);
      StyleConstants.setForeground(textStilArray[1], Color.red);
    }
// Textstil für die Rückgabe festlegen
    SimpleAttributeSet textStil = null;
    if (textStilStandard)
      textStil = textStilArray[0];
    else
      textStil = textStilArray[1];
// Textstil umschalten
    textStilStandard = !textStilStandard;
    return textStil;
  }

  public static void zeigeSelektiertenText(MainPanel mp)
  {
    String markierung = null;
    markierung = mp.getTpTextAnzeige().getSelectedText();
    if (markierung == null)
      ...
    else
    {
// Wenn es einen selektierten Bereich gibt, wirkt die nachfolgende Methode
// nur auf diesen Bereich!
      mp.getTpTextAnzeige().setCharacterAttributes(getAttributeSet(), true);
// Beschriftung des Buttons wechseln
      if (textStilStandard)
        mp.getButAnzeige().setText("markierten Text scharz einfärben");
      else
        mp.getButAnzeige().setText("markierten Text rot einfärben");
// Selektion in der GUI aufheben
      mp.getTpTextAnzeige().setSelectionStart(0);
      mp.getTpTextAnzeige().setSelectionEnd(0);
    }
  }
}
```

Man sieht, dass der schreibende Zugriff in die JTextPane über das Objekt der Klasse javax.swing.text.AbstractDocument erfolgt, das den in der JTextPane angezeigten Text verwaltet. Mit der Methode

```
public void insertString(int offs, String str, AttributeSet attrset) throws BadLocationException
```

dieser Klasse wird der angegebene String str nach Offset offs unter Verwendung der im AttributeSet attrset angegebenen Formatierung eingefügt. Das hierbei benötigte AttributeSet wird im Beispiel von der obigen Methode getAttributeSet() bereitgestellt, in der bei jedem Aufruf der Methode jeweils zwischen scharzer und roter Schrift umgeschaltet wird.

Die Methode getAttributeSet() wird auch beim Aufruf der Methode setCharacterAttributes(AttributeSet,boolean) verwendet, um das bei der Ausführung der Methode anzuwendende AttributeSet zu erzeugen.

Hinweis: Beispiel jpr13221 zeigt die Implementierung eines Programms mit der Funktionalität von Beispiel jpr13210, jedoch jetzt mit der Klasse JTextPane.

13.5 Programme mit grafischer Benutzeroberfläche und Datenbankzugriff

In diesem Kapitel werden die in Kap. 13 erlernten Kenntnisse zur Programmierung grafischer Benutzeroberflächen mit dem WindowBuilder Editor und die in Kap. 14 dargelegten Techniken für den Datenbankzugriff zusammengeführt. Voraussetzung für die erfolgreiche Bearbeitung von Kapitel 13.5 sind

- **Kap. 13.1 - 13.3, 13.4.4.1 (JTable) und 13.4.5.2 (JPasswordField)**
- **Kap. 14.1 - 14.5**

13.5.1 Aufbau und Abbau einer Datenbankverbindung mit dem Logon-Panel

Hinweis: Ein Datenbankzugriff wird oft so codiert, dass für jeden Verarbeitungsschritt

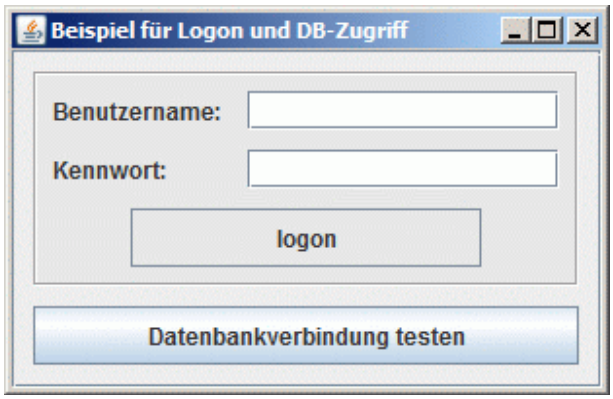
- eine Datenbankverbindung aufgebaut wird,
- die jeweilige Verarbeitung durchgeführt wird und
- danach direkt die Datenbankverbindung wieder abgebaut wird.

Abweichend davon kann die Datenbankverbindung gehalten werden, wenn mehrere Datenbankzugriffe vom Programm dicht aufeinanderfolgendend abgesetzt werden und wenn sichergestellt werden kann, dass nach dem letzten Datenbankzugriff die Datenbankverbindung wieder geschlossen wird.

Entsprechend sollte ein Logon-Vorgang in einem Programm wie folgt codiert werden:

- Der Logon-Vorgang stellt dem Programm Anmeldeinformationen in Form eines Benutzernamens mit zugehörigem Kennwort bereit, die für die Datenbankzugriffe verwendet werden können. Gegebenenfalls kann ein Verbindungsaufbau mit direkt nachfolgendem Verbindungsabbau erfolgen, um die Anmeldeinformatinen zu testen.
- Während des Programlaufs können die Anmeldedaten bei einem erneuten Verbindungsaufbau zum Datenbankserver aus dem Logon-Panel geholt werden.
- Der Logoff-Vorgang löscht die Anmeldeinformationen, so dass keine Datenbankzugriffe mehr möglich sind.

Das Beispielprogramm für den Zugriff auf eine Datenbank mit dem LogonPanel (jpr13310) illustriert diese Art der Verarbeitung. Es hat folgende GUI:

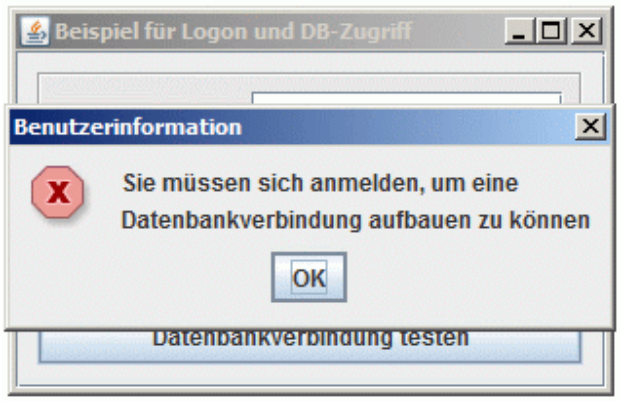


Durch Eingabe von Benutzername und Kennwort sowie Klick auf den Button "logon" wird das Programm in die Lage versetzt, auf die Datenbank zuzugreifen:

Nach Klick auf den Button "Datenbankverbindung testen" erfolgt ein Verbindungsaufbau. Falls dieser erfolgreich ist, wird eine entsprechende Meldung angezeigt und nach Bestätigung dieser Meldung die Datenbankverbindung wieder abgebaut:

Ist ein Verbindungsaufbau nicht möglich, erscheint eine entsprechende Fehlermeldung:

Falls kein Logon erfolgt ist, verfügt das Progamm nicht über geeignete Anmeldeinformationen und ein Datenbankzugriff ist nicht möglich:



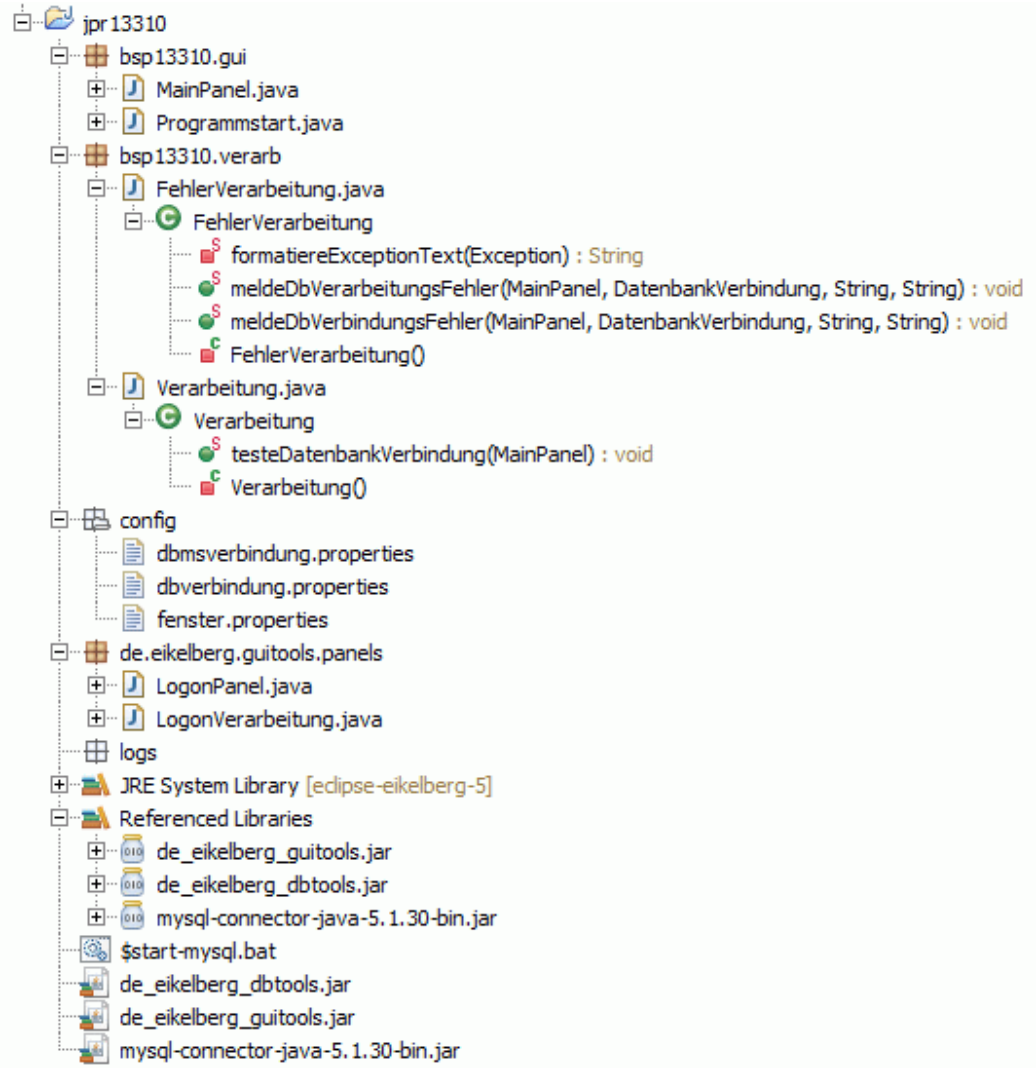
Das Beispiel jpr13310 für den Zugriff auf eine Datenbank mit dem LogonPanel ist eine Erweiterung des Beispiels zur JPasswordField (mit LogonPanel, Projekt jpr13200) und unterscheidet sich vom Beispiel jpr13200 wie folgt:

- Beispiel jpr13310 enthält zwei verschiedene Pakete:
  - Paket **bsp13310.gui** enthält die Klassen für den Programmstart und die GUI
  - Paket **bsp13310.verarb** enthält die Klasse bsp13310.verarb.**Verarbeitung** mit Verarbeitung des Programms. Ferner ist in diesem Paket die Klasse bsp13310.verarb.**FehlerVerarbeitung** enthalten, die Methoden für eine Anzeige geeignet aufbereiteter Fehlermeldungen bereitstellt.

  Das Paket bsp13310.verarb wird in den nachfolgenden Beispielen jpr13320 und jpr13330 weiter verwendet, um auch dort jeweils einen Test der Datenbankverbindung im Programm anzubieten sowie um geeignet aufbereitete Fehlermeldungen anzuzeigen.

- Die Klasse bsp13310.verarb.Verarbeitung enthält den Code für den Aufbau der Datenbankverbindung. Der Aufbau der Datenbankverbindung erfolgt mit der in Kapitel 14 erklärten Technik nach Klick auf den Button "Datenbankverbindung testen". Die Ereignisbehandlung zu dieser Benutzeraktion befindet sich in der statischen Methode
  ```
  public static void testeDatenbankVerbindung(MainPanel mp)
  ```
  der Klasse bsp03310.verarb.Verarbeitung.
- Das Programm greift auf die Testdatenbank testdb14011 zu. Hierzu sollte ein Logon mit dem Benutzerkonto user14011 und dem Kennwort pw14011 erfolgen.
- Der Verbindungsaufbau zur Datenbank erfolgt mit der Klasse de.eikelberg.dbtools.DatenbankVerbindung. Hierzu werden die jar-Dateien de_eikelberg_dbtools.jar und mysql-connector-java-5.1.30-bin.jar benötigt. Diese müssen im Java-Basisverzeichnis gespeichert sein und in die Batch-Datei $start-mysql.bat eingetragen werden. Da das Programm einen Datenbankzugriff durchführt, wird der Name der Batchdatei für den Programmstart um den Namen des Datenbanksystems erweitert, auf den das Programm zugreift.
- Die Informationen für den Verbindungsaufbau zur Datenbank müssen in Konfigurationsdateien dbverbindung.properties und dbmsverbindung.properties eingetragen sein. Beide Dateien müssen im Ordner config gespeichert sein.
- Es muss für die Speicherung der Logs zum Verbindungsaufbau ein Ordner logs vorhanden sein.

Insgesamt liegt somit folgende Projektstruktur vor:



Wie bereits oben erwähnt, erfolgt der Datenbankzugriff in der Methode

`public static void testeDatenbankVerbindung(MainPanel mp)`

Diese Methode stellt die Ereignisbehandlung zum Button "Datenbankverbindung testen" im MainPanel bereit. Da das Panel LogonPanel in das Panel MainPanel eingebettet ist, kann in der Methode testeDatenbankVerbindung(..) mit

`mp.getLogonPanel()`

auf das LogonPanel zugegriffen werden. Mit Hilfe der folgenden Instanzmethoden der Klasse LogonPanel (vgl. Kapitel 13.4.5.2.3) wird dann auf die Anmeldeinformationen zugegriffen:

- `mp.getLogonPanel().isLogon()` testet, ob ein Logon vorgenommen wurde und ob somit Anmeldeinformationen vorhanden sind.
- `mp.getLogonPanel().getBenutzername()` gibt den beim Logon angegebenen Benutzernamen zurück.
- `mp.getLogonPanel().getKennwort()` gibt das beim Logon angegebene Kennwort zurück.

Unter Verwendung dieser Methoden wird dann wie folgt die Datenbankverbindung getestet:

Der nachfolgende Quellcode aus Paket bsp13310.verarb enspricht der in Kapitel 14.1-14.2 erklärten Technik und zeigt damit das Vorgehen für einen Verbindungsaufbau zu einer Datenbank gemäß Kapitel 14!

Im gelisteten Quellcode sind die Zugriffe auf das Logon-Panel **rot** markiert. Stellen, an denen auf die Rerferenz auf das MainPanel-Objekt zugegriffen wird, sind **fett** hervorgehoben. Alle Stellen, an denen mit Anweisungen gemäß Kapitel 14 gearbeitet wird, sind **blau** markiert:

```
package bsp13310.verarb;

import javax.swing.JOptionPane;

// Der in der nächsten Zeile genannte Paketname muss angepasst werden,
// wenn die Klasse bsp13310.verarb.Verarbeitung mit einem Panel MainPanel
// verwendet wird, das in einem anderen Paket angeordnet ist
import bsp13310.gui.MainPanel;
```

```
import de.eikelberg.dbtools.DatenbankVerbindung;

public class Verarbeitung
{
  private Verarbeitung() { }

  public static void testeDatenbankVerbindung(MainPanel mp)
  {
//  Meldung ausgeben
    System.out.println("\nButton \"Datenbankverbindung testen\" wurde aktiviert!");

    if (!mp.getLogonPanel().isLogon())
    {
      JOptionPane.showMessageDialog(
          mp,
          "Sie müssen sich anmelden, um eine\n"+
          "Datenbankverbindung aufbauen zu können",
          "Benutzerinformation",
          JOptionPane.ERROR_MESSAGE
      );
    }
    else
    {
// Benutzername und Kennwort aus GUI holen
      String benutzername = mp.getLogonPanel().getBenutzername();
      String kennwort = mp.getLogonPanel().getKennwort();

// Datenbankverbindung aufbauen
      DatenbankVerbindung myDB = null;
      myDB = new DatenbankVerbindung(benutzername,kennwort);
      boolean verarbOK = true;

      if (myDB.getException() != null)
      {
        FehlerVerarbeitung.meldeDbVerbindungsFehler(
            mp,
            myDB,
            "Fehler beim Aufbau der Datenbankverbindung!",
            "Verbindungsfehler"
        );
        verarbOK = false;
      }
      else
      {
// Debug-Option aktivieren
        myDB.setDebugOn();

// Datenbankverbindung schließen
        myDB.closeDatenbankVerbindung();
      }

      if (myDB.getException() != null & verarbOK)
      {
        FehlerVerarbeitung.meldeDbVerbindungsFehler(
            mp,
            myDB,
            "Fehler beim Abbau der Datenbankverbindung!",
            "Verbindungsfehler"
        );
        verarbOK = false;
      }

      if (verarbOK)
      {
        JOptionPane.showMessageDialog(
            mp,
            "Datenbankverbindung wurde erfolgreich getestet!\n"+
            "Benutzername: "+benutzername+"\n"+
            "Datenbank: "+myDB.getDB()+"\n"+
            "JDBC-Treibername: "+myDB.getJdbcTreiberName()+"\n",
            "Benutzerinformation",
```

```
            JOptionPane.INFORMATION_MESSAGE
        );
      }
    }
  }
}
```

Im obigen Quellcode wird auch eine der beiden Methoden der Klasse bsp13310.verarb.FehlerVerarbeitung aufgerufen. Diese beiden Methoden haben folgenden Einsatzzweck:

- `public static void` **`meldeDbVerbindungsFehler`**`(MainPanel mp, DatenbankVerbindung myDB, String fehlertext, String fehlerart)` zeigt ein JOptionPane-Fenster an,
  - das modal zum MainPanel mp gesetzt wird,
  - das die im Objekt myDB der Klasse DatenbankVerbindung gespeicherte Exception **zum letzten über das Objekt myDB Anweisung für einen Verbindungsaufbau oder Verbindungsabbau** auswertet und aus der Exception einen geeigneten Fehlertext extrahiert,
  - das die angezeigte Fehlermeldung mit der angegebenen Zeichenkette fehlertext beginnt und
  - das das Fenster mit dem Text fehlerart beschriftet.

  Die Methode meldeDbVerbindungsFehler(...) sollte speziell nach einem Verbindungsaufbau oder Verbindungsabbau eingesetzt werden.

- `public static void` **`meldeDbVerarbeitungsFehler`**`(MainPanel mp, DatenbankVerbindung myDB, String fehlertext, String fehlerart)` zeigt ein JOptionPane-Fenster an,
  - das modal zum MainPanel mp gesetzt wird,
  - das die im Objekt myDB der Klasse DatenbankVerbindung gespeicherte Exception **zur letzten über das Objekt myDB abgesetzten Anweisung** auswertet und aus der Exception einen geeigneten Fehlertext extrahiert,
  - das die angezeigte Fehlermeldung mit der angegebenen Zeichenkette fehlertext beginnt und
  - das das Fenster mit dem Text fehlerart beschriftet.

  Die Methode meldeDbVerarbeitungsFehler(...) kann nach jedem Vorgang zu einem Objekt der Klasse DatenbankVerbindung eingestetzt werden. Nach einem Verbindungsaufbau oder Verbindungsabbau ergibt die Methode meldeDbVerbindungsFehler(...) jedoch detailliertere Fehlerangaben als die Methode meldeDbVerarbeitungsFehler(...).

Man beachte:

> Die Klassen des Paketes bsp13310.verarb und die in diesen Klassen enthaltenen Methoden für den Test der Datenbankverbindung sowie für die Anzeige von Fehlermeldungen zur Datenbankverbindung werden in allen nachfolgenden Beispielen von Kapitel 13.5 eingesetzt.

Exemplarisch wird hier der Quellcode der in der Klasse bsp13310.verarb.Verarbeitung verwendeten Methode meldeDbVerbindungsFehler(...) angezeigt, wobei wieder der Quellcode geeignet eingefärbt wird, um den Zugriff auf das LogonPanel, das MainPanel sowie auf die DatenbankVerarbeitung zu kennzeichnen:

```
public static void meldeDbVerbindungsFehler(
    MainPanel mp, DatenbankVerbindung myDB, String fehlertext, String fehlerart)
{
  String fehlermeldung =
    fehlertext+
    "\nBenutzername: "+mp.getLogonPanel().getBenutzername()+
    "\nJDBC-Treibername: "+myDB.getJdbcTreiberName();

  if (myDB != null)
  {
    Exception e = myDB.getException();
    fehlermeldung =
      fehlermeldung + "\n" + formatiereExceptionText(e);
  }

  JOptionPane.showMessageDialog(
      mp,
      fehlermeldung,
      fehlerart,
      JOptionPane.ERROR_MESSAGE
  );
}
```

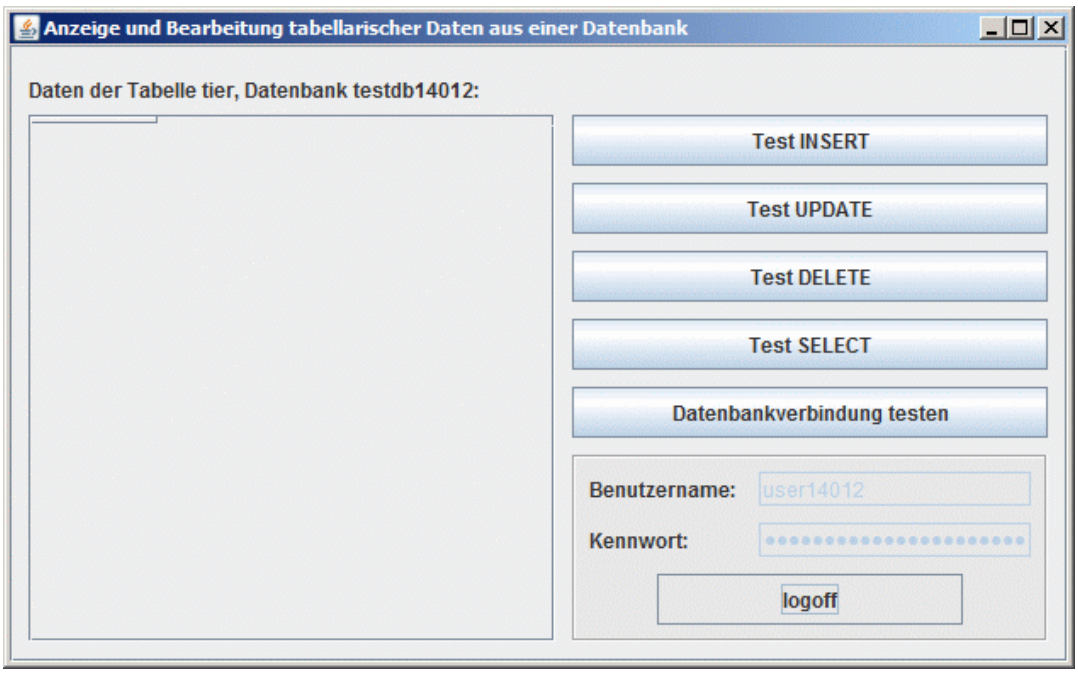
Die in obigem Quellcode enthaltene Methode

```
private static String formatiereExceptionText(Exception e)
```
bildet aus den im Exception-Objekt e enthaltenen Informationen eine Zeichenkette, die eine geeignete Fehlerbeschreibung enthält.

13.5.2 Ausführung von SQL-Anweisungen für den Zugriff auf die Daten einer Datenbank

13.5.2.1 Beispielprogramm jpr13320

Das Beispielprogramm jpr13320 zu diesem Kapitel hat folgende Benutzeroberfläche:



Das Beispielprogramm jpr13320 hat folgende Funktionalität:

- Nach Klick auf die Buttons "Test INSERT" bis "Test SELECT" wird jeweils eine entsprechend fest codierte SQL-Anweisung zu der Tabelle tier der Testdatenbank testdb14012 ausgeführt. Nach jeder verändernden Anweisung wird dabei jeweils einmal der Datenbestand der Tabelle tier gelesen und die Anzeige in der Benutzeroberfläche aktualisiert.
- Mit dem Button "Datenbankverbindung testen" kann wie in Kapitel 13.5.1 erklärt getestet werden, ob mit dem angegebenen Benutzerkonto und dem Kennwort eine Datenbankverbindung zur Testdatenbank testdb14012 auf dem Server, der in der Datei dbmsverbindung.properties eingetragen ist, aufgebaut werden kann.

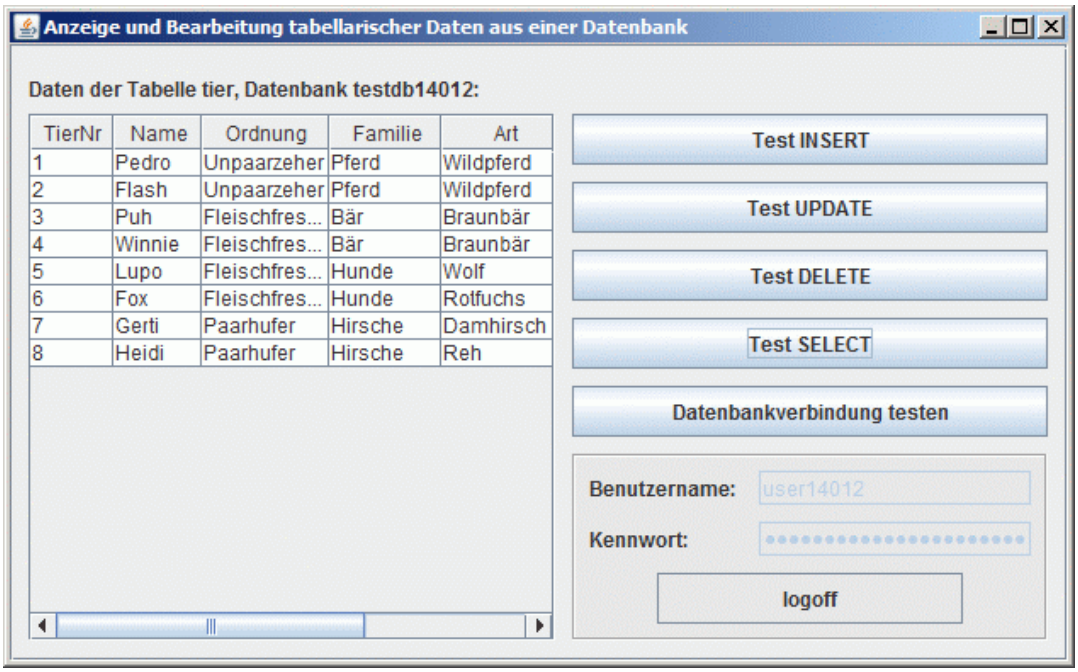
Hinweis: Das Beispielprogramm jpr13330 ist eine Erweiterung des Programms jpr13320 und kann auf eine Vielzahl von Tabellen angewendet werden. Es ermöglicht das Lesen, Einfügen, Verändern und Löschen frei wählbarer Daten (vgl. 13.5.2.4).

13.5.2.2 Lesender Zugriff auf die Daten einer Datenbank

Der lesende Zugriff auf die Daten einer Datenbank erfolgt stets mit der SQL-Anweisung SELECT. Diese SQL-Anweisung wird in Kapitel 14.3.2 erklärt. Im Beispiel jpr13320 erfolgt der lesende Zugriff nach Klick auf den Button "Test SELECT". Daraufhin wird die Methode

```
public static void Verarbeitung.liesDaten(MainPanel mp);
```

der Klasse bsp13320.verarb.Verarbeitung aufgerufen und es wird der Inhalt der Tabelle tier angezeigt:



Die lesende SQL-Anweisung für diesen Vorgang lautet:

```
SELECT * FROM tier ORDER BY tierNr
```

Sie wird in der Methode liesDaten(MainPanel mp) wie folgt abgesetzt - im Quellcode sind die Zugriffe auf das MainPanel, das LogonPanel sowie die DatenbankVerarbeitung wieder wie in Kapitel 15.1 farbig markiert:

```
  public static void liesDaten(MainPanel mp)
  {
//  Meldung ausgeben
    System.out.println("\nButton \"Anzeige aktualisieren\" wurde aktiviert!");

//  Anzeige aktualisieren
    if (!mp.getLogonPanel().isLogon())
    {
      JOptionPane.showMessageDialog(
          mp,
          "Sie müssen sich anmelden, um eine\n"+
          "Datenbankverbindung aufbauen zu können",
          "Benutzerinformation",
          JOptionPane.ERROR_MESSAGE
      );
    }
    else
    {
      getAktuelleDaten(mp);
    }
  }
```

Die Methode liesDaten(MainPanel mp) führt also nur eine kurze Verarbeitung durch. Das eigentliche Absetzen der SELECT-Anweisung wurde in eine Methode getAktuelleDaten(MainPanel mp), denn diese Anweisung für das Lesen der Daten und das Aktualisieren der GUI wird vom Programm jpr13320 auch nach dem Absetzen von verändernden Anweisungen ausgeführt:

```
//Methode für das Lesen und die Aktualisierung der Daten
//in der GUI, wird nach jedem Datenbankzugriff aufgerufen
  private static void getAktuelleDaten(MainPanel mp)
  {
    if (mp.getLogonPanel().isLogon())
    {
//    Benutzername und Kennwort feststellen
      String benutzername = mp.getLogonPanel().getBenutzername();
      String kennwort = mp.getLogonPanel().getKennwort();

//    Datenbankverbindung aufbauen
      DatenbankVerbindung myDB;
      myDB = new DatenbankVerbindung(benutzername,kennwort);
```

```
      boolean verarbOK = true;

// Lesender Zugriff auf die Datenbank
      ResultSet myresultset = null;
      if (myDB.getException() != null)
      {
        FehlerVerarbeitung.meldeDbVerbindungsFehler(
            mp,
            myDB,
            "Die Datenbankverbindung konnte nicht aufgebaut werden!",
            "Verbindungsfehler"
        );
        verarbOK = false;
      }
      else
      {
//      debug
        myDB.setDebugOn();

//      Select-Abfrage absetzen
        String query = "SELECT * FROM tier ORDER BY tierNr";
        myresultset = myDB.submitSelectQuery(query);
      }

      if ((myDB.getException() != null | myresultset == null) & verarbOK)
      {
        FehlerVerarbeitung.meldeDbVerarbeitungsFehler(
            mp,
            myDB,
            "Fehler beim Absetzen der SELECT-Anweisung!",
            "Fehler bei der Verarbeitung"
        );
        verarbOK = false;
      }
      else
      {
//      ResultSet verarbeiten
//      Spaltennamen feststellen und im Tabellenmodell speichern
        ...

//      Daten aus Resultset lesen und im Tabellenmodell speichern
        ...
      }

      if (verarbOK)
      {
//      Anzeige der Tabelle komplett neu aufbauen
        mp.getMyTableModel().fireTableChanged(
           new TableModelEvent(
             mp.getMyTableModel(),TableModelEvent.HEADER_ROW
           )
        );

//      ResultSet schließen
        myDB.schliesseResultSet(myresultset);
      }

      if (myDB.getException() != null & verarbOK)
      {
        JOptionPane.showMessageDialog(
            mp,
            "Fehler beim Schließen der Rückgabe einer SELECT-Anweisung!",
            "Fehler bei der Verarbeitung",
            JOptionPane.ERROR_MESSAGE
        );
        verarbOK = false;
      }
      else if (verarbOK)
      {
//      Datenbankverbindung schließen
        myDB.closeDatenbankVerbindung();
      }
```

```
      if (myDB.getException() != null & verarbOK)
      {
        FehlerVerarbeitung.meldeDbVerbindungsFehler(
            mp,
            myDB,
            "Fehler beim Abbau der Datenbankverbindung!",
            "Verbindungsfehler"
        );
        verarbOK = false;
      }
    }
  }
```

13.5.2.3 Verändernder Zugriff auf die Daten einer Datenbank

Der verändernde Zugriff auf die Daten einer Datenbank erfolgt mit den SQL-Anweisungen INSERT, UPDATE und DELETE. Diese SQL-Anweisungen werden in Kapitel 14.4.3 bis 14.4.5 erklärt. Das programmtechnische Vorgehen ist für alle drei SQL-Anweisungen im Prinzip gleich, so dass hier dieses Vorgehen exemplarisch anhand der INSERT-Anweisung vorgestellt wird.

Die INSERT-Anweisung wird nach Klick auf den Button "Test INSERT" ausgeführt. Daraufhin wird die Methode Daraufhin wird die Methode

```
public static void einfuegeDatensatz(MainPanel mp)
```

der Klasse bsp13320.verarb.Verarbeitung aufgerufen und es wird die folgende INSERT-Anweisung ausgeführt:

```
INSERT INTO tier
(
   TierNr , Name , Ordnung , Familie , Art , Geschlecht , GeburtsDatum
)
VALUES
(
   100 , 'Heinz' , 'Unpaarzeher' , 'Pferd', 'Rappe' , 'm' , '2004-01-01'
)
```

Da direkt nach dem Einfügen der Daten in die Tabelle tier vom Programm wieder die Methode getAktuelleDaten(MainPanel mp) ausgeführt wird, erscheinen die eingefügten Daten auch direkt in der GUI:

Die oben angeführte INSERT-Anweisung wird in der Methode einfuegeDatensatz(MainPanel mp) wie folgt an den Datenbankserver abgesetzt:

```
  public static void einfuegeDatensatz(MainPanel mp)
  {
//  Meldung ausgeben
    System.out.println("\nButton \"Datensatz einfuegen\" wurde aktiviert!");

    if (!mp.getLogonPanel().isLogon())
    {
      JOptionPane.showMessageDialog(
          mp,
          "Sie müssen sich anmelden, um eine\n"+
          "Datenbankverbindung aufbauen zu können",
          "Benutzerinformation",
          JOptionPane.ERROR_MESSAGE
      );
```

```
    }
    else
    {
//    Insert-Statement konstruieren
      int erfolg = 0;
      String insertstmt =
        "INSERT INTO tier "+
        "(TierNr,Name,Ordnung,Familie,Art,Geschlecht,GeburtsDatum) "+
        "VALUES "+
        "(100,'Heinz','Unpaarzeher','Pferd','Rappe','m','2004-01-01')";

      JOptionPane.showMessageDialog(
          mp,
          "Es wird folgender SQL-Befehl abgesetzt:\n"+
          insertstmt,
          "Benutzerinformation",
          JOptionPane.INFORMATION_MESSAGE
      );

// Benutzername und Kennwort aus GUI holen
      String benutzername = mp.getLogonPanel().getBenutzername();
      String kennwort = mp.getLogonPanel().getKennwort();

// Datenbankverbindung aufbauen
      DatenbankVerbindung myDB = null;
      myDB = new DatenbankVerbindung(benutzername,kennwort);
      boolean verarbOK = true;

// Datensatz einfügen
      if (myDB.getException() != null)
      {
        FehlerVerarbeitung.meldeDbVerbindungsFehler(
            mp,
            myDB,
            "Die Datenbankverbindung konnte nicht aufgebaut werden!",
            "Verbindungsfehler"
        );
        verarbOK = false;
      }
      else
      {
// Debug-Option aktivieren
        myDB.setDebugOn();

// Insert-Statement absetzen
        erfolg = myDB.submitInsertSqlBefehl(insertstmt);
      }

      if (myDB.getException() != null & verarbOK)
      {
        FehlerVerarbeitung.meldeDbVerarbeitungsFehler(
            mp,
            myDB,
            "Fehler beim Absetzen der INSERT-Anweisung!",
            "Fehler bei der Verarbeitung"
        );
        verarbOK = false;
      }
      else if (verarbOK)
      {
//      Datenbankverbindung schließen
        myDB.closeDatenbankVerbindung();
      }

      if (myDB.getException() != null & verarbOK)
      {
        FehlerVerarbeitung.meldeDbVerbindungsFehler(
            mp,
            myDB,
            "Fehler beim Abbau der Datenbankverbindung!",
            "Verbindungsfehler"
        );
```

```
        verarbOK = false;
      }

//    Anzeige aktualisieren
      if (verarbOK)
      {
        getAktuelleDaten(mp);
      }

//    Meldung
    ...

    }
  }
```

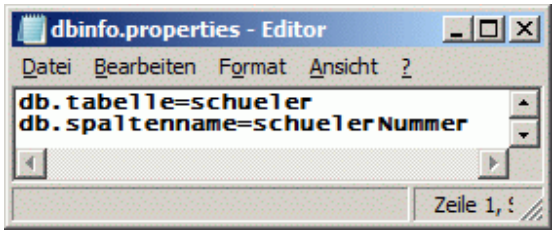
13.5.2.4 Beispielprogramm jpr13330

Das Beispielprogramm jpr13330 erweitert das Programm jpr13320 so, dass

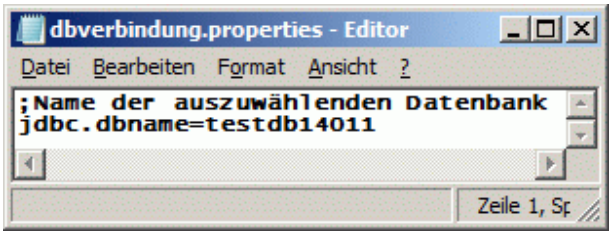
- für jede beliebige Datenbank
- jede Tabelle verarbeitet werden kann, deren Spalten einen Zahl-Datentyp oder den Datentypen char, varchar bzw. date haben.
- Für diese Tabelle können dann SELECT-, INSERT-, UPDATE- und DELETE-Anweisungen für beliebige Daten bzw. Datensätze abgesetzt werden.

Um das Programm jpr13330 zum Beispiel auf die Tabelle schueler der Testdatenbank testdb14011 anzuwenden, muss das Programm wie folgt geändert werden (vgl Programm jpr13331):

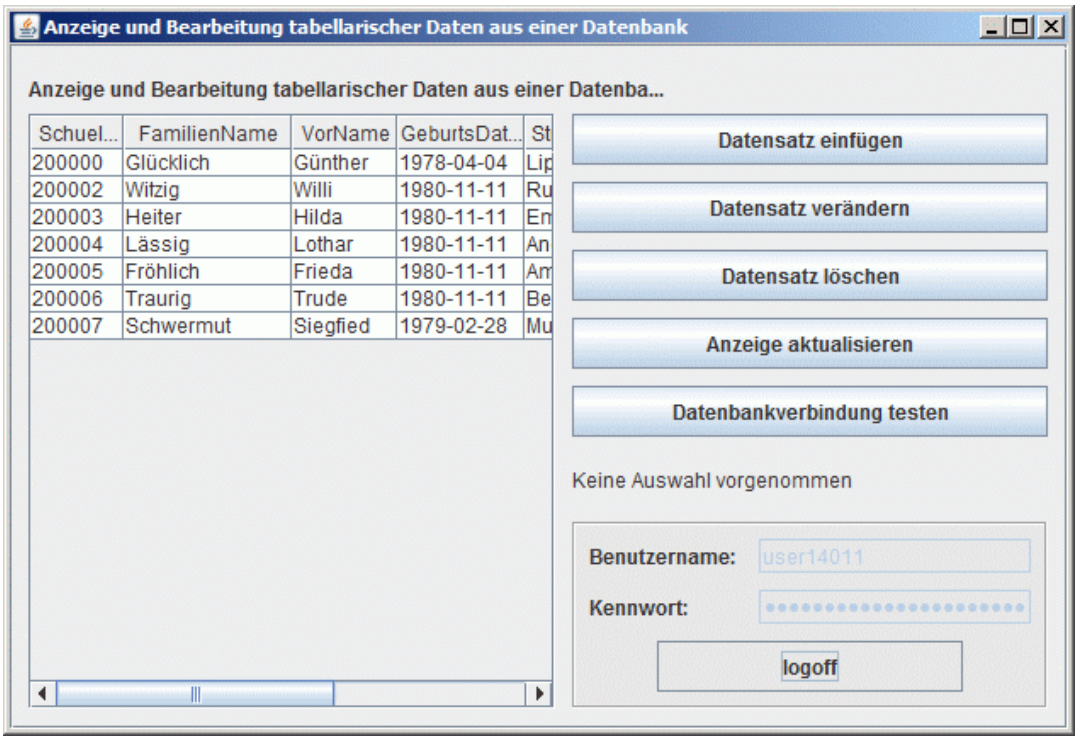
- Der Zugriff auf die Datenbank testdb14011 erfolgt mit dem Benutzerkonto user14011 und dem Kennwort pw14011. Daher muss die Konsolmeldung zur Benutzeranmeldung in der Klasse bsp13330.gui.Programmstart geeignet angepasst werden.
- In der Datei config/dbinfo.properties müssen die Namen der Tabelle (schueler) und der Name der Spalte, nach der sortiert werden soll (SchuelerNummer) eingetragen werden:



- In der Datei config/dbverbindung.properties muss der Name der Datenbank eingetragen werden:



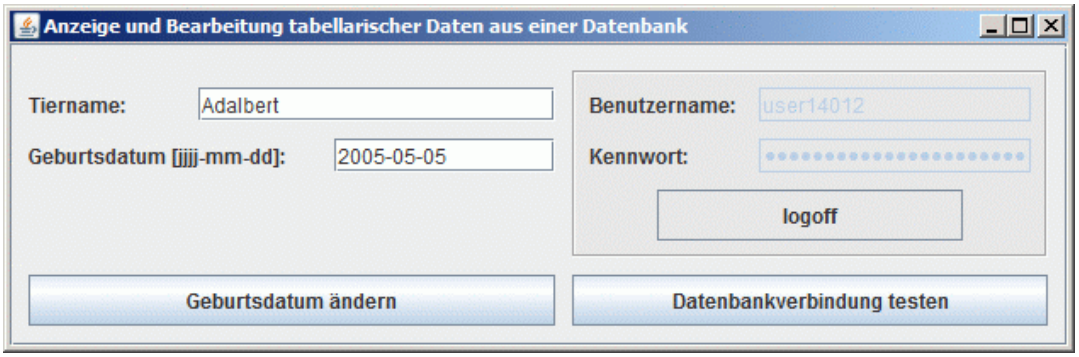
- Der Logon muss natürlich mit dem Benutzerkonto user14011 erfolgen:



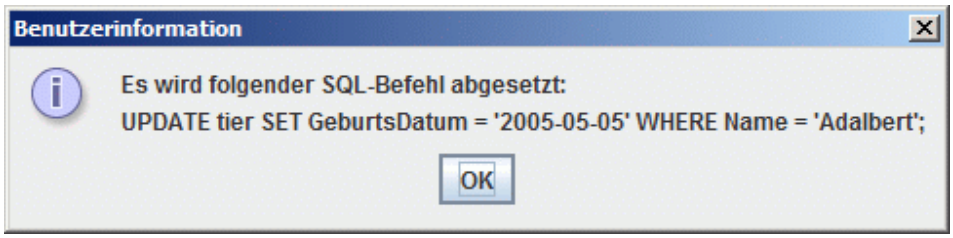
Man beachte, dass auch ohne Klick auf den Button "Anzeige aktualisieren" nach kurzer Zeit der Inhalt der Tabelle schueler angezeigt wird. Das Programm startet nämlich einen sogenannten Thread (vgl. Kapitel 3.9), der alle 15 Sekunden die Anzeige der Tabelle aktualisiert!

13.5.2.5 Exkurs: Anzeige von Werten für WHERE-Klauseln in einer JComboBox

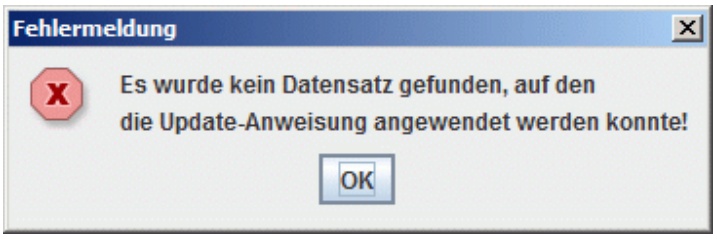
Werte, die für die Codierung von WHERE-Klauseln benötigt werden, müssen in der Datenbank vorhanden sein, wenn anhand der WHERE-Klausel Datensätze gefunden werden sollen. Beispiel jpr13341 zeigt diese Problematik. Wird dieses Beispielprogramm gestartet, so kann nach dem Logon ein Tiername angegeben werden, für den das gespeicherte Geburtsdatum geändert werden soll:



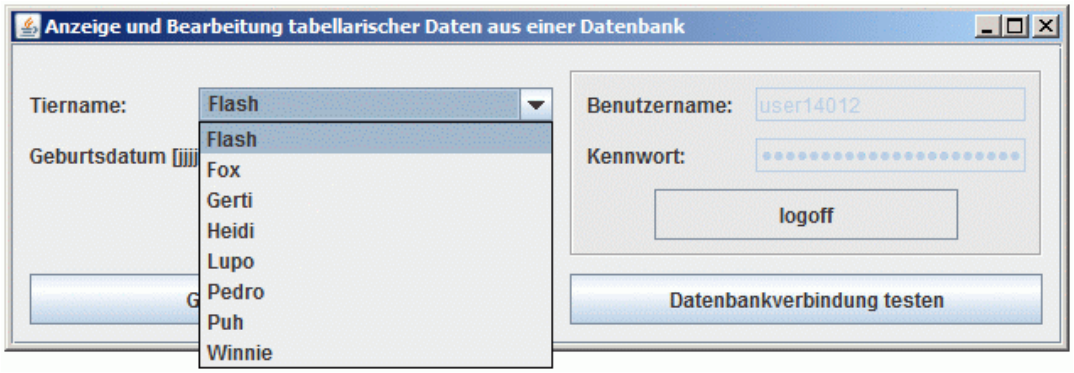
Daraufhin wird vom Programm folgender SQL Befehl generiert und abgesetzt:



Gibt es in der Datenbank jedoch kein Tier mit dem Namen Adalbert, so wird auch kein Datensatz geändert:



Möchte man sicherstellen, dass vom Anwender nur Werte eingegeben werden, die auch in der Datenbank vorhanden sind, muss die Benutzereingabe - am besten mit Hilfe einer JComboBox - wie im Beispiel jpr13342 abgesichert werden:



Diese Absicherung geschieht

- mit Hilfe eines Datenmodells für die JComboBox (Klasse bsp13342.gui.ComboBoxTierNameModel),
- einer Methode zur Aktualisierung der Liste der anzuzeigenden Einträge der JComboBox (Methode aktualisiereOptionen() in der Klasse bsp13342.gui.ComboBoxTierNameModel) sowie
- mit einer Ereignisbehandlung für Focus-Ereignisse für die JComboBox, so dass die Liste der anzuzeigenden Optionen immer dann neu aufgebaut wird, wenn die JComboBox den Focus erhält. Die Aktualisierung der Anzeige erfolgt dann automatisch ohne dass sie vom Programmierer angefordert werden muss.

Insgesamt wird die JComboBox im Panel mit der GUI (bsp13342.gui.MainPanel) wie folgt erzeugt:

```
  private JComboBox<String> cbTiername;
  private ComboBoxTierNameModel cmmTierNameModel;

  ...

// Sichtbarkeit manuell hochgesetzt auf public
  public JComboBox<String> getCmbTiername()
  {
    if (cbTiername == null)
    {
      cbTiername = new JComboBox<String>();
      cbTiername.addFocusListener(new FocusAdapter()
      {
// Das Aktualiseren der Optionen geschieht immer dann, wenn die JComboBox
// den Focus erhält. Hierzu wird von der Methode aktualisiereOptionen()
// der Array mit den anzuzeigenden Optionen neu erstellt
        @Override
        public void focusGained(FocusEvent arg0)
        {
          getCmmTierNameModel().aktualisiereOptionen();
        }
      });
      cbTiername.setBounds(110, 25, 210, 20);

// Manuell eingefügt
      cbTiername.setModel(getCmmTierNameModel());
    }
    return cbTiername;
  }

  /**
   * @wbp.nonvisual location=25,210
   */
```

```
private ComboBoxTierNameModel getCmmTierNameModel()
{
  if (cmmTierNameModel == null)
  {
    cmmTierNameModel = new ComboBoxTierNameModel(this);
  }
  return cmmTierNameModel;
}
```