Erstellen einer Datenbankanwendung mit dem Java-Editor

Wenn Sie alle Vorbereitungen aus der Anleitung *Java-Editor und SQLite-Datenbanken*[1] vorgenommen haben, kann es mit der Programmierung eines *JFrames* für die Datenbankanwendung losgehen. Die Anleitung bezieht sich auf solche Anwendungen, die passend zu einigen Benutzereingaben, eine Information liefern. Solche Anwendungen begegnen uns im Internet ständig, etwa wenn wir bei einem Onlineshop nach einem Produkt suchen oder bei einem sozialen Netzwerk Informationen über unsere Freunde erhalten. Im Folgenden wird die Erstellung einer solchen Datenbankanwendung Schritt für Schritt erklärt. Als kleines Beispiel betrachten wir eine Anwendung, die für einen Schüler oder eine Schülerin das Geburtsdatum aus einer Datenbank heraussucht. Sie können sich den Quellcode in der Datei *SucheGebDatum.java* aus dem Ordner *Beispiele → Java -SucheGebDatum* anschauen.

Schritt 1: Erstellen einer Benutzeroberfläche

Zunächst schauen wir uns an, wie man die Benutzeroberfläche erstellt, um einige Informationen vom Anwendenden zu erfragen. Um zum Beispiel das Geburtsdatum eines Schülers oder einer Schülerin zu finden, benötigen wir den Vornamen und den Nachnamen. Da Namen beliebige Zeichenketten sind, eignen sich dafür Textfelder, in die der Anwender bzw. die Anwenderin die Information eingibt.

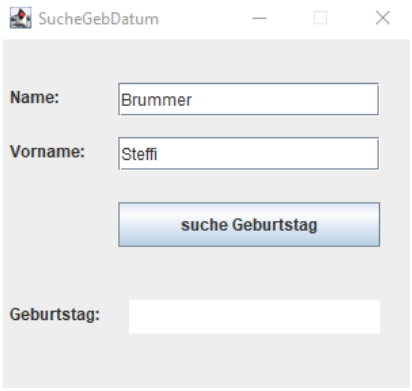


Abbildung 1: Benutzeroberfläche für die Suche nach dem Geburtsdatum

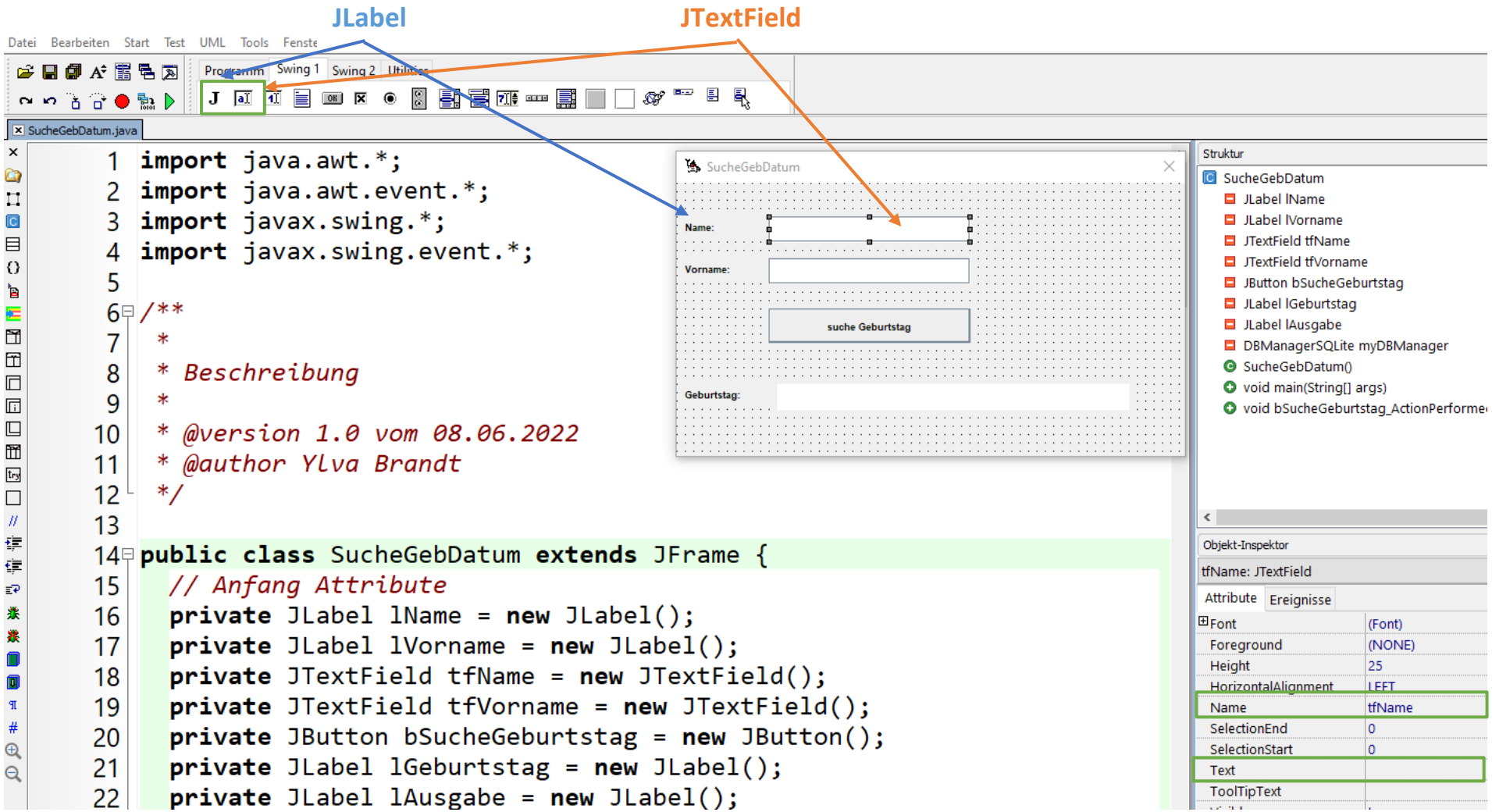
Erzeugen eines Textfeldes mit Beschriftung

Textfelder sind Objekte der Klasse *JTextField*, die wir im *Java-Editor* bei *Swing 1* finden (s. Abbildung 2). Nachdem wir ein Textfeld in das Designformular gezogen haben, können wir im Objektinspektor beim Attribut *Name* den Namen für die Objekt-Variable bestimmen. Beim Attribut *Text* können wir den Text angeben, der beim Starten des *JFrames* im Textfeld angezeigt werden soll. Wenn wir hier keinen Text angeben, ist das Textfeld zu Beginn leer.

Damit klar ist, welche Information er in ein Textfeld eingeben werden soll, ist es hilfreich dieses zu beschriften. Um Text, der nicht verändert werden kann, auf der Benutzeroberfläche auszugeben, verwenden wir ein *JLabel*. Dieses finden wir ebenfalls unter *Swing 1*. Im Objektinspektor können wir beim Attribut *Text*, den Text eingeben, der angezeigt werden soll. Beim Attribut *Name* können wir auch den Namen der entsprechenden Objektvariablen für das *JLabel* ändern.

In *Abbildung 2* sind alle wichtigen Stellen im *Java-Editor* umrandet. Der entsprechende Quelltext wird automatisch erzeugt.

[1] Falls Sie eine andere Datenbank verwenden und die entsprechenden Vorbereitungen getroffen haben, können Sie diese Anleitung ebenso verwenden. Die Klasse *DBManagerSQLite müsste dann von der Lehrkraft an das verwendete Datenbanksystem angepasst werden.*




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5
6  /**
7   *
8   * Beschreibung
9   *
10  * @version 1.0 vom 08.06.2022
11  * @author Ylva Brandt
12  */
13
14 public class SucheGebDatum extends JFrame {
15   // Anfang Attribute
16   private JLabel lName = new JLabel();
17   private JLabel lVorname = new JLabel();
18   private JTextField tfName = new JTextField();
19   private JTextField tfVorname = new JTextField();
20   private JButton bSucheGeburtstag = new JButton();
21   private JLabel lGeburtstag = new JLabel();
22   private JLabel lAusgabe = new JLabel();
```

Abbildung 2: Erstellen eines JLabel und eines JTextField im JavaEditor

Grundsätzlich kann jede Information über ein Textfeld erfragt werden. Möchte man aber zum Beispiel bei der Angabe eines Schulfaches Schreibfehler und Fantasiefächer vermeiden oder bei der Eingabe einer Note sicherstellen, dass eine Zahl zwischen 0 und 15 gewählt wird, müsste man die Eingabe in ein Textfeld aufwändigen Tests unterziehen. Alternativ kann man daher andere Dialogelemente für die Eingabe wählen, die die möglichen Antworten von vorneherein einschränken und damit ungültige Angaben verhindern. Im Anhang befindet sich daher eine Anleitung zum Erzeugen und Auswerten einiger weiterer hilfreicher Dialogelemente.

Erzeugen eines JButtons

Wenn die Anwendenden mit der Eingabe aller notwendigen Informationen fertig sind, sollten sie uns das mitteilen können, indem sie einen *Button* betätigen. In unserem Beispiel ist das der *Button* mit der Aufschrift *suche Geburtstag*.

Einen *JButton* finden wir bei *Swing 1*. Im Objektinspektor können wir beim Attribut *Text* die Beschriftung des *JButtons* angeben. Der Variablenname steht beim Attribut *Name*. Dieser wird automatisch an die Beschriftung angepasst, kann aber natürlich auch noch geändert werden.

Wenn wir im Designformular unseres *JFrames* einen Doppelklick auf den *JButton* machen, wird im Quellcode automatisch eine Methode erzeugt, die immer dann ausgeführt wird, wenn der Button in der Anwendung angeklickt wurde. Die Methode ist in Abbildung 3 blau hinterlegt. Der Methodenname ist in der Regel der Name der Objektvariablen für den Button ergänzt um *_ActionPerformed*. Dieser sollte nicht geändert werden, da er an anderen Stellen im automatisch generierten Quelltext verwendet wird und eine Änderung des Namens somit zu Fehlern beim Kompilieren führen würde. In dieser Methode ergänzen wir nun unseren Quellcode, der ausgeführt werden soll, wenn alle Informationen für die Suche eingegeben wurden. Das schauen wir uns im nächsten Schritt genauer an.

JButton

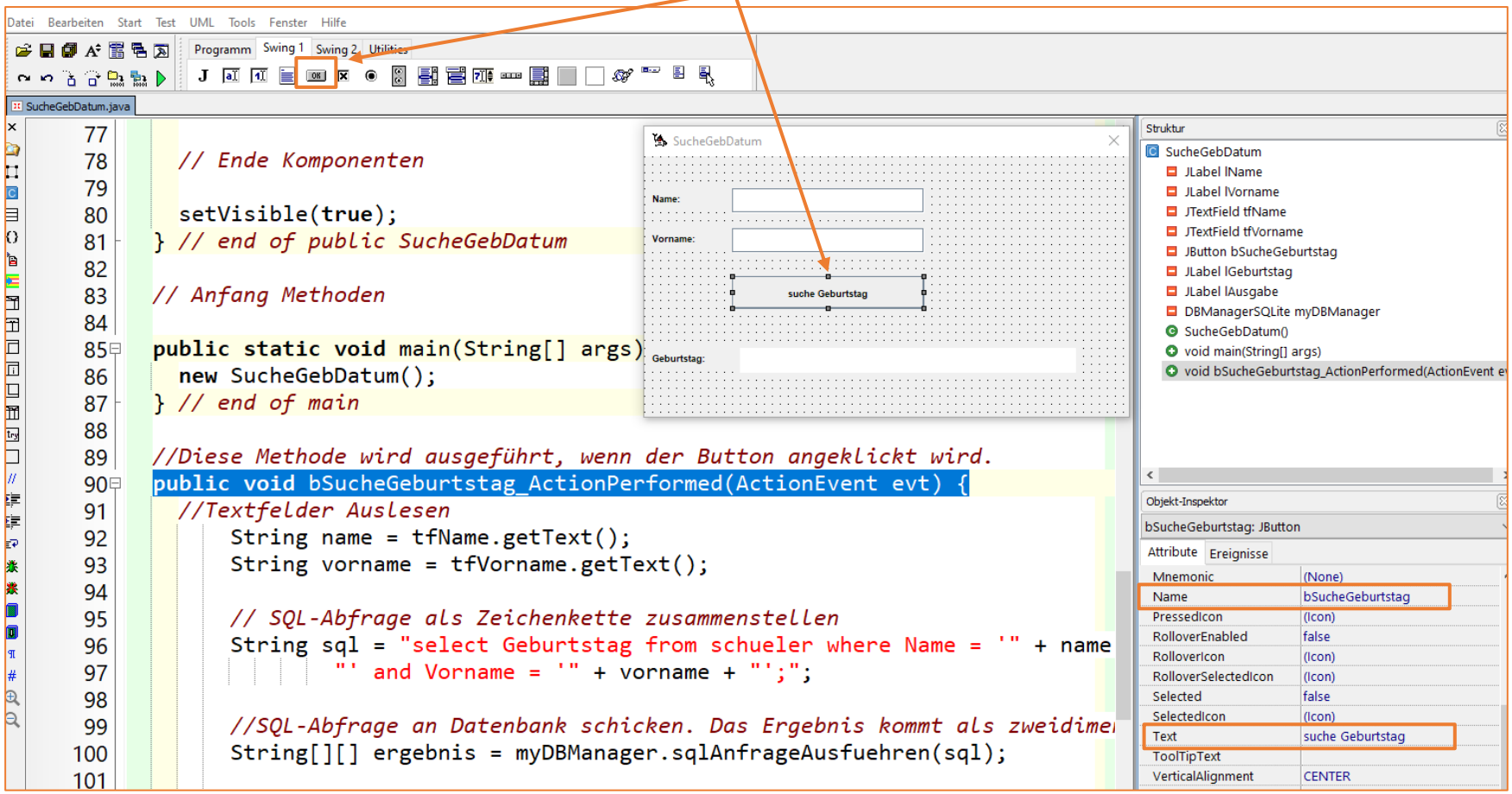


Abbildung 3: Erstellen eines JButton im Java-Editor

Schritt 2: Datenbankanfrage stellen

Schritt 2a: Benutzereingaben speichern - Auswerten eines Textfeldes

Als erstes sollten wir die Nutzereingaben in geeigneten Variablen speichern, damit uns diese Informationen für die weitere Arbeit zur Verfügung stehen. Dazu müssen wir die Texte aus unseren Textfeldern auslesen. Die Klasse *JTextField* stellt dafür die Methode `getText` zur Verfügung. Diese liefert den eingetragenen Text als Zeichenkette zurück. In unserem Beispiel beginnen wir also in der Methode `bSucheGeburtstag_ActionPerformed` mit den folgenden Codezeilen

```
String name = tfName.getText();
String vorname = tfVorname.getText();
```

Dadurch wird der Text aus dem Textfeld `tfName` in der Variablen `name` gespeichert und der Text aus dem Textfeld `tfVorname` in der Variablen `vorname`.

Schritt 2b: SQL-Anweisung zusammensetzen

Mithilfe der gespeicherten Informationen können wir nun die *SQL-Anfrage* für die Datenbank erstellen. Diese speichern wir als *Zeichenkette* in einer Variablen ab. Für unser Beispiel sieht das so aus:

```
String sql = "select Geburtstag from schueler where Name = '" + name +"'
and Vorname = '" + vorname + "';";
```

Wenn jemand in das Textfeld für den Namen „Brummer“ und in das Textfeld für den Vornamen „Steffi“ eingegeben hat, würde in der Variablen `sql` die folgende Anfrage gespeichert:

```
select Geburtstag from schueler where Name = 'Brummer' and Vorname =
'Steffi';
```

Nach welchem Schüler bzw. welcher Schülerin gesucht wird, hängt also von den Nutzereingaben ab, da in der SQL-Anfrage die Werte der Variablen `name` und `vorname` eingesetzt werden.

Schritt 2c: SQL-Abfrage an Datenbank schicken

Nun muss die Anfrage an die Datenbank geschickt werden, um das passende Ergebnis zu erhalten. Dazu benötigen wir ein Objekt der Klasse *DBManagerSQLite*, das für uns die Kommunikation mit der Datenbank regelt. Deshalb definieren wir eine globale Variable vom Typ *DBManagerSQLite*:

```
private DBManagerSQLite myDBManager = new DBManagerSQLite();
```

Als Variablenname wurde hier `myDBManager` gewählt. Dieser kann natürlich auch anders lauten. Wenn der *Konstruktor* ohne Parameter aufgerufen wird, wird eine Verbindung zu der mitgelieferten Beispieldatenbank *schule_erweitert* hergestellt. Soll eine andere Datenbank verwendet werden, muss der Name der Datenbank OHNE die Endung *.db* als Zeichenkette übergeben werden. Bei der folgenden Programmzeile würde z. B. eine Verbindung zur Datenbank *schule* hergestellt:

```
private DBManagerSQLite myDBManager = new DBManagerSQLite("schule");
```

Die Klasse *DBManagerSQLite* stellt die Methode `sqlAnfrageAusfuehren` zur Verfügung. Dieser kann unsere SQL-Anfrage als Zeichenkette übergeben werden. Da das Ergebnis einer SQL-Anfrage in der Regel eine Tabelle ist, liefert die Methode `sqlAnfrageAusfuehren` eine zweidimensionale Reihung vom Typ *Zeichenkette* zurück. Die folgende Codezeile führt unsere Anfrage aus der Variablen `sql` aus und speichert das Ergebnis in der Variablen `ergebnis`:

```
String[][] ergebnis = myDBManager.sqlAnfrageAusfuehren(sql);
```

Schritt 3: Ergebnispräsentation für den Anwender

Das Ergebnis der Abfrage

Die erste Zeile der Tabelle bzw. der zweidimensionalen Reihung `ergebnis` enthält die Spaltenüberschriften. Jeder Ergebnisdatensatz folgt in einer weiteren Zeile. Da die Reihung vom Typ *Zeichenkette* ist, liegen alle Attribute als Zeichenketten vor, auch wenn sie in der Datenbank andere Datentypen haben. Da die Ausgabe für die Anwendenden in der Regel sowieso als Zeichenkette erfolgt, wird das normalerweise kein Problem sein. Benötigt man eine Zahl doch einmal als Variable vom Typ *Ganzzahl* oder *Fließkommazahl*, kann man die Zeichenkette natürlich entsprechend umwandeln (s. Anhang Umwandeln von Zeichenketten in Zahlen, S.23).

Bei der zweidimensionalen Reihung interpretieren wir den ersten Index als Zeilennummer und den zweiten Index als Spaltennummer. In unserem Beispiel wird nur nach *einem* Geburtsdatum gesucht, und zwar für Steffi Brummer. Wenn wir davon ausgehen, dass es nur eine Schülerin mit dieser Kombination aus Vor- und Nachname gibt, enthält unsere Tabelle also nur eine Spalte und zwei Zeilen: eine Zeile für die Überschrift und eine Zeile für das Geburtsdatum. An der Position `ergebnis[0][0]` wird also die Zeichenkette „Geburtstag“ stehen und an der Position `ergebnis[1][0]` die Zeichenkette „2006-07-16“, da Steffi Brummer am 16.7.2006 geboren wurde. Als Tabelle können wir uns das so vorstellen:

Index	**0**
0	Geburtstag
1	2006-07-16

Nun gibt es natürlich auch die Möglichkeit, dass ein Name eingegeben wurde, den es in der Datenbank gar nicht gibt. In diesem Fall würde die Anfrage keine Datensätze zurückliefern. Die zweidimensionale Reihung würde dann nur eine Zeile mit den gewünschten Attributen als Spaltenüberschriften enthalten. Es gäbe aber keine weiteren Zeilen, da keine passenden Datensätze gefunden wurden. In unserem Beispiel gäbe es also nur das Feld `ergebnis[0][0]` mit dem Inhalt „Geburtstag“.

Es kann noch ein weiterer Fall auftreten. Wenn die SQL-Anfrage von der Datenbank nicht ausgeführt werden konnte, weil die SQL-Anfrage z. B. einen Syntaxfehler enthält, so besteht die zweidimensionale Reihung ebenfalls aus nur einem Feld, allerdings mit dem Inhalt „Fehler“. Diese Abfrage hat dann vermutlich auch eine *SQLException* ausgelöst, zu der man nähere Informationen unter *Meldungen* im *Java-Editor* findet. Um solche Fehler weitgehend auszuschließen, ist es sinnvoll die *SQL-Anfrage* zunächst in einem *Datenbankbrowser*[2] zu testen, in den man *SQL-Anfragen* direkt eingeben und an die Datenbank senden kann.

[2] Als *Datenbankbrowser* kann z.B. das Programm *DB Browser for SQLite* verwendet werden: http://sqlitebrowser.org/ [Datum des Zugriffs: 02.02.2024]

Das Ergebnis präsentieren

Das Ergebnis unserer SQL-Anfrage soll nun dem Anwender über die Benutzeroberfläche mitgeteilt werden. In unserer Beispiel-Anwendung wurde dafür ein weiteres *JLabel* vorbereitet. In dieses können wir nun mithilfe der Methode `setText` das gefundene Geburtsdatum schreiben.
Allerdings müssen wir dabei die soeben beschriebenen drei Fälle berücksichtigen, die beim Ausführen einer SQL-Anfrage auftreten können:

- 1. Fall: Es wurden passende Datensätze gefunden.
- 2. Fall: Die SQL-Anfrage war fehlerhaft.
- 3. Fall: Es wurde kein passender Datensatz gefunden.

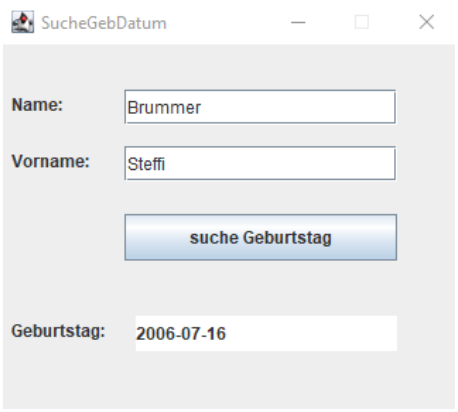


Abbildung 4: Ausgabe des Geburtsdatums

Der folgende Quelltext enthält daher einige Verzweigungen:

```
1  if(ergebnis.length > 1)
2     lAusgabe.setText(ergebnis[1][0]);                          //1. Fall
3  else{
4     if(ergebnis.length == 1){
5        if(ergebnis[0][0].equals("Fehler")) lAusgabe.setText("Es
            ist ein Fehler aufgetreten.");                       //2. Fall
6        else lAusgabe.setText("Diese Person gibt es in der
                                   Datenbank nicht!");           //3. Fall
7     }
8  }
```

Wenn die erste Bedingung in grün erfüllt ist, die Ergebnistabelle also aus mindestens zwei Zeilen besteht, so enthält die zweite Zeile das Geburtsdatum. Diese wird mit der Methode `setText` auf das JLabel `lAusgabe` für die Ausgabe des Geburtsdatums geschrieben.
Wenn die erste Bedingung nicht erfüllt ist, prüfen wir, ob die Ergebnistabelle aus genau einer Zeile besteht (rote Bedingung). In diesem Fall prüfen wir mit der blauen Bedingung, ob diese Zeile im ersten Feld den Eintrag „Fehler“ enthält. In diesem Fall teilen wir das den Anwendenden mit. (Wenn wir unsere SQL-Anfrage richtig erstellt haben, sollte dieser Fall natürlich nicht auftreten.) Andernfalls wurde die Anfrage zwar korrekt ausgeführt, es wurde aber kein passender Datensatz gefunden, so dass wir den Anwendenden über das JLabel `lAusgabe` mitteilen, dass es die gesuchte Person in der Datenbank nicht gibt. Wenn wir davon ausgehen, dass die Ergebnis-Reihung immer mindestens eine Zeile enthält, können wir uns das Prüfen der roten Bedingung natürlich auch sparen.

Bei vielen Anfragen wird das Ergebnis aus mehreren Spalten und Zeilen bestehen, so dass die Ausgabe unter Umständen zu lang für ein einzeiliges *JLabel* ist. Alternativ bietet sich dann die Ausgabe in einer *JTextArea* an. Diese wird genauso wie ein *JLabel* oder *JTextField* erstellt, kann aber im Unterschied zum Textfeld mehrere Textzeilen anzeigen. Damit die Anwendenden die Ausgabe nicht verändern können, kann das Attribut *Editable* auf `false` gesetzt werden. Ein Beispiel befindet sich im Anhang: *Ergebnispräsentation in einer JTextArea*, S.22.

Anhang

Die Bedienelemente aus dem Paket *javax.swing* lassen sich sowohl in Java-Applets als auch in *JFrames* verwenden. Die Definition der Komponenten unterscheidet sich dabei nicht. Auch wenn die Abbildungen hier die Verwendung in Java-Applets zeigen, können die Komponenten daher genauso auch in *JFrames* eingebunden werden.

Erzeugen und Auswerten von Dialogelementen für die Benutzereingabe

Im Folgenden werden zunächst *JRadioButtons*, *JComboBoxen* und *JCheckBoxen* vorgestellt. Diese Dialogelemente eignen sich vor allem für Angaben, bei denen die Antwortmöglichkeiten begrenzt sind und die als Text bzw. in Worten ausgedrückt werden können. Benötigt man hingegen Angaben, die sich durch Zahlen ausdrücken lassen, wie z. B. ein Datum oder eine Zensur, können ein *JSlider* oder ein *JSpinner* verwendet werden. Diese werden im Anschluss vorgestellt. Mithilfe dieser Dialogelemente lässt sich der Wertebereich der Antwortmöglichkeiten einschränken, so dass keine ungültigen Antworten gegeben werden können. Beispielsweise lässt sich damit der Notenbereich auf 0 bis 15 Punkte eingrenzen.

JRadioButtons

JRadioButtons eignen sich immer dann, wenn es nur wenige gültige Antwortmöglichkeiten gibt, von denen nur eine ausgewählt werden darf. Dies wäre z. B. bei dem Geschlecht der Fall. Unsere Schul-Datenbank enthält nur Schülerinnen und Schüler, die sich dem Geschlecht männlich oder weiblich zuordnen. Ein weiterer Radiobutton „divers“ kann bei Bedarf ergänzt werden.

Abbildung 5: Auswahl über Radio-Buttons

Erzeugen eines JRadioButtons

Den *JRadioButton* finden wir im *Java-Editor* unter *Swing 1*. Die Beschriftung des *JRadioButtons* wird im Objektinspektor unter *Text* eingetragen. Da wir eventuell mehrere Informationen mit *JRadioButtons* erfragen, so dass im gesamten Applet mehr als ein *JRadio-Button* ausgewählt sein darf, benötigen wir noch eine *ButtonGroup*, der alle *JRadioButtons* zugeordnet werden, die zu einer Frage gehören und von denen somit nur einer ausgewählt sein darf. In unserem Fall müssen also die beiden *JRadioButtons* für männlich und weiblich einer *ButtonGroup* zugeordnet werden. Die *ButtonGroup* finden wir ebenfalls unter *Swing 1* neben dem *JRadioButton*. Im Beispiel wurde sie `buttonGroupGeschlecht` genannt. Nachdem wir sie erstellt haben, kann sie im Objektinspektor bei *ButtonGroup* ausgewählt werden. In Abbildung 6 sehen wir, dass in Zeile 30 der entsprechende Quelltext erzeugt wird, der den *JRadioButton* der *ButtonGroup* zuordnet. Wo die *ButtonGroup* im Designformular positioniert wird, ist unwichtig, da sie bei der Ausführung der Anwendung nicht sichtbar ist. Bei dem Attribut `selected` kann im Objektinspektor mit dem Wert `true` festgelegt werden, dass ein Radiobutton beim Start der Anwendung vorausgewählt sein soll.

Möchten wir zusätzlich die sexuelle Orientierung erfragen, könnten wir weitere *JRadioButtons* für heterosexuell oder homosexuell erzeugen und eine weitere *ButtonGroup*, z. B. `buttonGroup-Orientierung` anlegen, der wir diese *JRadioButtons* zuordnen.

In Abbildung 6 sind alle wichtigen Stellen im *Java-Editor* umrandet. Der entsprechende Quelltext wird automatisch erzeugt.

Auswerten eines JRadioButtons

Wenn wir feststellen möchten, welcher *JRadioButton* ausgewählt wurde, verwenden wir die Methode `isSelected` der Klasse *JRadioButton*. Diese liefert den Wert `true`, wenn der *JRadio-Button* ausgewählt wurde und `false`, wenn der *JRadioButton* nicht ausgewählt wurde. Der folgende Quelltext würde in der Variablen `geschlecht` ein „m“ für männlich oder ein „w“ für weiblich speichern, je nachdem, welches Geschlecht ausgewählt wurde.

```
1  String geschlecht = "";
2  if(rbWeiblich.isSelected()) geschlecht = "w";
3  else geschlecht = "m";
```

Der *Java-Editor* stellt noch eine zweite Möglichkeit zur Verfügung, den ausgewählten *JRadioButton* bzw. dessen Beschriftung zu ermitteln. Beim Erzeugen einer *ButtonGroup* wird automatisch die Methode `buttonGroup1_getSelectedRadioButtonLabel()` erzeugt, wobei sich die Nummer (hier in rot) für jede neue *ButtonGroup* fortlaufend ändert. Diese Methode liefert die Beschriftung des ausgewählten *JRadioButtons* als Zeichenkette. Der folgende Quelltext würde also „weiblich“ bzw. „männlich“ in der Variablen `geschlecht` speichern, je nachdem welcher *JRadioButton* ausgewählt wurde:

```
String geschlecht = buttonGroup1_getSelectedRadioButtonLabel();
```

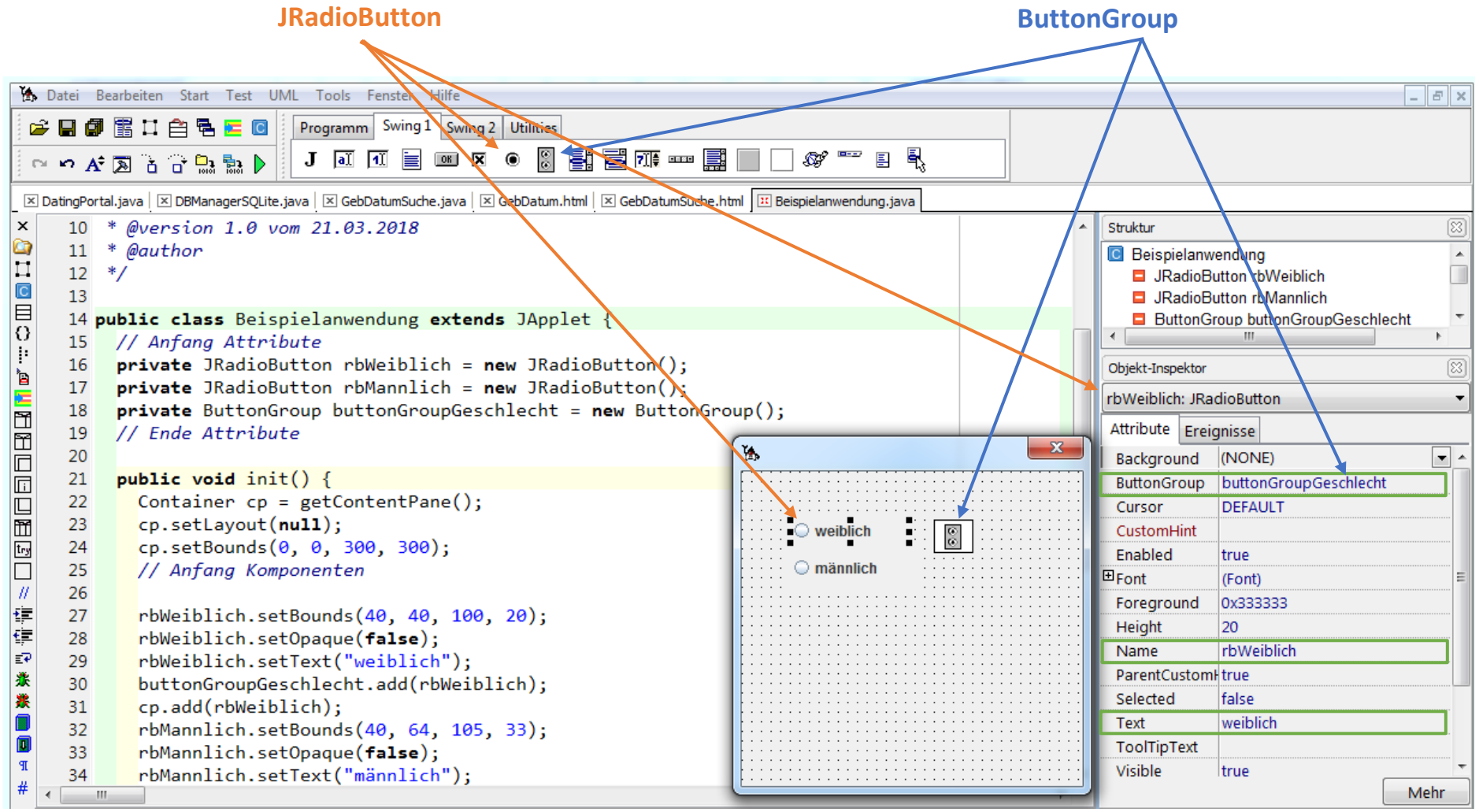


Abbildung 6: JRadioButtons im Java-Editor

JComboBox

Wenn es zu viele Antwortmöglichkeiten gibt, um sie alle auf der Benutzeroberfläche anzuzeigen, bietet sich die Präsentation der Antwortmöglichkeiten in einer *JComboBox* an. Diese stellt die Antwortmöglichkeiten in einer Dropdown-Liste dar, die sich nur dann ausklappt und die Antwortmöglichkeiten anzeigt, wenn der Pfeil an der Seite angeklickt wird. Wie Abbildung 7 zeigt, bietet sich eine *JComboBox* z. B. für die Auswahl des Lieblingsfaches an.

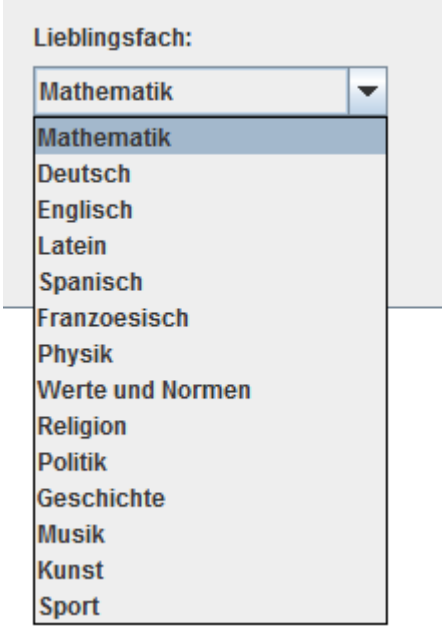


Abbildung 7: Auswahl über eine JComboBox

Erzeugen einer JComboBox

Die *JComboBox* befindet sich ebenfalls unter *Swing 1*. Die Einträge der *JComboBox* können im Objektinspektor unter *Items* eingegeben werden. Wird das Eingabefeld hinter *Items* angeklickt, öffnet sich ein Fenster. Hier schreiben wir jede Antwortmöglichkeit in eine neue Zeile.

Bei *MaximumRowCount* kann angegeben werden, wie viele Antwortmöglichkeiten beim Ausklappen sichtbar sind. Die restlichen Antwortmöglichkeiten erreicht man dann durch einen Schiebebalken.

In Abbildung 8 sind die wichtigen Stellen im *Java-Editor* umrandet. Die *JComboBox* wurde hier `jComboBoxFach` genannt.

Auswerten einer JComboBox

Um herauszufinden, welcher Wert in der *JComboBox* ausgewählt wurde, stellt die Klasse *JComboBox* die Methode `getSelectedItem` zur Verfügung. Der Rückgabewert dieser Methode ist jedoch vom Typ *Object*. Deshalb muss noch die Methode `toString` dahinter gehängt werden, um die Auswahl als Zeichenkette zu erhalten. Hat der Anwender beispielsweise das Fach *Politik* in der *JComboBox* ausgewählt, würde der folgende Quelltext die Zeichenkette „Politik“ in der Variablen `lieblingsfach` speichern:

```
String lieblingsfach = jComboBoxFach.getSelectedItem().toString();
```

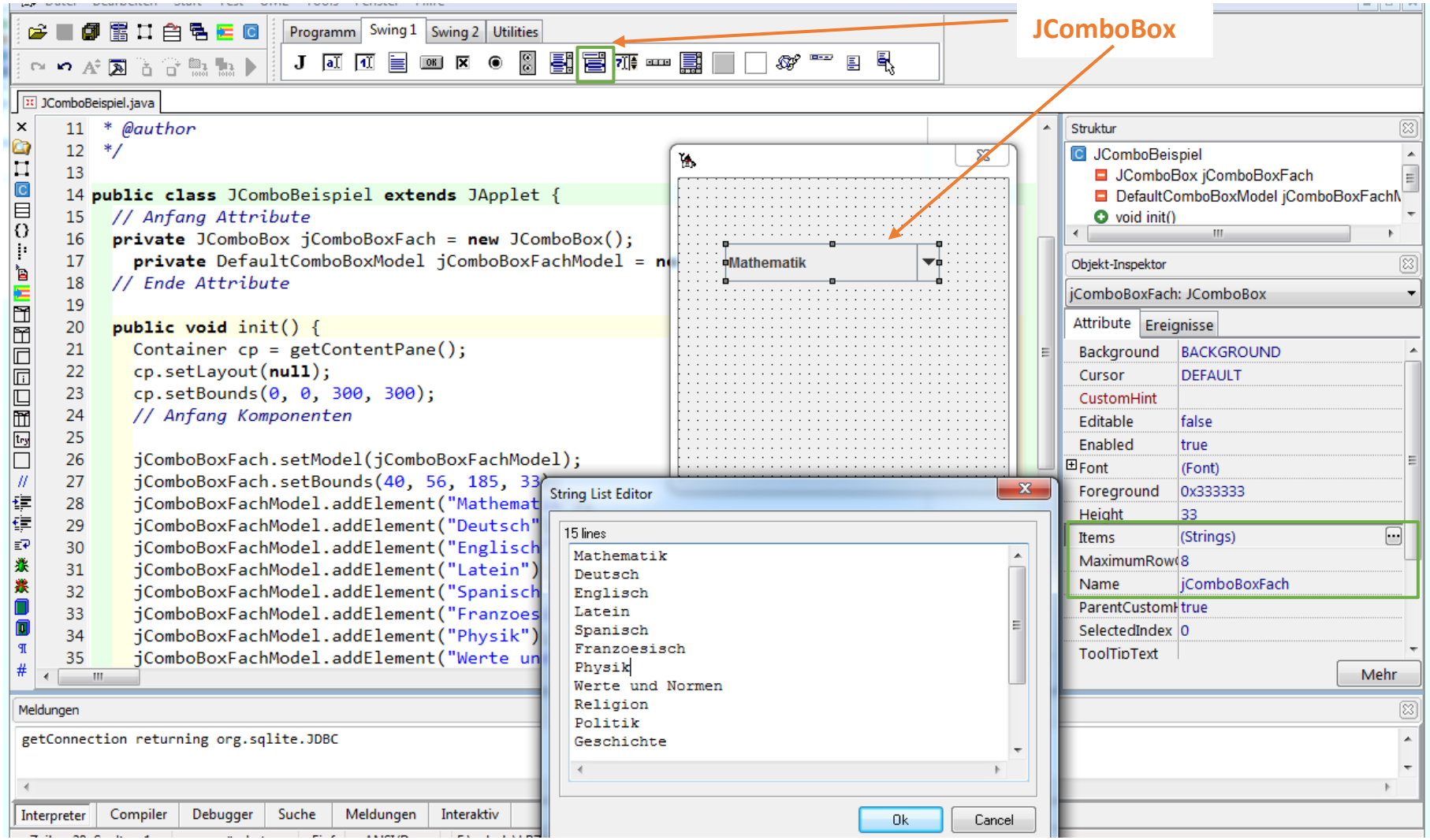


Abbildung 8: JComboBox im Java-Editor

JCheckBox

Eine *JCheckBox* eignet sich immer dann, wenn mehrere Antwortmöglichkeiten ausgewählt werden dürfen. Denn anders als bei den *JRadioButtons* können hier mehrere Häkchen gleichzeitig gesetzt werden. Ein Beispiel wäre die Frage „Welche Musik hörst du?“ Viele Anwendende mögen sicherlich mehr als eine Musikrichtung, so dass sie hier die Möglichkeit haben sollten, mehrere Musikrichtungen auszuwählen.

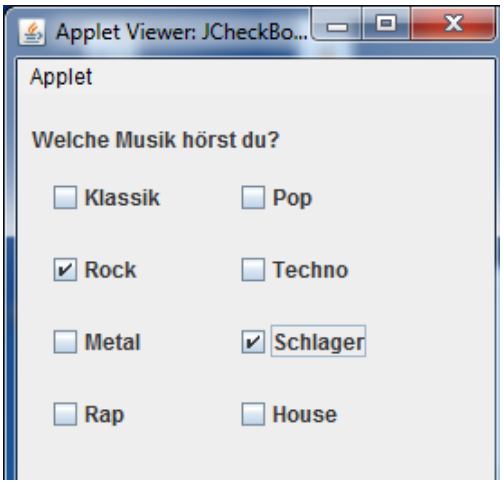


Abbildung 9: Auswahl über Check-Boxen

Eine JCheckBox erzeugen.

Die *JCheckBox* finden wir im *Java-Editor* unter *Swing 1* neben dem *JRadioButton*. Auch für die *JCheckBox* geben wir die Beschriftung im Objektinspektor unter *Text* ein. Da die *JCheckBoxen* unabhängig voneinander ausgewählt werden dürfen, müssen wir sie jedoch keiner Gruppe zuordnen. Bei *Selected* kann im Objektinspektor noch angegeben werden, ob die *JCheckBox* beim Starten der Anwendung bereits ausgewählt sein soll oder nicht. Für den Eintrag `true` wäre sie ausgewählt, für `false` entsprechend nicht ausgewählt.

In Abbildung 10 sind alle wichtigen Stellen im *Java-Editor* umrandet. Der Quelltext wird auch hier automatisch erzeugt.

JCheckBox auswerten

Um zu ermitteln, ob eine *JCheckBox* angehakt wurde, stellt die Klasse *JCheckBox* die Methode `isSelected` zur Verfügung. Da die *JCheckBoxen* unabhängig voneinander ausgewählt werden können, muss diese Methode für jede einzelne *JCheckBox* aufgerufen werden. Die Methode liefert `true` zurück, wenn die *JCheckBox* ausgewählt wurde, andernfalls `false`. Der folgende Quelltext würde in der booleschen Variablen `rock` den Wert `true` speichern, wenn das Häkchen bei *Rock* gesetzt wurde und ansonsten `false`.

```
boolean rock = cbRock.isSelected();
```

Die Klasse *JCheckBox* stellt auch die Methode `getText` zur Verfügung, die unabhängig davon, ob die *JCheckBox* ausgewählt wurde, die Beschriftung der *JCheckBox* als Zeichenkette zurückgibt.

JCheckBox

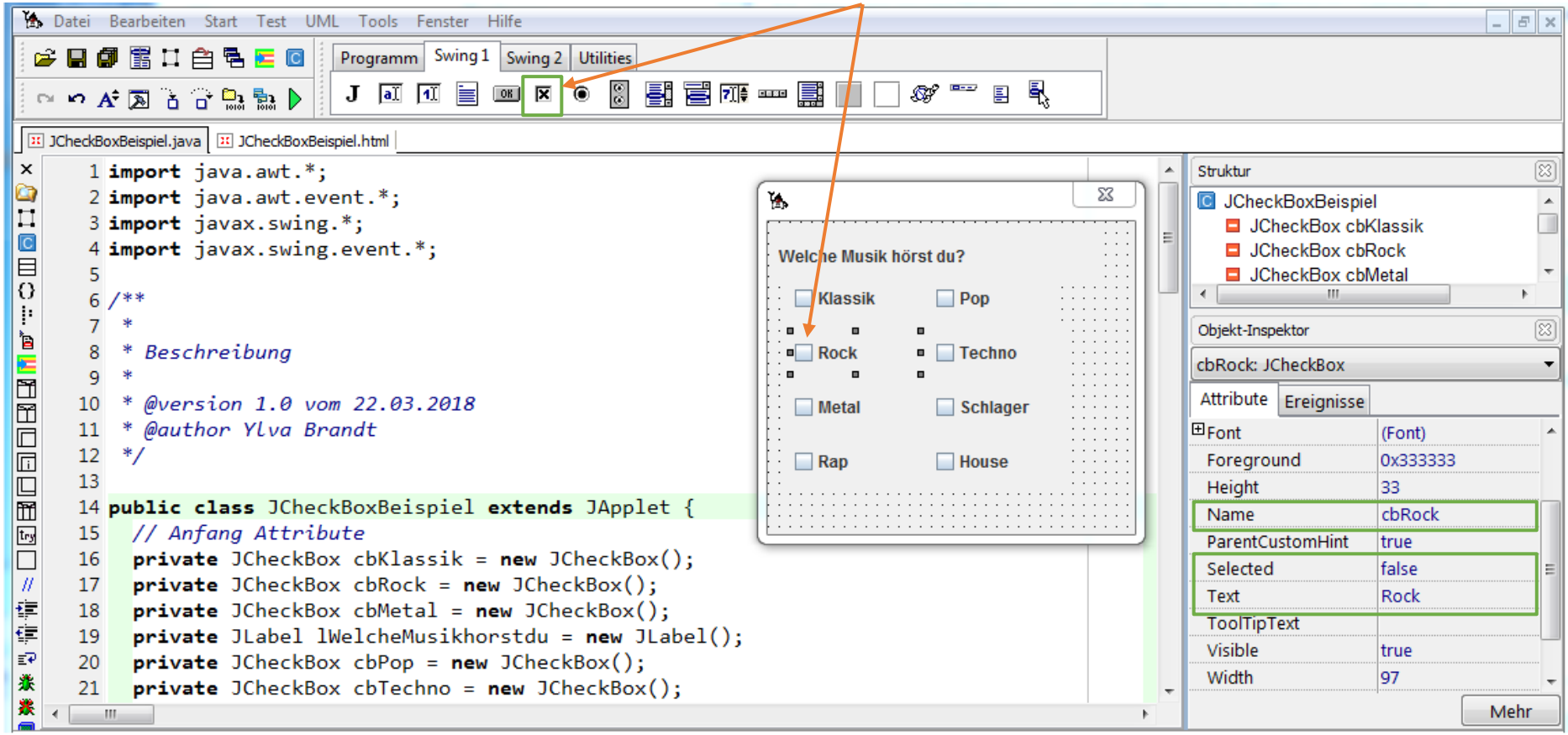


Abbildung 10: JCheckBox im Java-Editor

JSlider

Ein *JSlider* ist eine Art Schieberegler mit dem ein ganzzahliger Wert im angebotenen Bereich ausgewählt werden kann. Da nur ganzzahlige Werte zur Verfügung stehen, müsste man z. B. bei der Angabe der Körpergröße die Einheit cm verwenden und den Wert im Programm bei Bedarf in Meter umrechnen. Das schauen wir uns hier im Beispiel an.

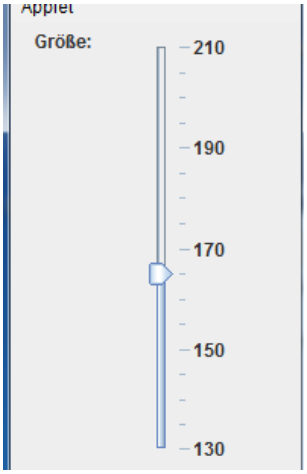


Abbildung 11: Auswahl über JSlider

Erzeugen eines JSliders

Der *JSlider* befindet sich im Bereich *Swing 2*. Für den *JSlider* können einige interessante Einstellungen im Objektinspektor vorgenommen werden. Die entsprechenden Eigenschaften des *JSliders* werden daher in der folgenden Tabelle erläutert:

Attribut im Objektinspektor	Bedeutung
Inverted	Bei der Angabe `false` steht der kleinste Wert links und der größte rechts. Bei der Angabe `true` steht der kleinste Wert rechts und der größte links.
MajorTickSpacing	Hier wird der Abstand der langen Markierungsstriche angegeben. Wenn bei *PaintLabels* `true` angegeben wird, werden diese Striche auch mit den entsprechenden Werten beschriftet.
Maximum	Der größte Wert, der eingestellt werden kann.
Minimum	Der kleinste Wert, der eingestellt werden kann.
MinorTickSpacing	Hier wird der Abstand der kleinen Markierungsstriche angegeben. Diese werden nicht beschriftet.
Name	Der Name der Variablen für den *JSlider*.
Orientation	Hier kann zwischen `HORIZONTAL` und `VERTICAL` gewählt werden. Bei `HORIZONTAL` verläuft der Schieberegler von links nach rechts, bei `VERTICAL` von unten nach oben.
PaintLabels	Bei `true` werden alle langen Markierungsstriche mit den entsprechenden Werten beschriftet. Bei `false` entfällt die Beschriftung mit Werten vollständig.
PaintTicks	Bei `true` werden die Markierungsstriche gezeichnet, bei `false` entfallen sowohl die langen als auch die kurzen Markierungsstriche.
PaintTrack	Bei `true` wird eine Art Schiene gezeichnet, auf der sich der Regler bewegt, bei `false` wird diese nicht gezeichnet.
SnapToTicks	Bei `true` rastet der Regler immer bei einer Markierung ein, bei `false` können auch ganzzahlige Werte zwischen den Markierungen gewählt werden. Wurde bei *MinorTickSpacing* z. B. 10 angegeben, so können bei `true` nur die Werte 10, 20, 30, 40 usw. ausgewählt werden. Bei `false` könnte der Regler auch auf den Werten 11, 12, 13, 14 usw. stehen.
Value	Der Wert, der zu Beginn ausgewählt ist.

In Abbildung 12 sind die wichtigen Stellen im *Java-Editor* umrandet. Der *JSlider* wurde hier `jSliderGroesse` genannt.

Hinweis: Der Quellcode für die Eigenschaften, die im Objektinspektor eingetragen werden, wird zwar automatisch erstellt. Damit der voreingestellte Wert im Applet richtig angezeigt werden kann, muss der entsprechende Methodenaufruf `setValue` aber nach den Methodenaufrufen `setMinimum` und `setMaximum` erfolgen. Die entsprechenden Codezeilen müssen daher ggf. richtig sortiert werden.

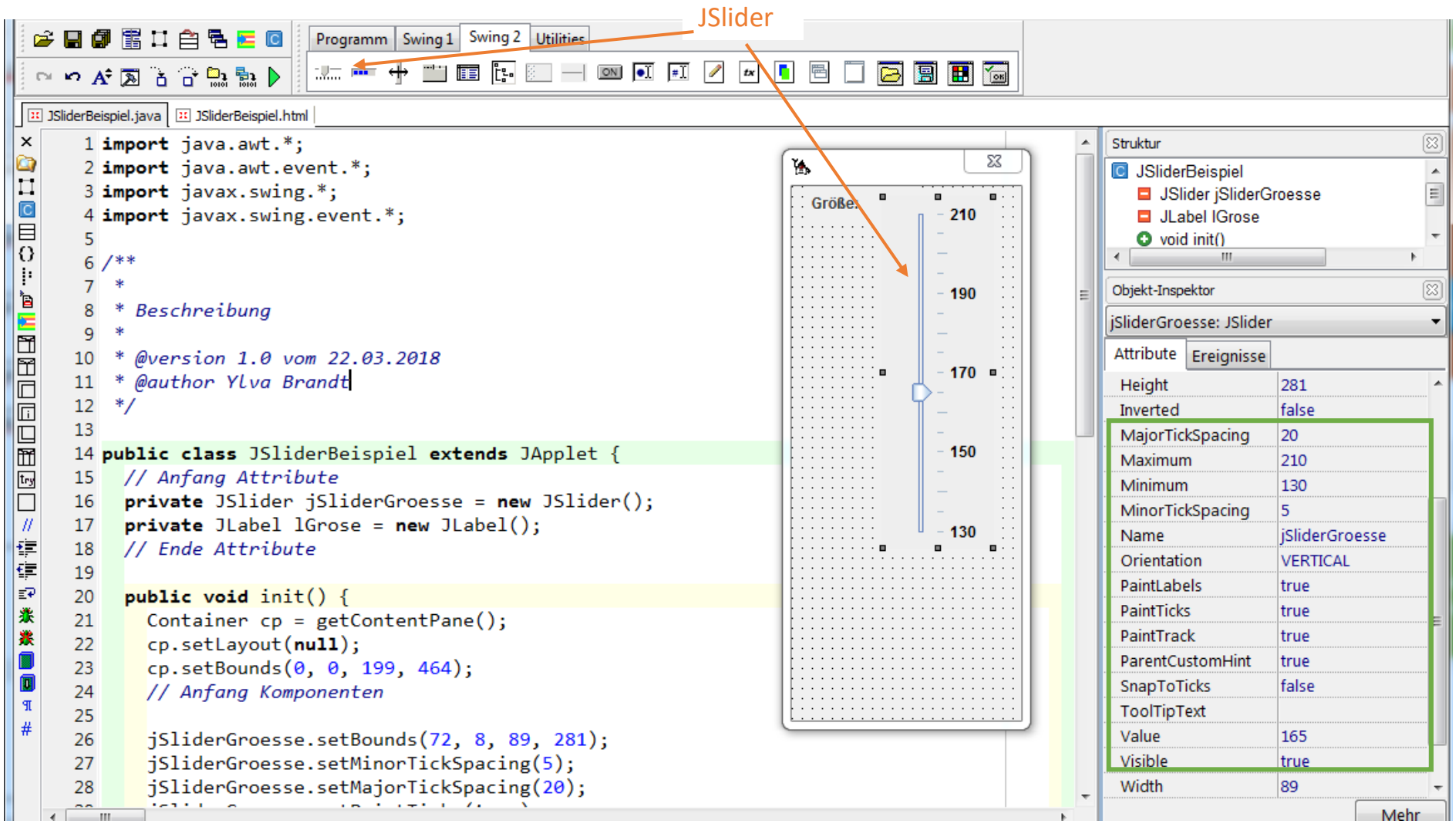
Auswerten des JSliders

Um zu ermitteln, welchen Wert der Anwender am *JSlider* eingestellt hat, steht in der Klasse *JSlider* die Methode `getValue` zur Verfügung, die den eingestellten Wert als *Ganzzahl* zurückgibt. Der folgende Quelltext würde also die eingestellte Größe in der Ganzzahlvariablen `groesse` speichern.

```
int groesse = jSliderGroesse.getValue();
```

Möchte man die Größe in Meter umwandeln, müsste man den Wert in einer Variablen vom Typ *float* speichern und durch 100 teilen:

```
float fgroesse = (float)groesse / 100;
```




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5
6  /**
7   *
8   * Beschreibung
9   *
10  * @version 1.0 vom 22.03.2018
11  * @author Ylva Brandt
12  */
13
14 public class JSliderBeispiel extends JApplet {
15   // Anfang Attribute
16   private JSlider jSliderGroesse = new JSlider();
17   private JLabel lGrose = new JLabel();
18   // Ende Attribute
19
20   public void init() {
21     Container cp = getContentPane();
22     cp.setLayout(null);
23     cp.setBounds(0, 0, 199, 464);
24     // Anfang Komponenten
25
26     jSliderGroesse.setBounds(72, 8, 89, 281);
27     jSliderGroesse.setMinorTickSpacing(5);
28     jSliderGroesse.setMajorTickSpacing(20);
```

Abbildung 12: JSlider im Java-Editor

JSpinner

Der *JSpinner* verfügt über zwei Pfeiltasten, mit denen man sich vorwärts und rückwärts durch eine Auswahl von Werten klicken kann. Der jeweils sichtbare Wert gilt als ausgewählt.

Abbildung 13: Auswahl über einen JSpinner

Erzeugen eines JSpinners

Der *JSpinner* befindet sich im Bereich *Swing 1*. Es gibt zwei Möglichkeiten, wie die Werte, die ausgewählt werden können, mit dem Objektinspektor festgelegt werden können. Betrachten wir ein Beispiel, bei dem ein *JSpinner* verwendet wird, um dem Anwender die Möglichkeit zu geben, ein Geburtsjahr auszuwählen. Wir gehen davon aus, dass nur Angaben zwischen 1997 und 2002 zulässig sind. Dazu geben wir im Objektinspektor bei *Maximum* den Wert 2002 und bei *Minimum* den Wert 1997 ein. Weiterhin muss bei *StepSize* angegeben werden, wie weit die Werte, die ausgewählt werden können, auseinanderliegen. Da als Geburtsjahr jede Zahl zwischen 1997 und 2002 in Frage kommt, geben wir hier 1 an. Würden wir hier z. B. 2 angeben, so enthielte die Auswahl nur die Zahlen 1997, 1999 und 2001. Zuletzt können wir bei *Value* noch den Wert angeben, der zu Beginn angezeigt werden soll.

In Abbildung 14 sind die wichtigen Stellen im *Java-Editor* umrandet. Der *JSpinner* wurde hier `jSpinnerJahr` genannt.

Da das Jahr hier als Zahl interpretiert wird, steht nach der Tausenderstelle für bessere Lesbarkeit ein Punkt. Dies ist bei der Schreibweise für Jahreszahlen eher unüblich. Wen das stört, kann die zweite Möglichkeit zur Festlegung der Werte nutzen. Wie bei der *JComboBox* kann auch hier bei *List* eine Liste von Zeichenketten angegeben werden. Klickt man auf die drei Punkte rechts im Eingabefeld hinter *List* öffnet sich ein kleines Fenster, in das die Werte untereinander eingetragen werden können. Da es sich hierbei um Zeichenketten handelt, können hier nicht nur Zahlwerte, sondern auch Wörter eingegeben werden. So ließe sich z. B. auch ein *JSpinner* für die Auswahl eines Schulfaches erstellen. Abbildung 15 und Abbildung 16 zeigen diese Möglichkeiten. Wenn eine Liste mit Werten angegeben wird, werden die Einträge bei *Minimum*, *Maximum* und *StepSize* ignoriert.

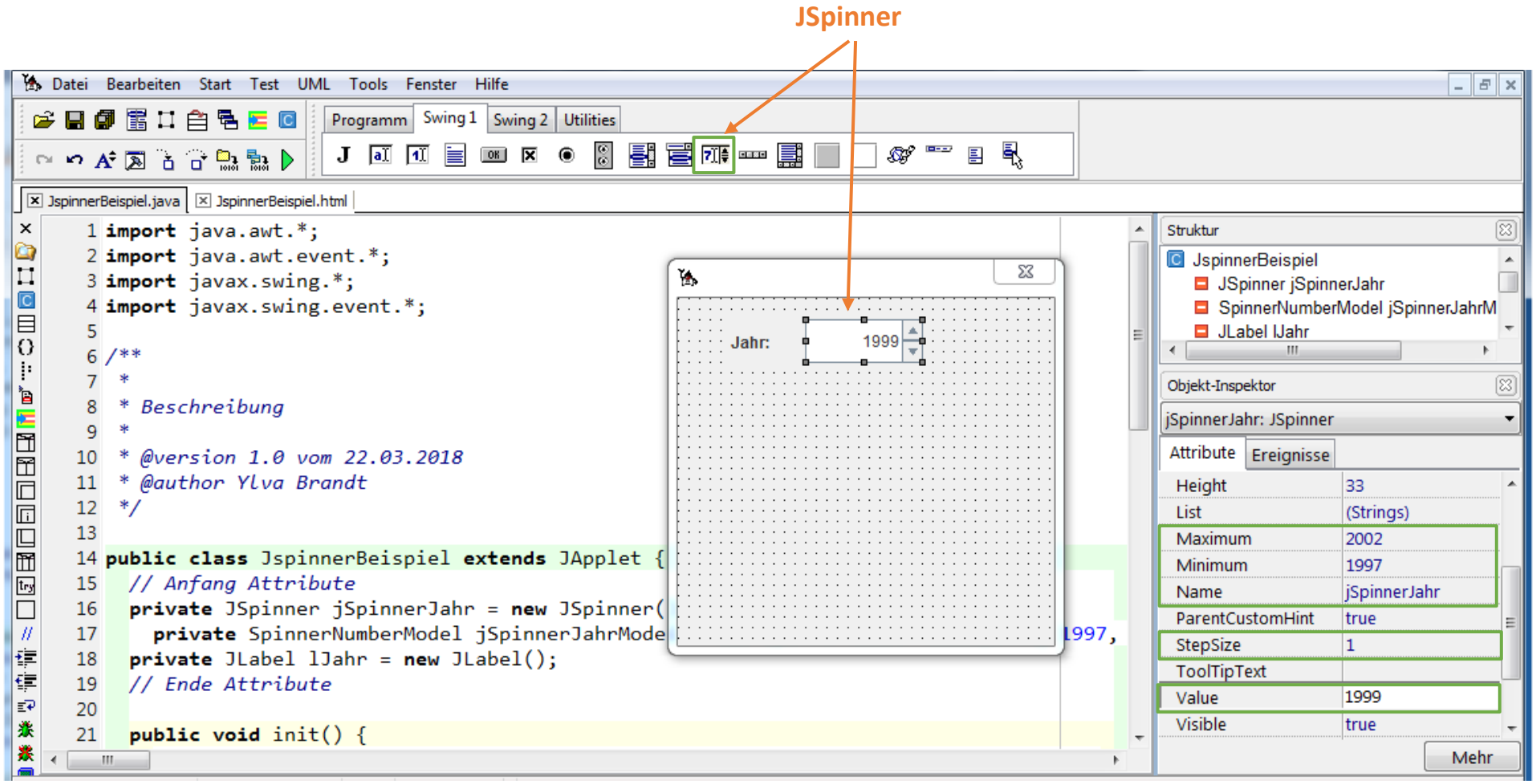
Auswerten eines JSpinners

Um den Wert zu ermitteln, der in einem *JSpinner* ausgewählt wurde, stellt die Klasse *JSpinner* die Methode `getValue` zur Verfügung. Da ein *JSpinner* sowohl Zahlen als auch Zeichenketten enthalten kann, ist der Rückgabewert der Methode vom Typ *Object*. Es muss daher ein `toString` angehängt werden, um den ausgewählten Wert als Zeichenkette zu erhalten.

```
String jahr = jSpinnerJahr.getValue().toString();
```

In der Regel ist es auch bei Zahlwerten ausreichend, diese als Zeichenkette auszulesen, da sie beim Zusammensetzen der *SQL*-Abfrage ohnehin in Zeichenketten umgewandelt würden. Benötigt man den Zahlwert jedoch tatsächlich als Zahl, weil man den Wert beispielsweise noch durch Rechenoperationen verändern möchte, müsste man die Zeichenkette entsprechend Parsen. Für eine *Ganzzahl* sähe das so aus:

```
int jahreszahl = Integer.parseInt(jSpinnerJahr.getValue().toString());
```




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.event.*;
5
6  /**
7   *
8   * Beschreibung
9   *
10  * @version 1.0 vom 22.03.2018
11  * @author Ylva Brandt
12  */
13
14 public class JspinnerBeispiel extends JApplet {
15   // Anfang Attribute
16   private JSpinner jSpinnerJahr = new JSpinner(
17     private SpinnerNumberModel jSpinnerJahrMode                1997,
18   private JLabel lJahr = new JLabel();
19   // Ende Attribute
20
21   public void init() {
```

Abbildung 14: JSpinner im Java-Editor

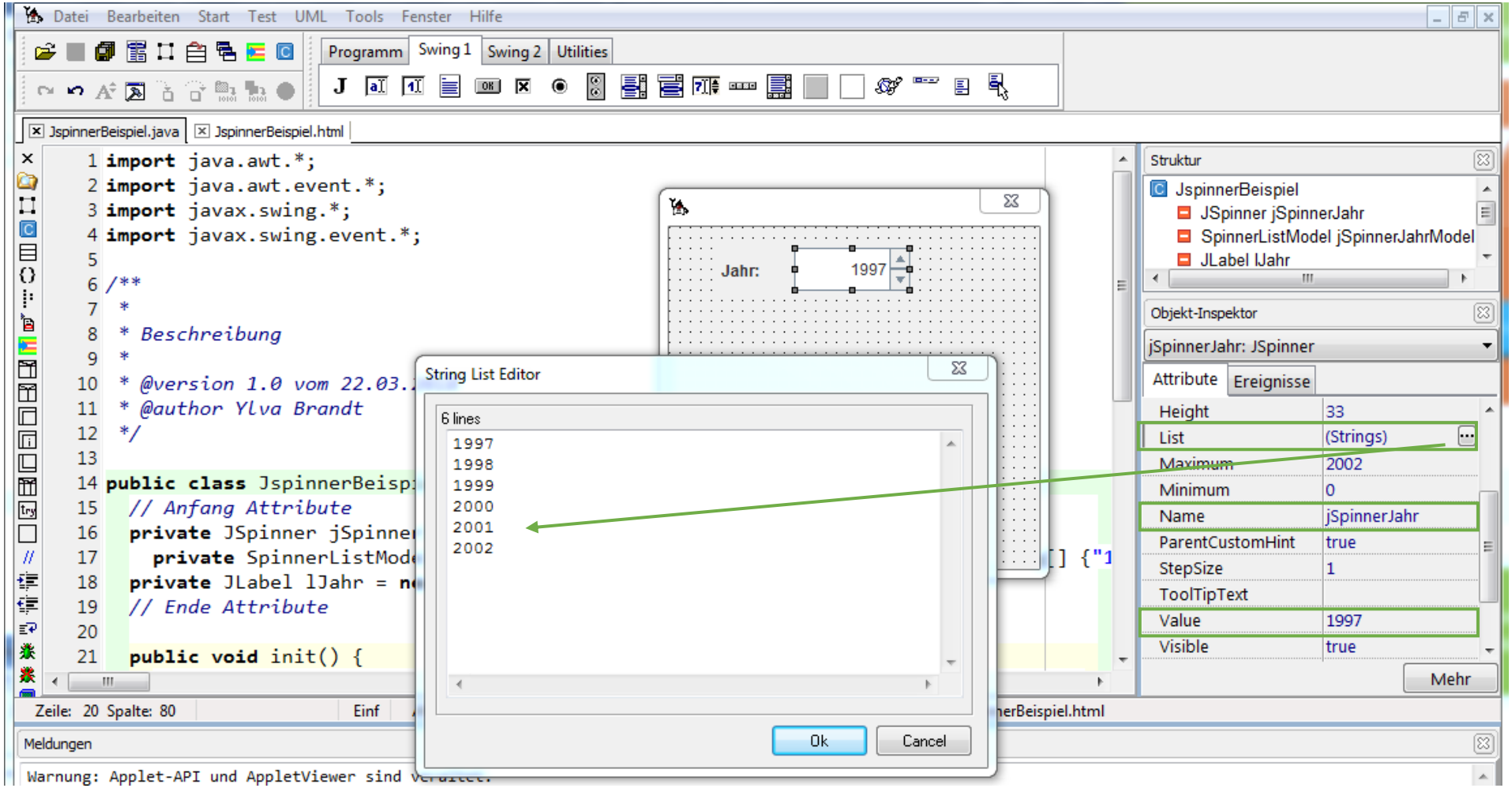


Abbildung 15: JSpinner mit einer Liste von Zahlwerten

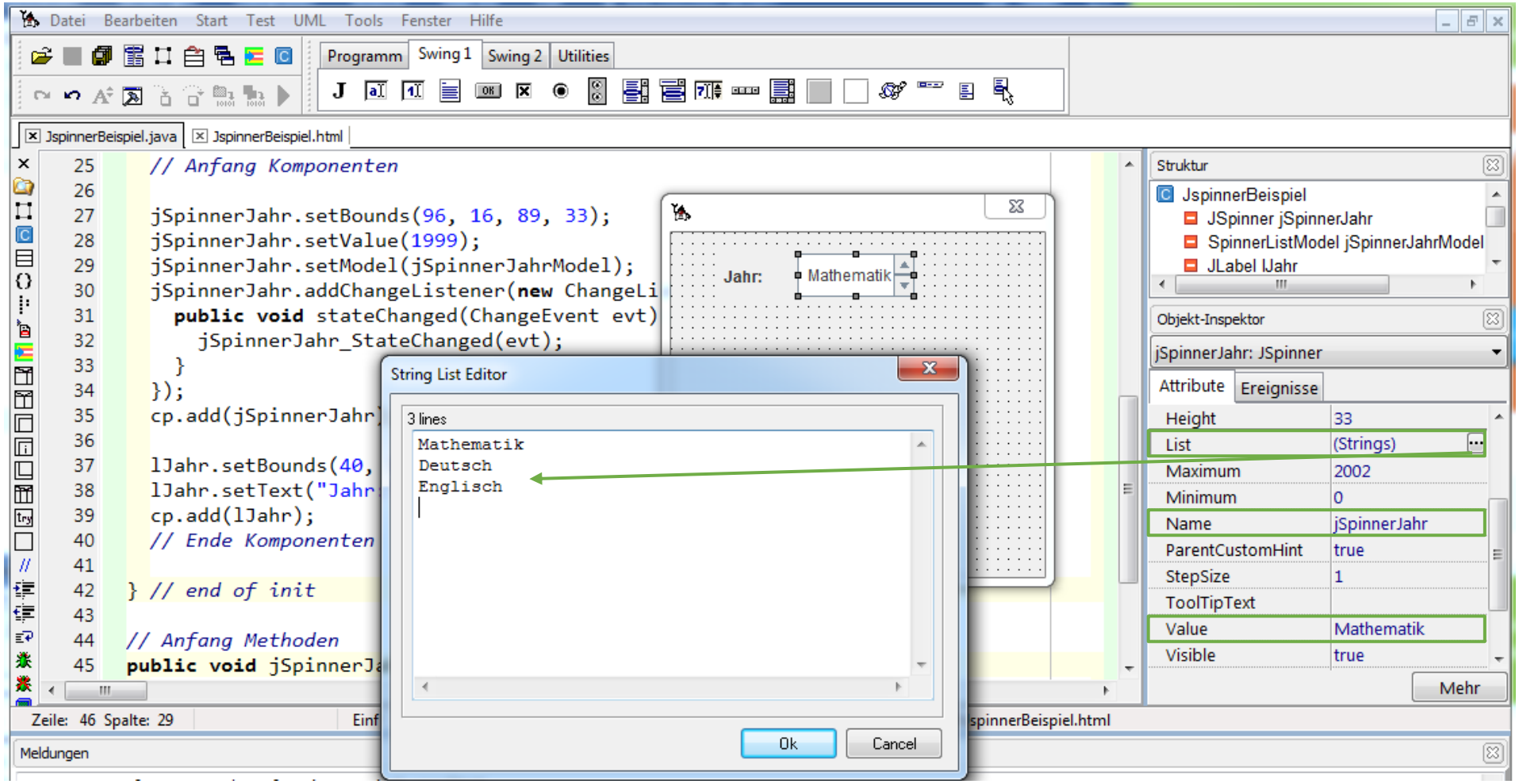


Abbildung 16: JSpinner mit einer Liste von Wörtern

Ergebnispräsentation in einer JTextArea

Die *JTextArea* kann im Unterschied zu einem *JTextField* mehrere Zeilen Text anzeigen (s. Abbildung 17). Das Erzeugen einer *JTextArea* funktioniert wie bei einem *JTextField*. Damit in der *TextArea* wie in Abbildung 17 ggf. gescrollt werden kann, falls die Größe für die Anzeige des gesamten Textes nicht ausreicht, muss zunächst eine *JScrollPane* erstellt werden. In die *JScrollPane* kann dann eine *JTextArea* gezogen werden. Wir finden die entsprechenden Dialogelemente ebenfalls bei *Swing 1*.

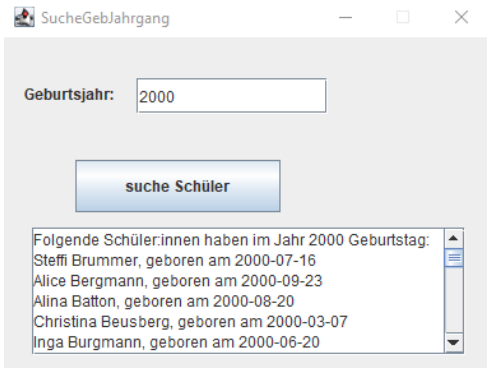


Abbildung 17: Ausgabe in einer JTextArea

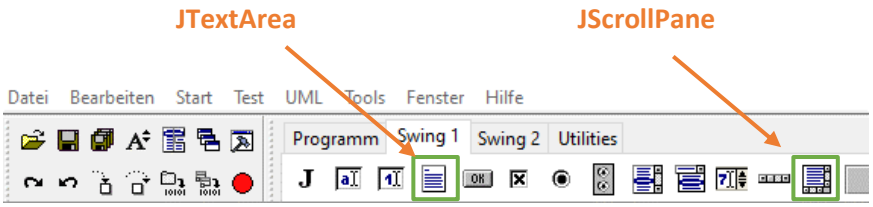


Abbildung 18: JTextArea und JScrollPane im JavaEditor

Die Zuweisung des Textes erfolgt wie beim *JTextField* mit der Methode `setText`. Alternativ kann mit der Methode `append` weiterer Text an den bisherigen Inhalt der *TextArea* angehängt werden. Zeilenumbrüche erfolgen dabei nicht automatisch, sondern müssen ggf. ergänzt werden. Wie wir den Inhalt unserer zweidimensionalen Reihung `ergebnis` in eine *TextArea* übertragen, schauen wir uns im Folgenden an einem Beispiel an.

Die Anwendung in Abbildung 17 sucht zu einem Geburtsjahr alle Schülerinnen und Schüler aus der Datenbank heraus, die in diesem Jahr geboren wurden. Das Textfeld, in welches das Geburtsjahr eingegeben wird, hat den Variablennamen `tfJahr`. Wir erzeugen zunächst die passende *SQL-Anfrage* und senden diese an die Datenbank:

```
1  String jahr = tfJahr.getText();
2             String sql = "SELECT Vorname, Name, Geburtstag FROM schueler
              where Geburtstag like '" + jahr + "%' ;";
3  String[][] ergebnis = myDBManager.sqlAnfrageAusfuehren(sql);
```

Bei erfolgreicher Suche enthält die Variable `ergebnis` nun eine Tabelle, die in der ersten Spalte die Vornamen, in der zweiten die Nachnamen und in der dritten die Geburtstage der passenden Schülerinnen und Schüler enthält. Mit folgendem Quellcode können wir daher die Ausgabe wie in Abbildung 17 angezeigt erzeugen:

```
1  taAusgabe.setText("Folgende Schüler:innen haben im Jahr " + jahr + " Geb
     urtstag:" +"\n");
2  if(ergebnis.length > 1){
3    for(int i = 1; i < ergebnis.length;i++)  {
4       taAusgabe.append(ergebnis[i][0] + " " + ergebnis[i][1] + ", geboren
          am " + ergebnis[i][2]+"\n");
5    }
6  }else{
7     if(ergebnis[0][0].equals("Fehler")) taAusgabe.setText("Es ist ein
        Fehler aufgetreten.");
8     else taAusgabe.setText("Es gibt keine Schüler, die im Jahr " + jahr +
        " geboren wurden.");
9  }
```

In Zeile 1 überschreiben wir den bisherigen Inhalt der TextArea `taAusgabe` mit der ersten Zeile für die Ausgabe. Die Jahreszahl fügen wir dabei passend mithilfe der Variablen `jahr` ein. Am Ende steht „\n“. Damit erzeugen wir einen Zeilenumbruch.

In Zeile 2 testen wir, ob passende Datensätze gefunden wurden. Das ist der Fall, wenn die zweidimensionale Reihung `ergebnis` mehr als eine Zeile enthält. In Zeile 3 bis 5 wird die zweidimensionale Reihung dann zeilenweise durchlaufen. Für jede Zeile der zweidimensionalen Reihung, die einen Datensatz enthält, wird eine Zeile an den Inhalt der TextArea `taAusgabe` angehängt. Als Zeilenindex geben wir den aktuellen Wert des Schleifenzählers `i` an. Als Spaltenindex geben wir 0 für den Vornamen, 1 für den Nachnamen und 2 für den Geburtstag an, da wir die Attribute in der *SQL-Anweisung* in dieser Reihenfolge angefordert haben. Am Ende fügen wir wieder „\n“ an, damit jede Person in einer neuen Textzeile steht.

In Zeile 7 und 8 unterscheiden wir noch die Fälle, dass die *SQL-Anfrage* einen Fehler ausgelöst hat (Zeile 7) bzw. dass kein Schüler und keine Schülerin zu dem angegebenen Geburtsjahr gefunden wurde (Zeile 8). Der Inhalt der TextArea `taAusgabe` wird dann jeweils mit dem passenden Text überschrieben.

Den gesamten Quelltext zu dem Beispiel finden Sie in der Datei *sucheGebJahrgang.java* im Ordner *Beispiele → Java*.

Umwandeln von Zeichenketten in Zahlen

Es kann an verschiedenen Stellen das Problem auftreten, dass eine Zahl als Zeichenkette vorliegt, wir diese aber in einer Variablen vom Typ `int` oder `float` speichern möchten, um sie z. B. durch Rechenoperationen zu verändern. Das Problem kann z. B. auftreten, wenn eine Zahl in ein Textfeld eingegeben wurde. Auch im Ergebnis unserer Datenbankanfrage liegen die Zahlen als Zeichenketten vor. Um den Datentyp von `String` in `int` oder in `float` zu ändern, stehen die Methoden `Integer.parseInt` bzw. `Float.parseFloat` zur Verfügung, die als Parameter eine Ganzzahl bzw. eine Fließkommazahl als Zeichenkette erwarten und die Zahl in dem entsprechenden Datentyp zurückliefern. Handelt es sich bei der Zeichenkette nicht um eine entsprechende Zahl, wird eine *NumberFormatException* ausgelöst. Ein Beispiel ist im Abschnitt *Auswerten eines JSpinners* auf Seite 18 zu finden.

Die Klasse DBManagerSQLite

Die Klasse *DBManagerSQLite* übernimmt die Kommunikation mit der Datenbank und erleichtert so das Erstellen eine Datenbankanwendung. Wie der Name bereits verrät, unterstützt die Klasse nur die Kommunikation mit einer SQLite-Datenbank. Für die Kommunikation mit anderen Datenbanksystemen müsste sie entsprechend angepasst werden. Die Klasse stellt folgende Konstruktoren und Methoden zur Verfügung.

Konstruktoren

Konstruktor	Beschreibung
`DBManagerSQLite()`	erzeugt ein DBManagerSQLite-Objekt für die Datenbank schule_erweitert und stellt eine Verbindung zur Datenbank her
`DBManagerSQLite(String dbName)`	erzeugt ein DBManagerSQLite-Objekt für die angegebene Datenbank und stellt eine Verbindung zur Datenbank her

Übersicht Methoden

Rückgabewert	Methode	Beschreibung
`int`	`datensatzAendern(String sql)`	führt die übergebene SQL-Update-Anweisung aus
`int`	`datensatzEinfuegen(String sql)`	führt die übergebene SQL-Einfüge-Anweisung aus
`int`	`datensatzLoeschen(String sql)`	führt die übergebene SQL-Lösch-Anweisung aus
`String[][]`	`sqlAnfrageAusfuehren(String sqlAnfrage)`	führt die übergebene SQL-Anfrage aus

Die Methoden im Detail

datensatzAendern

```
public int datensatzAendern(String sql)
```

führt die übergebene SQL-Update-Anweisung aus

Parameter:
sql - Die SQL-Anweisung, die ausgeführt werden soll, als Zeichenkette

Rückgabewert:
Die Anzahl der betroffenen Datensätze. Der Rückgabewert ist 0, wenn die Anweisung keine Änderung in der Datenbank bewirkt hat. Der Rückgabewert -1 zeigt an, dass ein Fehler aufgetreten ist.

datensatzEinfuegen

`public int datensatzEinfuegen(String sql)`

führt die übergebene SQL-Einfüge-Anweisung aus

Parameter:

`sql` - Die SQL-Anweisung, die ausgeführt werden soll, als Zeichenkette.

Rückgabewert:

Die Anzahl der betroffenen Datensätze. Der Rückgabewert ist 0, wenn die Anweisung keine Änderung in der Datenbank bewirkt hat. Der Rückgabewert -1 zeigt an, dass ein Fehler aufgetreten ist.

datensatzLoeschen

`public int datensatzLoeschen(java.lang.String sql)`

führt die übergebene SQL-Lösch-Anweisung aus

Parameter:

sql - Die SQL-Anweisung, die ausgeführt werden soll, als Zeichenkette

Rückgabewert:

Die Anzahl der betroffenen Datensätze. Der Rückgabewert ist 0, wenn die Anweisung keine Änderung in der Datenbank bewirkt hat. Der Rückgabewert -1 zeigt an, dass ein Fehler aufgetreten ist.

sqlAnfrageAusfuehren

`public String[][] sqlAnfrageAusfuehren(String sqlAnfrage)`

führt die übergebene SQL-Anfrage aus

Parameter:

`sqlAnfrage` - Die SQL-Anfrage, die ausgeführt werden soll, als Zeichenkette.

Rückgabewert:

Das Ergebnis der SQL-Anfrage als zweidimensionale Reihung vom Typ Zeichenkette. Interpretiert man den ersten Index als Zeilen- und den zweiten als Spatennummer, enthält die erste Zeile der Reihung die Überschriften der Spalten. Danach folgt pro Datensatz eine Zeile mit den entsprechenden Werten. Diese werden unabhängig von den Datentypen der Datenbank als Zeichenkette gespeichert. Enthält eine Zelle in der Datenbank den Wert null, wird die Zeichenkette "null" in das entsprechende Feld der zweidimensionalen Reihung geschrieben.
Schlägt der Versuch, die SQL-Anfrage zu stellen, fehl, enthält die zweidimensionale Reihung nur ein Feld mit dem Inhalt "Fehler".