Erstellen einer Datenbankanwendung mit Processing

Die Anleitung bezieht sich auf Datenbankenanwendungen, 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 der Datenbank heraussucht. Sie können sich den Quellcode in dem Processing-Projekt *BspProcSucheGebDatum* aus dem Ordner *Beispiele* → *Processing* anschauen.

Vorbereitung

Um das Beispiel ausprobieren und eigene Datenbankanwendungen erstellen zu können, müssen in Processing zwei Bibliotheken und ein Tool installiert werden. Wählen Sie dazu den Menüpunkt *Sketch* → *Library importieren …* → *Manage Libraries…*

Installieren Sie hier die Bibliotheken *BezierSQLib* von Florian Jenett und *G4P* von Peter Lager sowie das zugehörige Tool *G4P GUI Builder* von Peter Lager.

Nun sollten Sie das Programm *BspProcSucheGebDatum* ausführen können.

Für eigene Projekte können Sie dieses Programm als Vorlage nehmen und anpassen. Die SQLite Datenbank, die Sie verwenden möchten, muss dazu im Ordner *data* Ihres Processing-Projektes gespeichert sein.

Schritt 1: Erstellen einer Benutzeroberfläche

Eine Benutzeroberfläche, die es Anwendenden erlaubt einige Informationen einzugeben, kann mit dem Tool *G4P GUI Builder* erstellt werden. 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 für die Eingabe Textfelder.

Abbildung 1: Benutzeroberfläche für die Suche nach dem Geburtsdatum

Erzeugen eines Textfeldes mit Beschriftung

In der oberen Menüleiste des *G4P GUI Builder* können verschiedene Dialogelemente wie Textfelder, Labels und Buttons ausgewählt und auf der Oberfläche platziert werden. (s. Abbildung 2). Nachdem wir z. B. ein Textfeld in das Design-formular gezogen haben, können rechts im Bereich *Properties* verschiedene Eigenschaften für das Textfeld festlegt werden. Dazu gehören z. B. der Name der entsprechenden Variablen oder der Text, der beim Starten des Programms im Textfeld angezeigt werden soll.

Damit klar ist, welche Information in ein Textfeld eingeben werden soll, ist es hilfreich dieses zu beschriften. Um Text, der nicht verändert werden kann, auf der Benutzeroberfläche auszugeben, bietet sich ein *Label* an.

In *Abbildung 2* sind alle wichtigen Stellen im Tool *G4P GUI Builder* umrandet. Der entsprechende Quelltext wird in dem Tab *gui* automatisch erzeugt und darf nicht verändert werden. Die Kommentare im generierten Quelltext sollten unbedingt beachtet werden.

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. Hier bieten sich z. B. eine Drop List oder ein Slider an.

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*.

Wir platzieren einen entsprechenden Button und legen die Eigenschaften fest. Neben dem Text für die Beschriftung und dem Namen für die Variable, ist die Eigenschaft *Event method* für den Button wichtig. Hier steht, wie die Methode heißen soll, die ausgeführt wird, wenn der Button angeklickt wird. Die Methode wird im Quelltext automatisch angelegt. Der Inhalt der Methode ist das Einzige, was wir im Quelltext verändern dürfen. Als Vorlage steht hier ein `println`-Befehl, der eine Testausgabe in der Konsole erzeugt. An dieser Stelle kann der eigene Quelltext ergänzt werden, der ausgeführt werden soll, wenn der Button angeklickt wird:

```
public void bSucheGebDatum_click(GButton source, GEvent event) {
  println("button1 - GButton >> GEvent." + event + " @ " + millis());
```

Diesen Teil schauen wir uns im nächsten Schritt genauer an.

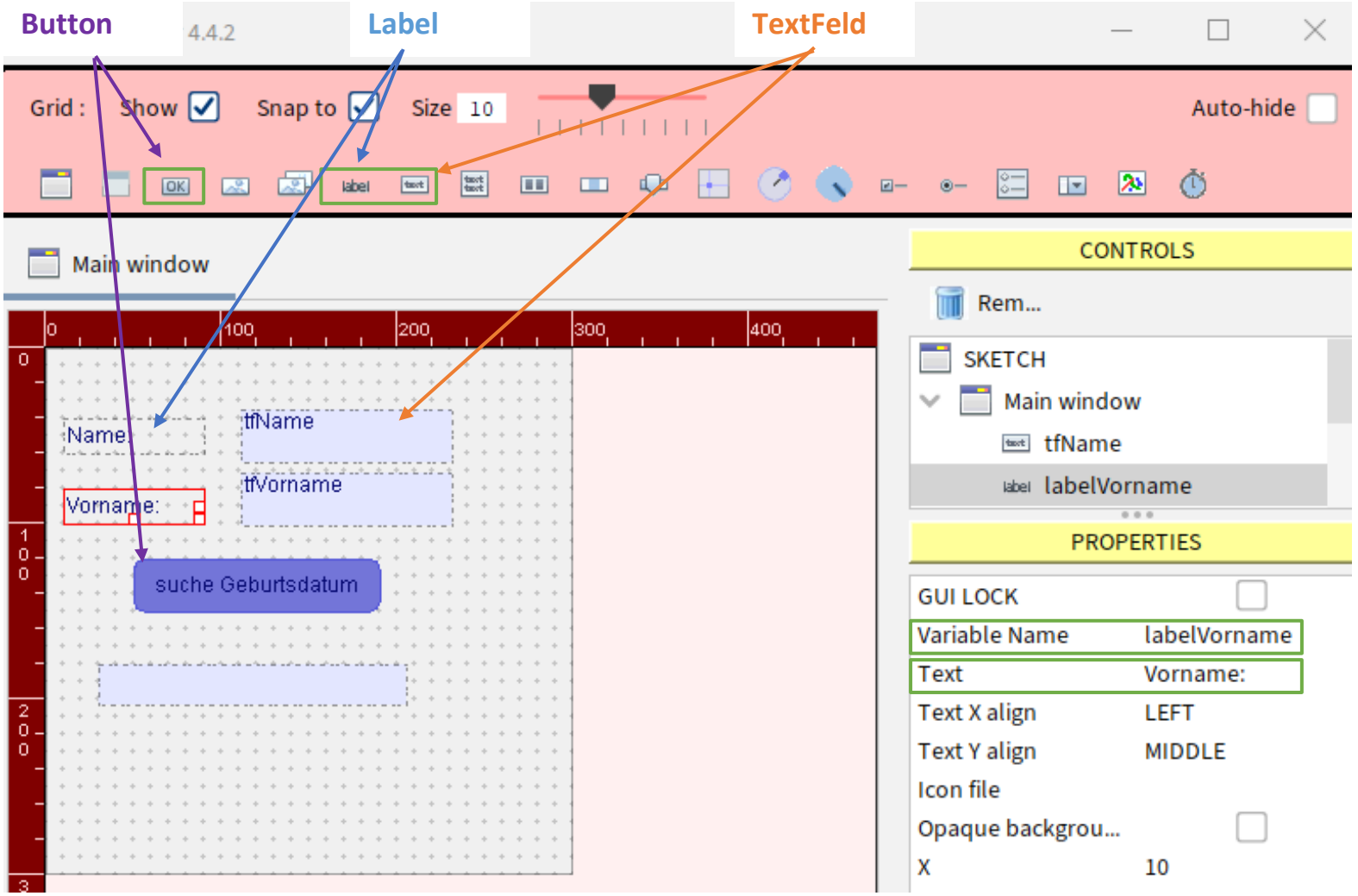

Abbildung 2: Erstellen eines Labels, eines Textfeldes und eines Buttons mithilfe des Tools G4P GUI Builder

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 Textfelder auslesen. Die Klasse *GTextField* stellt dafür die Methode `getText` zur Verfügung. Diese liefert den eingetragenen Text als Zeichenkette zurück. In unserem Beispiel beginnen wir also in der Methode `bSucheGebDatum_click` 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 *DBManagerProcSQLite*, das für uns die Kommunikation mit der Datenbank regelt. Deshalb erzeugen wir in der setup-Methode ein Objekt vom Typ *DBManagerProcSQLite* und speichern es in einer globalen Variablen. Diese steht dann auch im Tab *gui* zur Verfügung. Als Parameter benötigt der Konstruktor eine Referenz auf den aktuellen Sketch:

```
DBManagerProcSQLite myDBManager;
void setup(){
      …
     myDBManager = new DBManagerProcSQLite(this);
}
```

Als Variablenname wurde hier `myDBManager` gewählt. Dieser kann natürlich auch anders lauten. Wenn der *Konstruktor* ohne weitere 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:

```
myDBManager = new DBManagerProcSQLite(this, "schule");
```

Die Klasse *DBManagerProcSQLite* stellt die Methode `sqlAnfrageAusfuehren` zur Verfügung. Dieser Methode 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);
```

Der Aufruf erfolgt in der Methode `bSucheGebDatum_click` im Tab *gui*.

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).

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 nur aus einem Feld, allerdings mit dem Inhalt „Fehler“. Diese Abfrage hat dann vermutlich auch eine *SQLException* ausgelöst, zu der nähere Informationen in der Konsole stehen. Um solche Fehler weitgehend auszuschließen, ist es sinnvoll die *SQL-Anfrage* zunächst in einem *Datenbankbrowser*[1] zu testen, in den man *SQL-Anfragen* direkt eingeben und an die Datenbank senden kann.

[1] 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

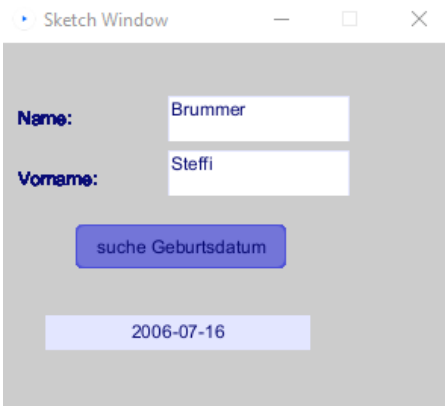

Abbildung 3: Ausgabe des Geburtsdatums

Das Ergebnis unserer SQL-Anfrage soll nun dem Anwender über die Benutzeroberfläche mitgeteilt werden. In unserer Beispiel-Anwendung wurde dafür ein weiteres *Label* 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.

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 Label `lAusgabe` für die Ausgabe 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 Label `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 *Label* ist. Alternativ bietet sich dann die Ausgabe in einer *TextArea* an. Diese wird genauso wie ein *Label* oder *TextField* erstellt, kann aber im Unterschied zum Textfeld mehrere Textzeilen anzeigen. Ein Beispiel befindet sich im Anhang: *Ergebnispräsentation in einer TextArea*.

Anhang

Ergebnispräsentation in einer TextArea

Eine *TextArea* kann im Unterschied zu einem *TextField* mehrere Zeilen Text anzeigen (s. Abbildung 4). Das Erzeugen einer *TextArea* funktioniert wie bei einem *TextField*. Im Bereich *Properties* kann festgelegt werden, ob die *TextArea* in horizontale bzw. vertikale Richtung scrollbar sein soll (s. Abbildung 5).

Abbildung 4: Ausgabe in einer TextArea

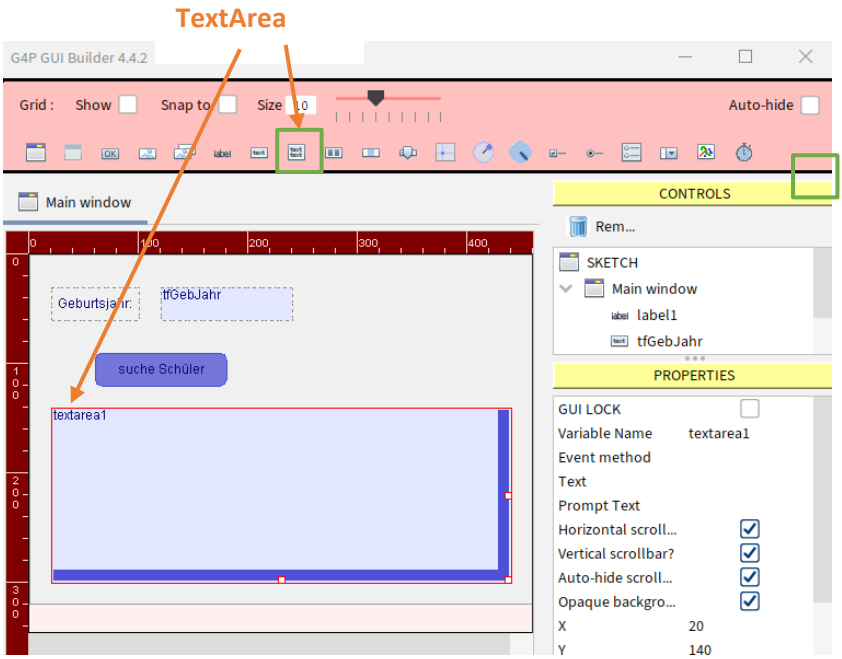

Abbildung 5: Erzeugen einer TextArea im Tool G4P GUI Builder

Eine *TextArea* stellt wie ein *TextField* eine Methode `setText` zur Verfügung, mit der ein neuer Inhalt in die *TextArea* geschrieben werden kann. Zusätzlich verfügt eine *TextArea* über die Methode `append`, mit der eine neue Textzeile an den bisherigen Inhalt angefügt werden kann.

Betrachten wir dazu ein Beispiel. Die Anwendung in Abbildung 4 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 `tfGebJahr`. Wir erzeugen zunächst die passende *SQL-Anfrage* und senden diese an die Datenbank:

```
1  String gebJahr = tfGebJahr.getText();
2  String sql = "select Name, Vorname, Geburtstag from schueler where
   Geburtstag like '" + gebJahr + "%';";
3  String[][] erg = myDBManager.sqlAnfrageAusfuehren(sql);
```

Bei erfolgreicher Suche enthält die Variable `erg` 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 4 angezeigt erzeugen:

```
1  textarea1.setText("");
2  if(erg.length > 1){
3  textarea1.appendText("Folgende Schülerinnen und Schüler wurden im Jahr "
   + gebJahr + " geboren:");
4  for(int i = 1; i < erg.length; i++){
5   textarea1.appendText(erg[i][0] + ", " + erg[i][1] + ", " + erg[i][2]);
6  }
7  }else{
8    if(erg[0][0].equals("Fehler")) textarea1.setText("Fehler");
9    else textarea1.setText("Keine Schüler gefunden.");
10 }
```

In Zeile 1 löschen wir den bisherigen Inhalt der TextArea `textarea1`.

In Zeile 2 testen wir, ob passende Datensätze gefunden wurden. Das ist der Fall, wenn die zweidimensionale Reihung `erg` mehr als eine Zeile enthält. In Zeile 3 wird die erste Zeile für die Ausgabe erstellt. Die Jahreszahl fügen wir dabei passend mithilfe der Variablen `gebJahr` ein. In Zeile 4 bis 6 wird die zweidimensionale Reihung dann zeilenweise durchlaufen. Für jede Zeile in der zweidimensionalen Reihung, die einen Datensatz enthält, wird eine Zeile an die `textarea1` 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.

In Zeile 8 und 9 unterscheiden wir noch die Fälle, dass die *SQL-Anfrage* einen Fehler ausgelöst hat (Zeile 8) bzw. dass kein Schüler und keine Schülerin zu dem angegebenen Geburtsjahr gefunden wurde (Zeile 9). Der Inhalt der `textarea1` wird dann jeweils auf den passenden Text gesetzt.

Den gesamten Quelltext zu dem Beispiel finden Sie in dem Ordner *BspProcSucheJg* im Ordner *Beispiele → Processing*.

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. Wir erhalten eine Zahl z. B. als Zeichenkette, 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. Das folgende Beispiel zeigt, wie die Jahreszahl, die in ein Textfeld eingegeben wird, in den Datentyp Ganzzahl umgewandelt wird:

```
1    String gebJahr = tfGebJahr.getText();
2    int gebJahrZahl = Integer.parseInt(gebJahr);
```

Die Klasse DBManagerProcSQLite

Die Klasse *DBManagerProcSQLite* 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. Sie stellt folgende Konstruktoren und Methoden zur Verfügung.

Konstruktoren

Konstruktor	Beschreibung
`DBManagerProcSQLite(` `PApplet papplet)`	erzeugt ein DBManagerProcSQLite-Objekt für den übergebenen Sketch und die Kommunikation mit der Datenbank schule_erweitert. Der aufrufende Sketch kann mit „this“ übergeben werden. Die Datenbank schule_erweitert.db muss als Datei im Unterordner *data* gespeichert sein.

`DBManagerProcSQLite(`
`PApplet papplet,`
`String dbName)`
	erzeugt ein DBManagerProcSQLite-Objekt für den übergebenen Sketch und die Kommunikation mit der übergebenen Datenbank. Der aufrufende Sketch kann mit „this“ übergeben werden. Die Datenbank muss als Datei im Unterordner *data* mit dem übergebenen Namen gespeichert sein.

Übersicht Methoden

Rückgabewert	Methode	Beschreibung
`void`	`datensatzAendern(String sql)`	führt die übergebene SQL-Update-Anweisung aus
`void`	`datensatzEinfuegen(String sql)`	führt die übergebene SQL-Einfüge-Anweisung aus
`void`	`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

datensatzEinfuegen

`public int datensatzEinfuegen(String sql)`

führt die übergebene SQL-Einfüge-Anweisung aus

Parameter:

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

datensatzLoeschen

`public int datensatzLoeschen(String sql)`

führt die übergebene SQL-Lösch-Anweisung aus

Parameter:

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

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".