Objektorientierte Systementwicklung:

Zugriff auf relationale Datenbanken mit Java

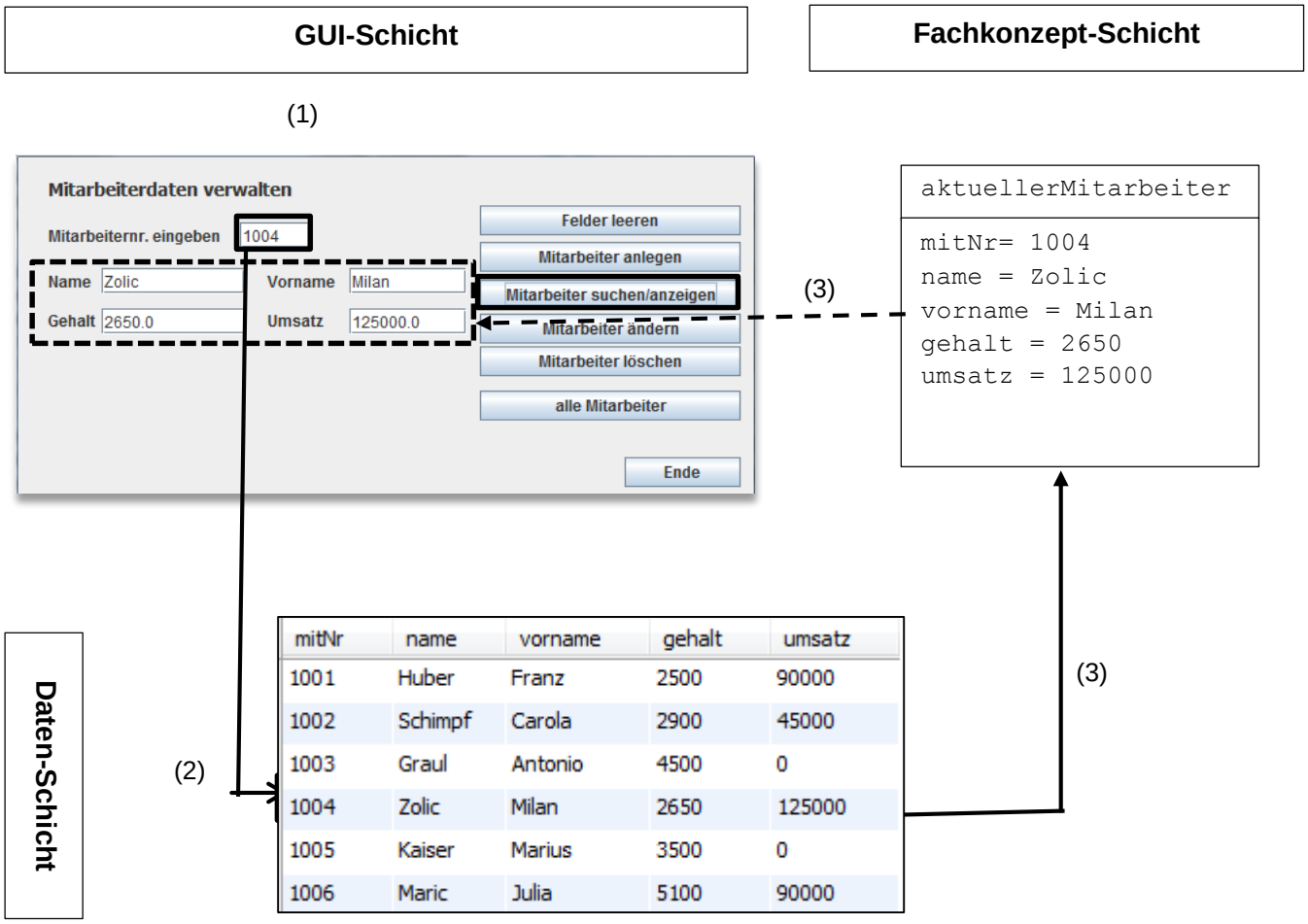
1 Einführung

1.1 Problemstellung

Die GeLa GmbH verwaltet ihre Mitarbeiterdaten mithilfe einer objektorientierten Software.
Im bisherigen Entwicklungsstand kann immer nur jeweils ein Mitarbeiterobjekt erzeugt werden oder es werden alle zu verwalteten Mitarbeiter in einer Liste in der Anwendung bereitgestellt. Eine dauerhafte Speicherung der Daten ist nicht möglich.

Nun soll die Software dahin gehend erweitert werden, dass die Stammdaten der Mitarbeiter in einer Datenbank dauerhaft gespeichert und verwaltet werden können. Es soll möglich sein, , die Daten der Mitarbeiter in die Datenbank aufzunehmen, diese zu ändern sowie zu löschen. Außerdem soll es möglich sein, alle Mitarbeiter der GeLa GmbH in einer Liste auszugeben oder nach einem bestimmten Mitarbeiter anhand der Mitarbeiternummer zu suchen um seine Daten anzuzeigen.

Die nachfolgende Abbildung soll den Ablauf des zu realisierenden Datenbankzugriffs veranschaulichen.

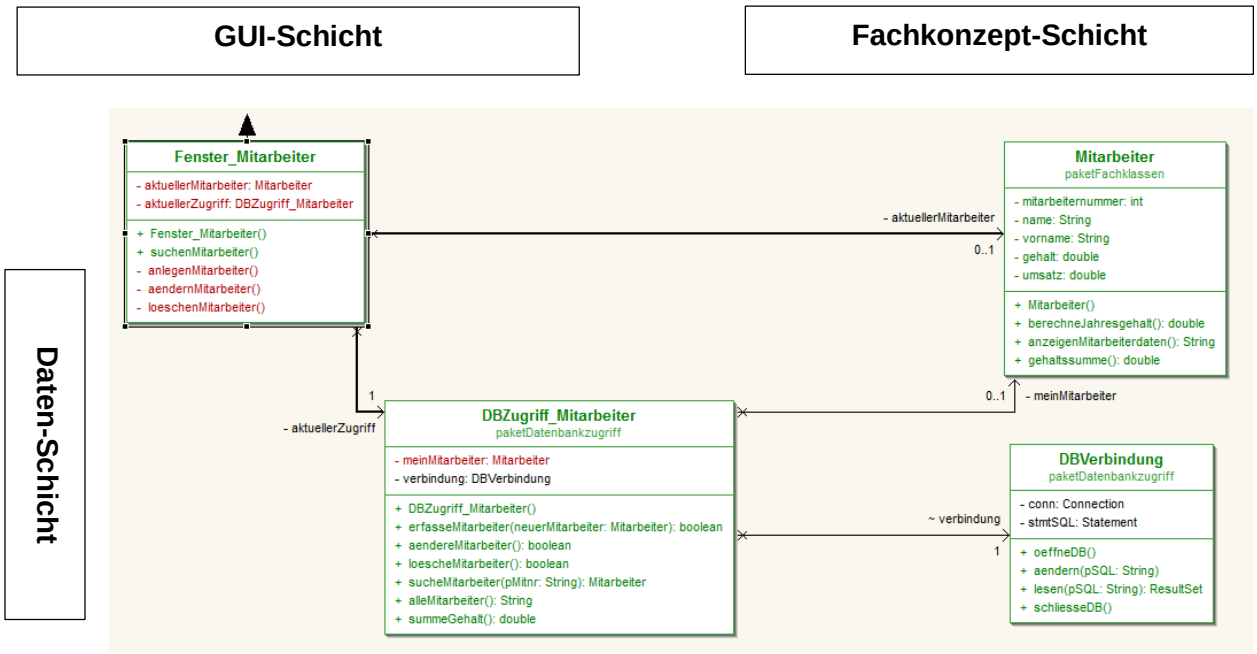


(1) Die Mitarbeiternummer des zu suchenden Mitarbeiters wird in die Maske eingegeben.
(2) Anhand der eingegebenen Mitarbeiternummer wird dann in der Tabelle Mitarbeiter der Datenbank der Datensatz mit der übereinstimmenden Mitarbeiternummer gesucht und
(3) Die gefundenen Daten werden an das Objekt *aktuellerMitarbeiter* übergeben.
(4) Die Attributswerte des Objekts *aktuellerMitarbeiter* werden in den Textfeldern der Maske angezeigt.

1.2 Das Drei-Schichten-Modell

Aus der Darstellung des Ablaufs wird bereits die Trennung von Darstellung, Verarbeitung und Speicherung der Daten deutlich.

In der nachfolgenden Abbildung ist die Drei-Schichten-Architektur erkennbar. Das bisherige Modell, bestehend aus der GUI-Schicht und der Fachkonzept-Schicht ist um die Datenschicht erweitert.



Erläuterungen

Die Drei-Schichten-Architektur ist eine bewährte Struktur zum Modellieren von Anwendungen:

- **Die Benutzeroberfläche (GUI-Schicht)**ist als Schnittstelle zum Benutzer sowohl für die Darstellung der Daten als auch für die Dialogführung zuständig. In der GUI-Schicht werden die Zugriffe auf die Datenhaltungsschicht koordiniert.

- In der **Fachkonzept-Schicht** werden die fachlichen Probleme der Anwendung, losgelöst von Speicherung und Präsentation, beschrieben (Attribute und Methoden).

- In der **Datenhaltungsschicht** wird die Datenspeicherung in der Datenbank vorgenommen.

Die beteiligten Klassen der Mitarbeiterverwaltung mit Datenbankzugriff

In der **Fachklasse** »Mitarbeiter« werden die zu bearbeitenden Mitarbeiter beschrieben. Für den Austausch mit der Benutzungsoberfläche werden Standardmethoden („setter“ und „getter“) zur Verfügung gestellt. Die Fachkonzeptschicht besitzt kein Wissen über die GUI-Schicht. Bei einer Änderung der Benutzungsoberfläche muss das Fachkonzept nicht angepasst werden.

Die Klasse »Fenster_Mitarbeiter« **der GUI-Schicht** beschreibt die Maske für die zu bearbeitenden Mitarbeiterobjekte und koordiniert die Zugriffe auf die Datenbank mit den Methoden **`anlegenMitarbeiter()`**, **`suchenMitarbeiter`**, **`aendernMitarbeiter()`** und **`loeschenMitarbeiter()`**.
Die GUI-Klasse »Fenster_Mitarbeiter« enthält also eine Referenz zur Fachklasse »Mitarbeiter« (Referenzattribut *aktuellerMitarbeiter*) und zur Zugriffsklasse »DBZugriff_Mitarbeiter« (Referenzattribut *aktuellerZugriff*.

Die Datenhaltungsschicht umfasst die Klassen »DBZugriff_Mitarbeiter« und »DBVerbindung».
Die eigentliche Kommunikation mit der Datenbank („was soll in der Datenbank gemacht werden“) wird mithilfe der Klasse »DBZugriff_Mitarbeiter« beschrieben. Ihre Methoden *erfasseMitarbeiter()*, *sucheMitarbeiter()*, *aendereMitarbeiter()*, *loescheMitarbeiter()* sowie *alleMitarbeiter()* enthalten die jeweiligen SQL-Anweisungen für die entsprechenden Operationen in der Datenbank. In den Methoden der Klasse »Fenster_Mitarbeiter« kommen auch Standardmethoden („setter“ und „getter“) der Fachklasse »Mitarbeiter« zum Einsatz.
Die Klasse »DBZugriff_Mitarbeiter« besitzt eine Referenz zur Fachklasse »Mitarbeiter« (Referenzattribut *aktuellerMitarbeiter*)

Während in der Klasse »DBZugriff_Mitarbeiter« die Kommunikation mit der Datenbank in Form von SQL-Anweisungen beschrieben wird, wird sie mithilfe der Klasse »DBVerbindung« auf der Datenbank ausgeführt. In der Klasse »DBZugriff_Mitarbeiter« besteht eine Referenz zur Klasse »DBVerbidnung« (Referenzattribut *verbindung*).
Die Klasse »DBVerbidnung« enthält die notwendigen Datenbanktreiber sowie die Methoden *oeffneDB(), lesen(), aendern()* und *schliesseDB().*

Quellcode der Klasse DBVerbindung mit ergänzenden Erläuterungen

```
(1)  import java.sql.Connection;                    // oder import java.sql.*
     import java.sql.DriverManager;
     import java.sql.ResultSet;
     import java.sql.SQLException;
     import java.sql.Statement;

     public class DBVerbindung {
     //Variablen für den Verbindungsaufbau
(2)  Connection conn = null;
     Statement stmtSQL = null;
     //Methoden
(3)  public void oeffneDB() {
     try {  System.out.println("* Treiber laden");
(3a)       Class.forName("com.mysql.jdbc.Driver").newInstance();
           System.out.println(Class.forName("com.mysql.jdbc.Driver"));
           System.out.println(Class.forName("com.mysql.jdbc.Driver").newInstance());
       }
     catch (Exception e)     {
           System.err.println("Unable to loa
           e.printStackTrace();
       }

     try {   conn =
(3b) DriverManager.getConnection("jdbc:mysql://localhost:3306/mitarbeiterDB?user=root");
           stmtSQL = conn.createStatement();
           System.out.println("Connection etabliert");
           }
     catch (SQLException ex) {
           System.out.println("SQLException: " + ex.getMessage());
           System.out.println("SQLState: " + ex.getSQLState());
           System.out.println("VendorError: " + ex.getErrorCode());
           }
     }

(4)  public boolean aendern(String pSQL){
     try {
           stmtSQL.executeUpdate(pSQL);
            return true;
     }
     catch(SQLException err) {
           System.err.println(err);
           return false;
     }
     }
(5)  public ResultSet lesen(String pSQL)
     ResultSet rs;
     try    {
           rs = stmtSQL.executeQuery(pSQL);
           return rs;
           }
     catch(SQLException err) {
           System.err.println(err);
           rs = null;
           return rs;
           }
     }
(6)  public void schliesseDB(){
     try   {stmtSQL.close();
           conn.close();
           }
     catch (SQLException err) {
           System.err.println(err);
           }
      }
     }
```

Klassen aus der Bibliothek java.sql.* werden importiert

Objekte zur Herstellung der DB-Verbindung werden initialisiert

Herstellen der Datenbankverbindung
- mithilfe des „*jdbdc“-Datenbanktreibers*
- wird die Datenbank *mitarbeiterDB* auf dem Server *localhost* geöffnet

Die Datenbank wird mit dem übergebenen SQL-Befehl aktualisiert. Es kann eine
- Einfügeabfrage (**`insert into`**…)
- Aktualisierungabfrage (**`update`** …)
- Löschabfrage (**`delete from`**…)

übergeben werden.

Die Datenbank wird mit dem übergebenen SQL-Befehl ausgewertet (abgefragt).
Es wird eine Auswahlabfrage (**`select`**) übergeben, deren Ergebnis (keine, ei oder mehrere Datensätze) in einer **virtuellen Tabelle** (Objekt *rs* der Klasse »ResultSet«) zurück gegeben werden

Schließen/Beenden der Datenbankverbindung

Erläuterungen

(1) Für die Verbindung zu einer Datenbank aus Java werden aus dem Paket **`java.sql.*`** Klassen mit den notwendigen Methoden zur Verfügung gestellt. Für die Verbindung zur Datenbank werden insbesondere die Klasse »DriverManager« und die Klasse »Connection« sowie die Klasse »Statement« verwendet.

(2) Für die Verbindung wird das Objekt *conn* der Klasse Connection, für SQL-Abfragen das Objekt *stmt* der Klasse »Statement« vorbereitet.

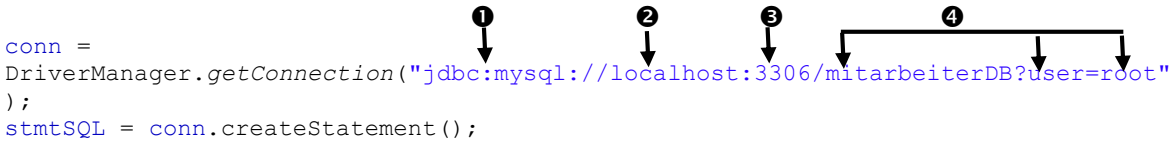
(3) Die Verbindung zur Datenbank wird mit der Methode **`public void oeffneDB()`** hergestellt und läuft in zwei Schritten ab:

(3a) Das Laden des Datenbanktreibers
mit der Anweisungszeile

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

wird das Treiberobjekt erzeugt. Wie der MySQL-Treiber mithilfe der Entwicklungsumgebung *Eclipse* in das Projekt eingebunden wird, wird an späterer Stelle erläutert.

(3b) Die Verbindung zur Datenbank herstellen
Nach dem Laden des Datenbanktreibers wird die Verbindung zur Datenbank mithilfe des Connection-Objekts aufgebaut.



Die Methode `getConnection` der Klasse «Connection» weist dem Objekt *conn* die Verbindungsdaten zur Datenbank zu.

Die Methode `getConnection()` erwartet folgende Angaben:

❶ die verwendete Datenbank (hier *MySQL*)
❷ den PC, auf dem die Datenbank betrieben wird. *localhost* steht für den lokalen PC. Anstelle des Hosts kann auch dessen IP-Adresse angegeben werden (der lokale PC hat die IP-Adresse 127.0.0.0).
❸ den *Port* für die Verbindung. 3306 ist der Standardport für MySQL.
❹ Namen der Datenbank (hier *mitarbeiterDB*, Datenbankbenutzer (hier *root*). Optional kann auch das Passwort mit angegeben werden, sofern es in MySQL festgelegt wurde. Wurde beispielsweise „*hallodb*“ als als Passwort für die Datenbank mitarbeiterDB festgelegt, würden die Parameter für die Datenbankverbindung folgendermaßen erweitert:

```
("jdbc:mysql://localhost:3306/mitarbeiterDB?user=root&password=hallodb")
```

In der Zeile

```
stmtSQL = conn.createStatement();
```

wird das Objekt *stmtSQL* mit den Verbindungsdaten des Verbindungsobjekts *conn* initialisiert. Mit Hilfe des Objekts *stmtSQL* können in der Folge mit den Methoden **`executeQuery`** und **`executeUpdate`** Operationen auf der Datenbank ausgeführt werden (siehe Schritte (4) bis (6)).

(4) Mit der Methode **`public boolean`** `aendern(String pSQL)` werden Datenbankänderungen durchgeführt. Die Methode fordert als Übergabewert einen String mit der betreffenden SQL-Anweisung an und gibt einen Wahrheitswert *true* zurück, wenn die Änderung erfolgreich,

beziehungsweise *false*, wenn sie nicht erfolgreich war. Bei Änderungen in der Datenbank werden **`insert into`**-, **`update`**- und **`delete from`**- Anweisungen übergeben.

Mit der Methode **`executeUpdate`** des Objekts *stmtSQL* (`stmtSQL.executeUpdate(pSQL);`) wird die übernommene SQL-Anweisung auf der Datenbank ausgeführt.

(5) Mit der Methode **`public ResultSet`** `lesen(String pSQL)` werden Datenbankabfragen durchgeführt. Die Methode fordert als Übergabewert einen *String* mit der betreffenden SQL-Auswahlabfrage (**`Select`**) an und liefert als Ergebnis ein Objekt der Klasse «ResultSet». Mit der Methode **`executeQuery`** des Objekts *stmtSQL* (**`rs=stmtSQL.executeQuery(pSQL;)`**) wird dem ResultSet-Objekt *rs* das Ergebnis der SQL-Abfrage in Form einer Ergebnistabelle zugewiesen. Mit der Anweisung **`return rs;`** wird dann die Ergebnistabelle zurückgegeben.

Je nach Problemstellung kann das ResultSet-Objekt *rs* keine, eine oder mehrere Zeilen umfassen.

(6) Mit der Methode **`public void`** `schliesseDB()` wird die Datenbankverbindung mit der Methode `close()` für das Verbindungsobjekt *conn* (`conn.close();`) beendet.

Hinweis:
Im Unterricht wird die Klasse «DBVerbindung» zur Verfügung gestellt und erklärt. Der Quellcode der Klasse wird nicht verändert.

2 Mitarbeiterverwaltung für die GeLa GmbH

Problemstellung

Die GeLa GmbH verwaltet ihre Mitarbeiterdaten mithilfe einer objektorientierten Software.
Im bisherigen Entwicklungsstand der Software können die Daten nicht dauerhaft gespeichert werden.
Nun soll die Software erweitert werden, so dass die Stammdaten der Mitarbeiter in einer Datenbank dauerhaft gespeichert und verwaltet werden können (siehe Seite 2).

Benutzeroberfläche: **Datenbank MitarbeiterDB**

Bei den Aktivitäten der Software, die durch die jeweiligen Befehlsschaltflächen [*Mitarbeiter anlegen*], **`[Mitarbeiter suchen/anzeigen]`**, **`[Mitarbeiter ändern]`** und **`[Mitarbeiter löschen]`** sowie **`[alle Mitarbeiter]`**), ausgelöst werden, muss auf die Tabelle *mitarbeiter* der Datenbank *MitarbeiterDB* zugegriffen werden.

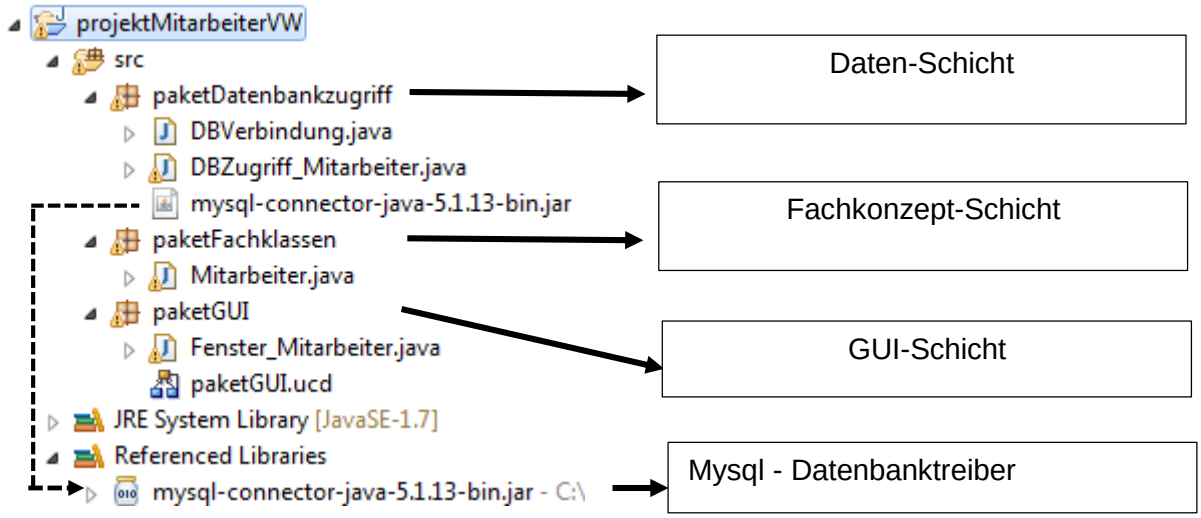
2.1 „Anwendungsgerüst“ unter der Entwicklungsumgebung Eclipse

Wird der Zugriff auf eine relationale Datenbank mit Java realisiert, kann man das „Drei – Schichten – Modell“ (siehe Seite 2) im Javaprojekt abbilden. Dabei wird für jede Schicht ein eigenes Paket angelegt, in das dann die jeweiligen Klassen gespeichert werden.

Setzt man die Entwicklungsumgebung Eclipse ein, wird im Javaprojekt die nachfolgend dargestellte Projektstruktur (=“Anwendungsgerüst“) angelegt.

Aufgabenstellung:

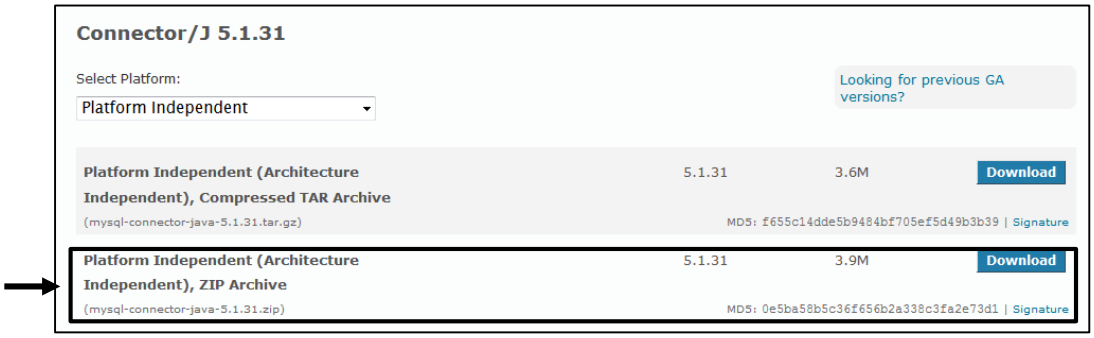
Starten Sie die Entwicklungsumgebung *Eclipse* und importieren Sie das Projekt *projektMitarbeiterVW* (**`File→Import→Existing Projects into Workspace`**)



Damit eine Verbindung aus Java zu einer relationalen Datenbank aufgebaut werden kann, muss ein „JDBC“-Treiber in das Projekt eingebunden werden (JDBC = Java Database Connectivity). Hierbei handelt es sich um eine standardisierte Schnittstelle, die Klassen und Methoden bereitstellt, um relationale Datenbanken von Java aus zu nutzen.
Für MySQL-Datenbanken kann dieser Treiber aus dem Internet (http://www.mysql.com) als ZIP-Archiv bezogen werden. Er muss dann entpackt und in den Klassenpfad aufgenommen werden.

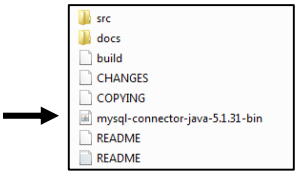
Vorgehensweise:

(1) MySQL-Treiber (=**MySQL-Connector**) von der Seite (http://www.mysql.com) **downloaden**



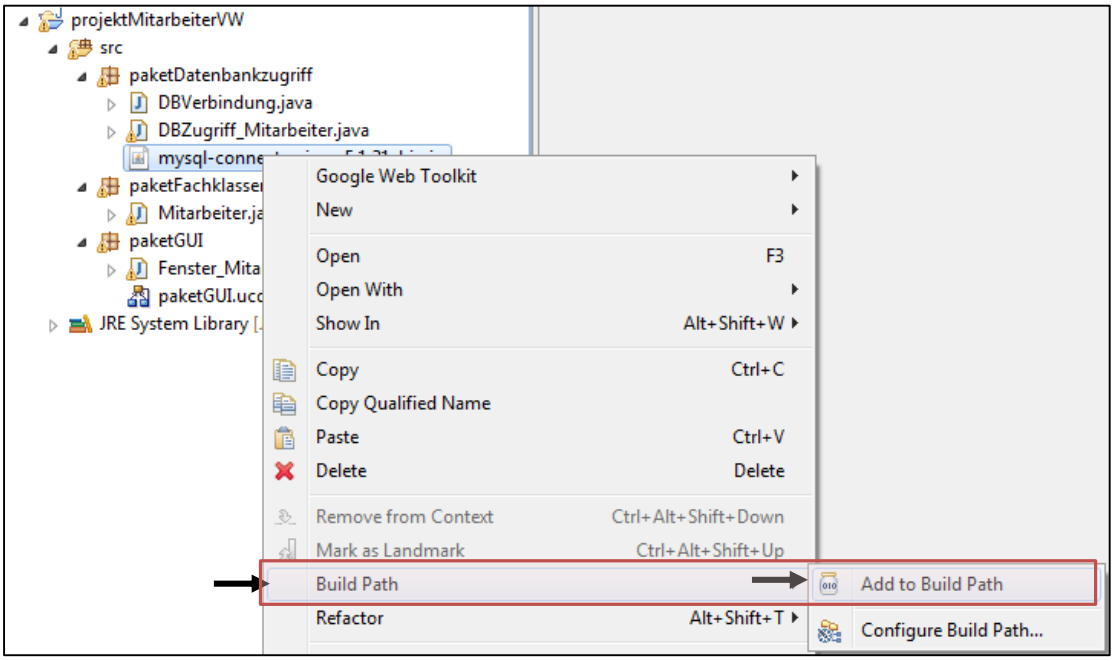
Hinweis: um den Download nutzen zu können, muss man sich vorher bei MySQL registrieren. Kosten entstehen dabei nicht.

(2) Nachdem das ZIP-Archiv entpackt ist wird die Datei
`mysql-connector-java-5.1.31-bin.jar`

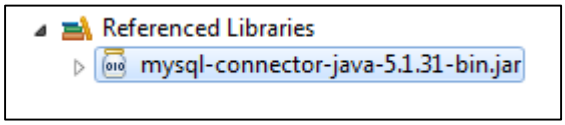


in das Paket *paketDatenbankzugriff* kopiert. Von dort aus wird der Treiber dann

(3) in den Klassenpfad aufgenommen.
Dazu wird der Connector angeklickt und mit der rechten Maustaste über das Kontextmenü (Klick mit rechter Maustaste) der Menübefehl **`Build Path→ Add to Build Path`** aufgerufen.

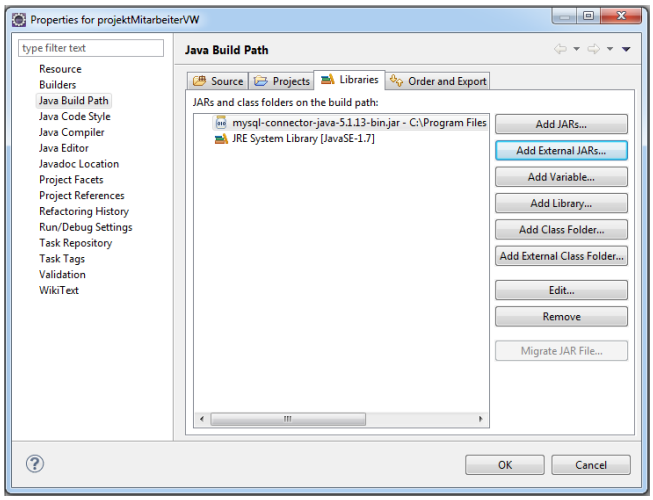


(4) Der Connector wird in den Klassenpfad aufgenommen und erscheint im Projekt unter dem Eintrag *Referenced Libraries.*



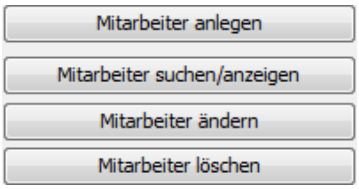
Alternativ kann der MySQL-Connector auch aus jedem beliebigen Verzeichnis heraus eingebunden werden.

Vorgehensweise:
Das Projekt im Package Explorer markieren und dann mit der rechten Maustaste den Menübefehl `Build Path → Configure Build Path` öffnen. Im nun erscheinenden Fenster kann mit der Schaltfläche `[Add External JARs]` zum Verzeichnis, in dem der MySQL-Connector sich befindet, navigiert und dieser eingebunden werden.
Der eingebundene Konnektor erscheint dann ebenfalls im Projekt unter dem Eintrag *Referenced Libraries* (siehe Abbildung oben).


Properties for projektMitarbeiterVW
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project Facets
Project References
Refactoring History
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
mysql-connector-java-5.1.13-bin.jar - C:\Program Files
JRE System Library [JavaSE-1.7]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel

2.2 Stammdatenverwaltung für die Mitarbeiter der GeLa GmbH

Problemstellung
Für die Verwaltung der Mitarbeiterdaten in der MySQL-Datenbank *MitarbeiterDB* sollen die in den nebenstehenden Schaltflächen abgebildeten Aktivitäten programmiert werden.



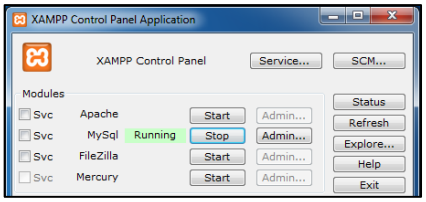
Die Mitarbeiterdaten sind in der Tabelle *mitarbeiter* der Datenbank *MitarbeiterDB* auf dem MySQL-Server zu speichern.

Einrichten der Datenbank MitarbeiterDB auf dem MySQL-Server

Für die Einrichtung der Datenbank *MitarbeiterDB* mit der Tabelle *mitarbeiter* und den dazugehörenden Mitarbeiterdaten auf dem Datenbankserver *MySQL* liegt das Skript *mitarbeiterdb_mitarbeiter.sql* vor[1]. Damit die Datenbank mit den Daten auf dem Datenserver eingerichtet werden kann, muss das Skript mithilfe der Software *MySQL Workbench* geöffnet und ausgeführt werden.

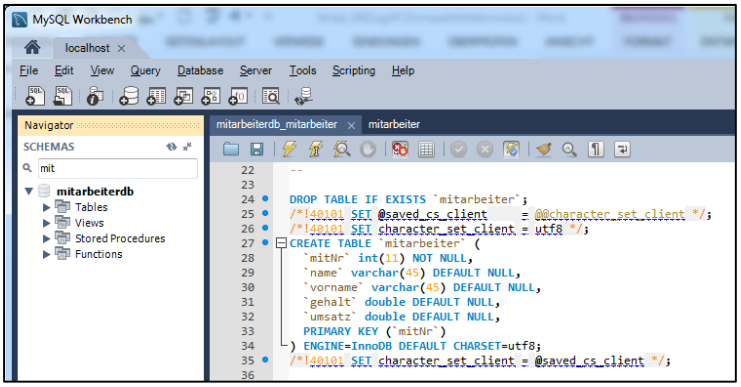
Vorgehensweise

(1) Starten des *MySQL*-Servers aus dem *XAMPP*-Paket

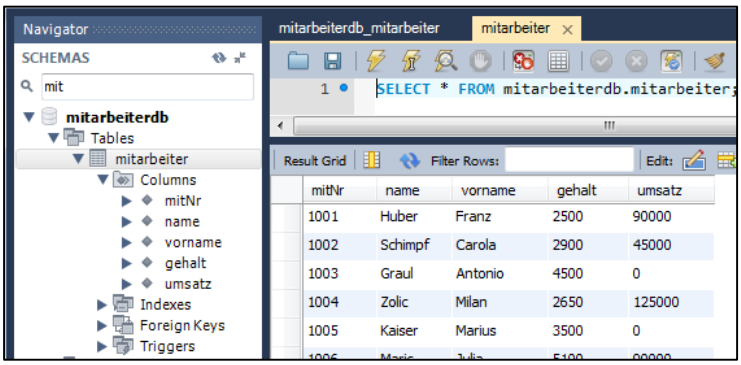


(2) Das Datenbank-Frontend MySQL Workbench starten und das Skript *mitarbeiterdb_mitarbeiter.sql* öffnen (Befehl **`File→Open SQL Script`**) und ausführen ()

(3) Anschließend die Ansicht (Navigator) aktualisieren (F5).



(4) Alle Datensätze der Tabelle *mitarbeiter* auflisten (`select * from mitarbeiter`)

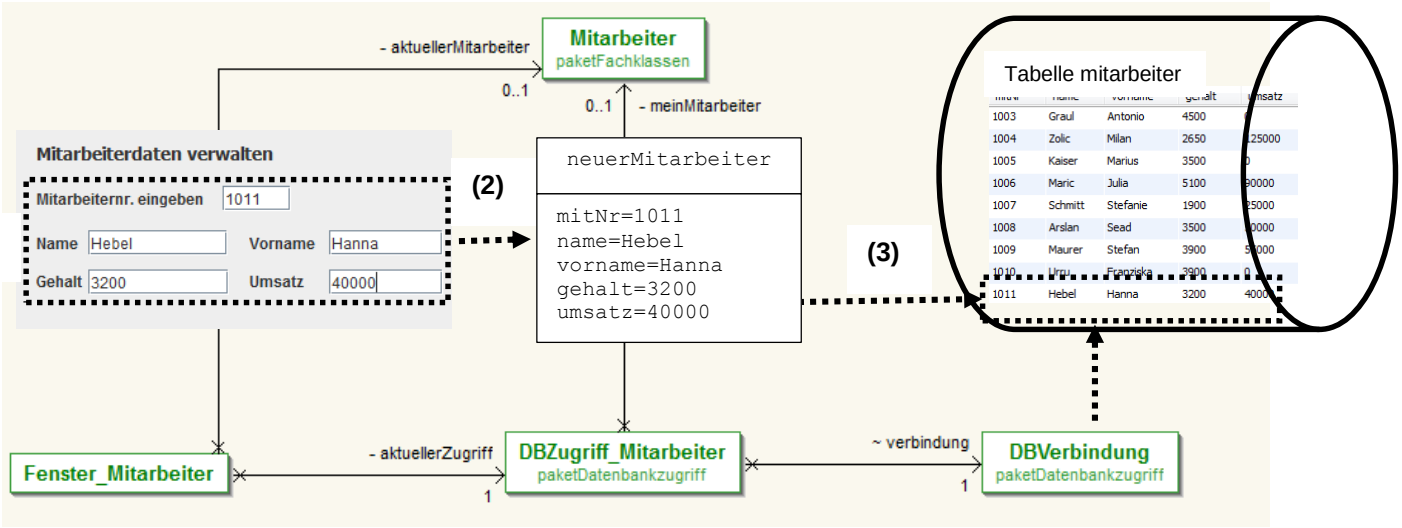


[1] Weitere Ausführungen zum Arbeiten mit der MySQL Workbench können unter http://www.schule-bw.de/unterricht/faecher/informatik/material/datenbanken/relationale-datenbanken/ aufgerufen werden.

2.2.1 Erfassen von Mitarbeitern in die Datenbank

Problemstellung:
Die neu eingestellte Mitarbeiterin Hanna Hebe (Monatsgehalt 320000 €; Umsatz 40000,00 €) soll erfasst und in der Datenbank gespeichert werden.

Die nachfolgende Abbildung veranschaulicht, wie ein neuer Mitarbeiter in die Datenbank aufgenommen wird.



(1) die Mitarbeiterdaten werden in der GUI erfasst und
(2) in das Objekt *neuerMitarbeiter* übernommen.
(3) Die Attributswerte des Objekts *neuerMitarbeiter* werden in die Tabelle *mitarbeiter* der Datenbank *MitarbeiterDB* geschrieben.

Arbeitsaufträge

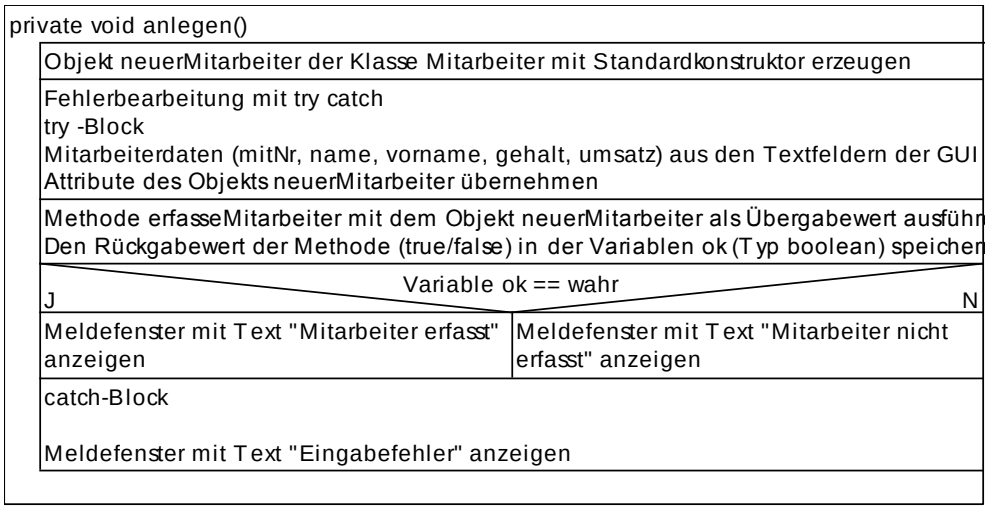
1) Beschreiben Sie, welche Objekte und Methoden beim Speichern eines neuen Mitarbeiters in die Tabelle *Mitarbeiter* der Datenbank benötigt werden.

2) Mit der Methode `public boolean erfasseMitarbeiter(Mitarbeiter neuerMitarbeiter)` der Klasse «DBZugriff_Mitarbeiter» werden die mithilfe der GUI in das Objekt *neuerMitarbeiter* erfassten Daten in der Tabelle *mitarbeiter* der Datenbank *mitarbeiterdb* gespeichert.

   Erklären Sie die Anweisungen des Quellcodes dieser Methode.

```
41    public boolean erfasseMitarbeiter(Mitarbeiter neuerMitarbeiter)
42    {
43    try
44    {
45        mSQL = "INSERT INTO mitarbeiter (mitnr, name, vorname, gehalt, umsatz) ";
46        mSQL += "VALUES ('" + neuerMitarbeiter.getMitarbeiternummer() + "','";
47        mSQL +=neuerMitarbeiter.getName() + "','" +neuerMitarbeiter.getVorname()+ "',";
48        mSQL += neuerMitarbeiter.getGehalt() + "," + neuerMitarbeiter.getUmsatz() + ")";
49        verbindung.oeffneDB();
50        ok = verbindung.aendern(mSQL);
51        verbindung.schliesseDB();
52    }
53    catch (Exception e)
54    {
55        
56        System.out.println(e);
57        ok=false;
58    }
59    return ok;
60    }
```

3) Für die Methode **`private void anlegen()`** in der Klasse «Fenster_Mitarbeiter» wurde das nachfolgende Struktogramm erstellt.



Codieren Sie das Struktogramm!

Lösungshinweise

1) Benötigte Objekte und Methoden zum Erfassen eines Mitarbeiters in die Datenbank

   Die Daten des neu zu speichernden Mitarbeiters werden

   (1) in die Textfelder der GUI (Objekte der Klasse «JTextField») erfasst.
   Wenn die Schaltfläche **`[Mitarbeiter anlegen]`** gedrückt wird, werden die Daten

   (2) aus den Textfeldern der GUI in das Objekt *neuerMitarbeiter* der Fachklasse «Mitarbeiter» übertragen.

   (3) Mit der Methode **`erfasseMitarbeiter(…)`** des Objekts *aktuellerZugriff)* der Klasse «DBZugriff_Mitarbeiter» wird die Aufnahme des Mitarbeiters in die Datenbank vorgenommen und mit der Methode **`aendern(pSQL)`** des Objekts *verbindung* der Klasse «DBVerbindung» auf dem Datenbankserver ausgeführt.

2) Erläuterung des Quellcodes der Methode `public boolean erfasseMitarbeiter(…)`

Zeile	Erläuterung
41	Die Methode `public boolean erfasseMitarbeiter(Mitarbeiter neuerMitarbeiter)` ist öffentlich. Sie benötigt als Übergabewert ein Objekt der Klasse «Mitarbeiter».
43	Um etwaige Fehler abzufangen, ist eine Fehlerbearbeitung eingerichtet. Im try-Block wird die SQL-Anweisung zum Speichern eines Mitarbeiters als *String* in der Variablen *mSQL* gespeichert. Der *String* wird aus Gründen der Übersichtlichkeit zeilenweise aufgebaut: **`mSQL +=`** "…" bedeutet, dass nach jeder Zeile der String in der Variablen *mSQL* um den Ausdruck, der rechts von += steht erweitert wird.

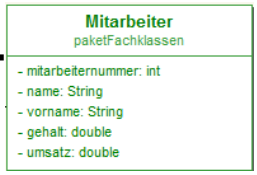
Attributsbezeichnungen aus der Tabelle *mitarbeiter*

mitNr	name	vorname	gehalt	umsatz
1011	Hebel	Hanna	3200	40000

bis 48

```
mSQL = "INSERT INTO mitarbeiter (mitnr, name, vorname, gehalt, umsatz) ";
mSQL += "VALUES ('" + neuerMitarbeiter.getMitarbeiternummer() + "','";
mSQL +=neuerMitarbeiter.getName() + "','" +neuerMitarbeiter.getVorname()+ "',";
mSQL += neuerMitarbeiter.getGehalt() + "," + neuerMitarbeiter.getUmsatz() + ")";
```

Zugriffsmethoden aus der Klasse «Mitarbeiter»



49	Die Datenbankverbindung wird mit der Methode **`oeffneDB()`** des Objekts *verbindung* geöffnet.
50	Die Methode **`aendern(mSQL)`** wird mit dem Objekt *verbindung* der Klasse «DBVerbindung» ausgeführt. Die Methode erhält als Übergabewert den gespeicherten SQL-Befehl und gibt als Ergebnis den Wahrheitswert *true* oder *false* zurück, je nachdem ob die Speicherung erfolgreich war oder nicht.
51	Die Datenbankverbindung wird mit der Methode **`schliesseDB()`** des Objekts *verbindung* der Klasse «DBVerbindung» geschlossen.
53 57	Im *catch*-Block der *try…catch*-Struktur wird der Ausnahmefehler abgefangen und dem Objekt *e* übertragen (Zeile 53). Anschließend (Zeile 56) wird der Inhalt der Meldung im Konsolenfenster angezeigt (Zeile 57). Die „Steuervariable“ *ok* erhält dann den Wahrheitswert *false.*
59	Der Wert der Variablen *ok* wird zurückgegeben.

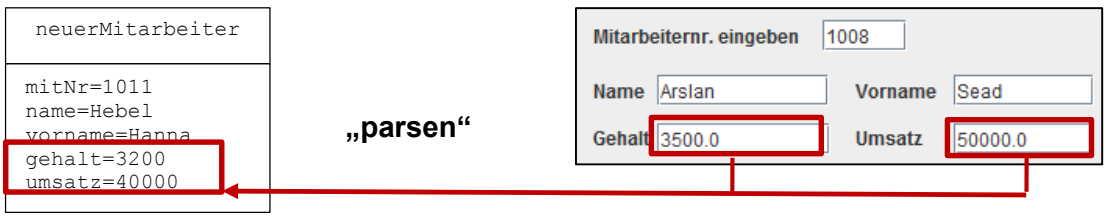
3) Quellcode der Methode in der Klasse *Fenster_Mitarbeiter*

```
292     // Methode anlegenMitarbeiter() um neue Mitarbeiter in der Datenbank anzulegen
293     private void anlegenMitarbeiter()
294     {  // Objekt neuerMitarbeiter der Klasse Mitarbeiter
295         Mitarbeiter neuerMitarbeiter = new Mitarbeiter();
296         try
297         {  // Eingabedaten aus den Textfeldern in die Attribute des Objekts neuerMitarbeiter übernehmen
298             neuerMitarbeiter.setMitarbeiternummer(Integer.parseInt(txtMaNr.getText()));
299             neuerMitarbeiter.setName(txtName.getText());
300             neuerMitarbeiter.setVorname(txtVorname.getText());
301             neuerMitarbeiter.setGehalt(Double.parseDouble(txtGehalt.getText()));
302             neuerMitarbeiter.setUmsatz(Double.parseDouble(txtUmsatz.getText()));
303             // Methode erfasseMitarbeiter mit dem Objekt neuerMitarbeiter als Übergabewert ausführen.
304             // Den Rückgabewert der Methode (true/false) in der Variablen ok (Typ boolean) speichern.
305             boolean ok = aktuellerZugriff.erfasseMitarbeiter(neuerMitarbeiter);
306             // Auswertung der Variablen ok
307             if (ok== true)
308             {
309                 JOptionPane.showMessageDialog(this, "Mitarbeiter erfasst");
310             } else
311             {
312                 JOptionPane.showMessageDialog(this, "Mitarbeiter nicht erfasst");
313             }
314         }
315         // Abfangen von Eingabefehlern
316         catch (Exception e)
317         {
318             JOptionPane.showMessageDialog(null, "Eingabefehler");
319         }
320     }
```

Erläuterungen

Zeile	Erläuterung
297	Eingabedaten aus den Textfeldern in die Attribute des Objekts *neuerMitarbeiter* übernehmen.
302	Wenn die Datentypen von Quelle und Ziel unterschiedlich sind, müssen die Daten vom Typ *String* bei der Übernahme in nummerische Attribute „geparst“ werden (Zeile 301 und 302). Im vorliegenden Fall sollen die Daten *Gehalt* und *Umsatz* aus den Textfeldern der GUI (=Quelle) in die nummerischen Attribute *gehalt* und *umsatz* des Objekts (=Ziel) übertragen werden.

Ziel: nummerische Attribute **Quelle: Textfelder der GUI**




```
neuerMitarbeiter.setGehalt(Double.parseDouble(txtGehalt.getText()));
neuerMitarbeiter.setUmsatz(Double.parseDouble(txtUmsatz.getText()));
```

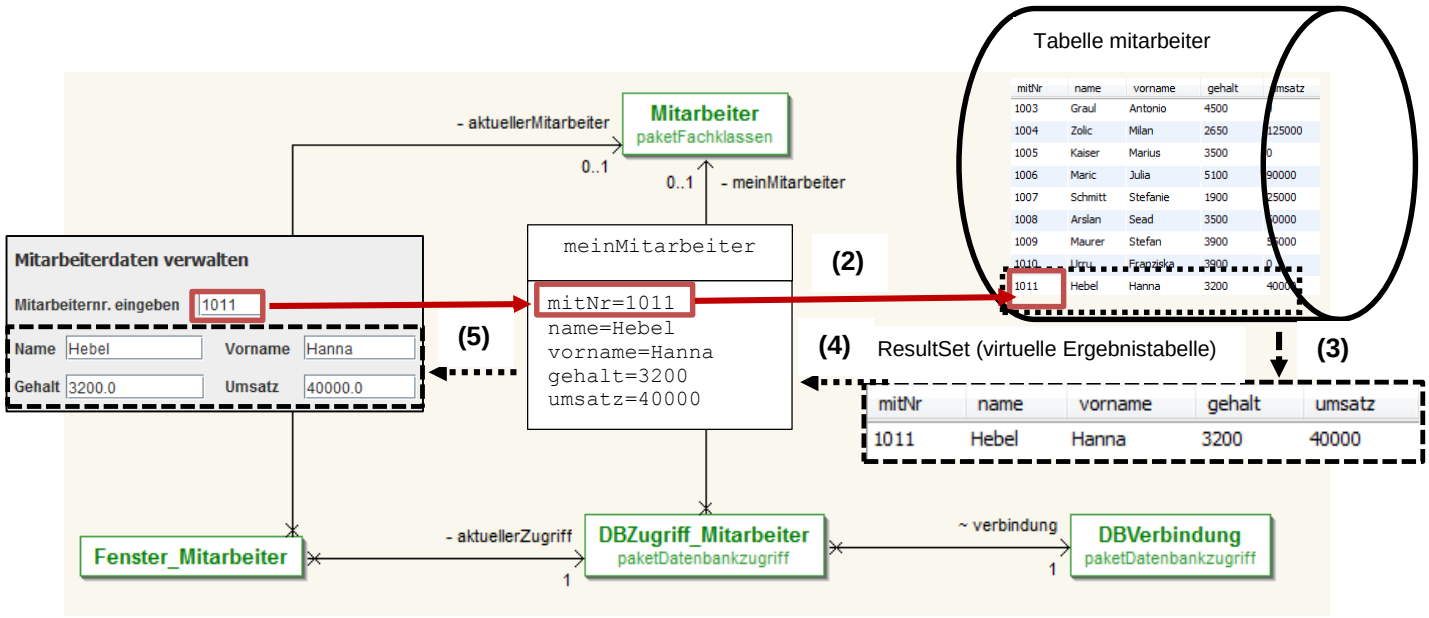
2.2.2 Mitarbeiter suchen und anzeigen

Problemstellung

Die gespeicherten Mitarbeiter sollen einzeln anhand der Mitarbeiternummer ausgewählt und in der GUI angezeigt werden können. Beispielsweise sollen die Daten der neu aufgenommenen Mitarbeiterin Hanna Weber (Mitarbeiternummer 1011) angezeigt werden können.

Ablauf des Datenbankzugriffs

In der nachfolgenden Abbildung wird veranschaulicht, wie ein Mitarbeiter anhand der eingegebenen Mitarbeiternummer in der Tabelle *mitarbeiter* der Datenbank *mitarbeiterdb* gesucht und seine Daten dann in der GUI angezeigt werden.



Die Mitarbeiternummer des zu suchenden Mitarbeiters wird

(1) in das Textfeld der GUI eingegeben. Wenn die Schaltfläche **`[Mitarbeiter suchen/anzeigen]`** gedrückt wird, wird die
(2) Mitarbeiternummer mithilfe des Zugriffsobjekts, das die SQL-Anweisung zum Suchen des Mitarbeiters enthält, sowie des Verbindungsobjekts zur Datenbank gesandt. Dort wird die SQL-Anweisung ausgeführt und liefert
(3) als Ergebnis eine Tabelle mit dem gefundenen Mitarbeiter. Die Ergebnisse aus Auswahlabfragen werden in virtuellen Ergebnistabellen, sogenannten *ResultSets,* gespeichert und können von dort aus in die
(4) Attribute des Objekts der Klasse «Mitarbeiter» übernommen werden, von wo aus sie in die
(5) Textfelder der GUI übertragen werden.

Arbeitsaufträge

1) Erklären Sie, was man unter einem *ResultSet*-Objekt versteht. Welchen Inhalt kann ein *ResultSet*-Objekt haben?

2) Implementieren Sie in der Klasse «Fenster_Mitarbeiter» die Methode **`private void anzeigenMitarbeiterdaten()`** mit der die Daten eines Mitarbeiters (Mitarbeiternummer, Name, Vorname, Gehalt und Umsatz) aus den Attributen des Objekts *aktuellerMitarbeiter* in die Textfelder der GUI übernommen werden. Vor der Ausgabe soll zunächst die Methode **`leeren()`** ausgeführt werden, damit die Ausgabe in eine leere Maske erfolgen kann.

3) Mit der Methode **`public Mitarbeiter sucheMitarbeiter(String pMitnr)`** der Klasse «DBZugriff_Mitarbeiter» wird die von der GUI übernommene Mitarbeiternummer zum Suchen des gewünschten Mitarbeiters verwendet.

   Erläutern Sie dazu den nachfolgend dargestellten Code der Methode.

```
//Methode zum Suchen eines Mitarbeiters
   public Mitarbeiter sucheMitarbeiter(String pMitnr)                    //(1)
   {
         ResultSet rsM;                                                  //(2)
         meinMitarbeiter = new Mitarbeiter();
         mSQL = "SELECT * FROM mitarbeiter ";                            //(3)
         mSQL = mSQL + "WHERE mitNr = '"+pMitnr+"';";
         verbindung.oeffneDB();                                          //(4)
         rsM = verbindung.lesen(mSQL);                                   //(5)
         try
         {
               rsM.next();                                                 //(6)
               meinMitarbeiter.setMitarbeiternummer(rsM.getInt("mitNr"));  //(7)
               meinMitarbeiter.setName(rsM.getString("name"));
               meinMitarbeiter.setVorname(rsM.getString("vorname"));
               meinMitarbeiter.setGehalt(rsM.getDouble("gehalt"));
               meinMitarbeiter.setUmsatz(rsM.getDouble("umsatz"));
         }
         catch(SQLException err)
         {
               meinMitarbeiter = null;                                   //(8)
         }
         verbindung.schliesseDB();                                       //(9)
         return meinMitarbeiter;                                         //(10)
   }
```

4) Die Methode `public void` **`suchenMitarbeiter()`** der Klasse «Fenster_Mitarbeiter» wird durch das Ereignis zur Schaltfläche **`[Mitarbeiter anlegen`**] ausgeführt.
   Zunächst wird das Objekt *aktuellerMitarbeiter* der Klasse «Mitarbeiter» mit dem Standardkonstruktor erzeugt (das Objekt *aktuellerMitarbeiter* ist bereits deklariert). Dann wird für dieses Objekt die Methode **`sucheMitarbeiter(…)`** ausgeführt. Als Übergabewert wird der Methode der Inhalt des Textfelds *txtMaNr* mitgegeben.
   Wenn das von der Methode **`sucheMitarbeiter(…)`** als Ergebnis zurückgegebene Objekt keinen Inhalt hat (**`aktuellerMitarbeiter ==null`**), wird ein Meldefenster mit dem Text *„keinen Mitarbeiter gefunden“* angezeigt. Hat das zurückgegebene Objekt einen Inhalt, wird die Methode **`anzeigenMitarbeiterdaten`** (siehe Aufgabe 2) ausgeführt und die gefundenen Mitarbeiterdaten werden in der GUI angezeigt.

   Erstellen Sie für die beschriebene Aufgabe ein Struktogramm und erstellen Sie daraus den Javacode der Methode.

5) Warum ist es sinnvoll, für das Anzeigen der Mitarbeiterdaten (siehe Aufgabe 2) eine eigene Methode zu erstellen?

Lösungshinweise

1) Auswahlabfragen, die mit der Methode **`public ResultSet lesen(String pSQL)`** des Verbindungsobjekts *verbindung* der Klasse «DBVerbindung» ausgeführt werden, liefern als Ergebnis virtuelle Tabellen, die in Objekte der Klasse «ResultSet» übernommen werden. Der Inhalt eines *ResultSet-Objekts* kann der zugrunde gelegten Abfrage keine, eine oder mehrere Zeilen (Datensätze) umfassen.

Problemstellung	Sql-Abfrage	Zeilenzahl d. Ergenistabelle
Beispiel 1: den Mitarbeiter mit der Mitarbeiternummer 1011 suchen.	`Select * from Mitarbeiter Where mitNr=1011`	Keine Zeile mitNr | name | vorname | gehalt | umsatz oder eine Zeilen mitNr | name | vorname | gehalt | umsatz 1011 | Hebel | Hanna | 3200 | 40000
Beispiel 2: alle Mitarbeiter der GeLa GmbH	`Select * from Mitarbeiter`	Eine oder mehrere Zeilen mitNr | name | vorname | gehalt | umsatz 1001 | Huber | Franz | 2500 | 90000 1002 | Schimpf | Carola | 2900 | 45000 1003 | Graul | Antonio | 4500 | 0 1004 | Zolic | Milan | 2650 | 125000

Die Klasse «ResulSet» stellt eine Vielzahl von Methoden zur Verfügung, mit denen die Spalten des *ResultSet*-Objekts angesprochen werden können, um die Zeilen auszuwerten.

Beispiele:

```
ResultSet rsM  // Deklarieren Objekts
               //rsM der Klasse ResultSet
rsM.next()     // Mit der Methode next()
               // wird die erste Zeile
               // aktiviert.
               // Sie liefert true/false
               // je nachdem, ob die
               // Zeile einen Inhalt hat
               // oder leer ist.

String name = rsM.getString("name");
Double gehalt = rsM.getDouble("gehalt");
:
```

mitNr	name	vorname	gehalt	umsatz
1001	Huber	Franz	2500	90000
1002	Schimpf	Carola	2900	45000
1003	Graul	Antonio	4500	0
1004	Zolic	Milan	2650	125000

Mit der Methode getXXXX() können Attributswerte aus dem *ResultSet*-Objekt *rs* ausgelesen werden

- `String var1 =rs.getString("Spalte1")`
- `Int var2=rs.getInt("Spalte2")`
- `Double var3 =rs.getDouble("Spalte3")`

Soll ein ResultSet-Objekt mit mehreren Zeilen durchsucht werden verwendet man dazu eine Schleife:

```
While rs.next()
      {
      var = rs.getXXXX("Spalte";
      :
      }
```

mitNr	name	vorname	gehalt	umsatz
1001	Huber	Franz	2500	90000
1002	Schimpf	Carola	2900	45000
1003	Graul	Antonio	4500	0
1004	Zolic	Milan	2650	125000

Solange die Methode **`next()`** true liefert, hat sie einen Inhalt.

2) Quellcode der Methode `private void anzeigenMitarbeiterdaten()`

```
private void anzeigenMitarbeiterdaten()
{
leeren();
txtMaNr.setText(Integer.toString(aktuellerMitarbeiter.getMitarbeiternummer()));
txtName.setText(aktuellerMitarbeiter.getName());
txtVorname.setText(aktuellerMitarbeiter.getVorname());
txtGehalt.setText(Double.toString(aktuellerMitarbeiter.getGehalt()));
txtUmsatz.setText(Double.toString(aktuellerMitarbeiter.getUmsatz()));
}
```

3)

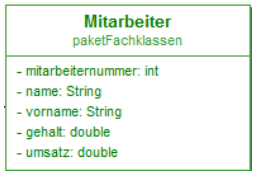
(1)	Die Methode *sucheMitarbeiter* benötigt als Übergabewert die eingegebene Mitarbeiternummer und gibt als Ergebnis ein Objekt der Klasse «Mitarbeiter» zurück.
(2)	Für die Ergebnistabelle (der Mitarbeiterobjekte) wird das Objekt *rsM* der Klasse «ResultSet» deklariert.
(3)	Die SQL-Anweisung zum Suchen des Mitarbeiters mit der übergebenen Mitarbeiternummer wird in der Variablen *mSQL* gespeichert.
(4)	Die Datenbankverbindung wird mit der Methode **`void oeffneDB()`** des Objekts *verbindung* der Klasse «DBVerbindung» geöffnet.
(5)	Die Methode **`lesen(mSQL)`** wird ausgeführt. Sie fordert als Übergabewert den in der Variablen *mSQL* gespeicherten SQL-Befehl und liefert das Ergebnis in das *ResultSet*-Objekt *rsM*. Der Inhalt des *ResultSet*-Objekts kann keine, eine oder mehrere Zeilen umfassen.
(6)	Die erste Zeile des *ResultSet*-Objekts *rsM* wird aktiviert. Wenn die Methode **`next()`** den Wert *true* zurückgibt, wurde ein Mitarbeiter gefunden.
(7)	. Die Attributswerte werden aus dem ResultSet-Objekt *rsM* ausgelesen und in die Attribute des Objekts *meinMitarbeiter* der Klasse «Mitarbeiter» übertragen.

Attributsbezeichnungen aus der Tabelle *mitarbeiter* der Datenbank

mitNr	name	vorname	gehalt	umsatz
1001	Huber	Franz	2500	90000

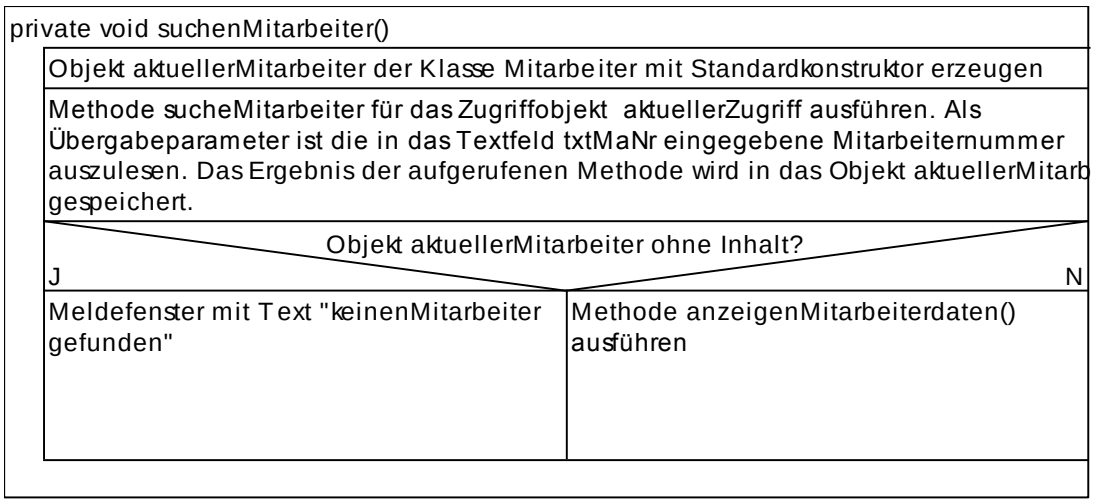
```
meinMitarbeiter.setMitarbeiternummer(rsM.getInt("mitNr"));
meinMitarbeiter.setName(rsM.getString("name"));
meinMitarbeiter.setVorname(rsM.getString("vorname"));
meinMitarbeiter.setGehalt(rsM.getDouble("gehalt"));
meinMitarbeiter.setUmsatz(rsM.getDouble("umsatz"));
```

Zugriffsmethoden aus der Klasse «Mitarbeiter»



(8)	Wenn im *try*-Block ein Fehler auftritt, d. h. wenn die übergebene Mitarbeiternummer nicht gefunden wurde, wird dieser im catch-Block abgefangen. Das Objekt *meinMitarbeiter* erhält dann den Wert ***null*** (kein Inhalt).
(9)	Datenbankverbindung mit der Methode **`void schliesseDB`** des Objekts *verbindung* der Klasse »DBVerbindung «schließen
(10)	Das Objekt *meinMitarbeiter* als Ergebnis zurückgeben. Es enthält entweder die gefundenen und aus dem *ResultSet*-Objekt *rsM* übernommenen Daten (*try*-Block) oder keinen Inhalt (*catch*-Block).

4) Struktogramm zur Methode **`public void suchenMitarbeiter()`** der Klasse «Fenster_Mitarbeiter»



Javacode der Methode

```
// DBZugriff
	public void suchenMitarbeiter()
	{
	aktuellerMitarbeiter = new Mitarbeiter();
	aktuellerMitarbeiter = aktuellerZugriff.sucheMitarbeiter(txtMaNr.getText());
	if (aktuellerMitarbeiter == null)
	{
		JOptionPane.showMessageDialog(null,"keinen Mitarbeiter gefunden");
	}
	else
	{
		anzeigenMitarbeiterdaten();
	}
	}
```

```
private void anzeigenMitarbeiterdaten()
{
    txtName.setText(aktuellerMitarbeiter.getName());
    txtVorname.setText(aktuellerMitarbeiter.getVorname());
    txtGehalt.setText(Double.toString(aktuellerMitarbeiter.getGehalt()));
    txtUmsatz.setText(Double.toString(aktuellerMitarbeiter.getUmsatz()));
}
```

5) Die Anweisungen zum Anzeigen der Mitarbeiterdaten wird in die Methode **`private void`** `anzeigenMitarbeiterdaten()` ausgelagert. Die Methode wird dann von der ereignisgesteuerten Methode **`private void`** `suchenMitarbeiter()` (siehe Abb. Aufgabe 4) aufgerufen. Dieser modulare Programmaufbau hat den Vorteil, dass der Programmcode

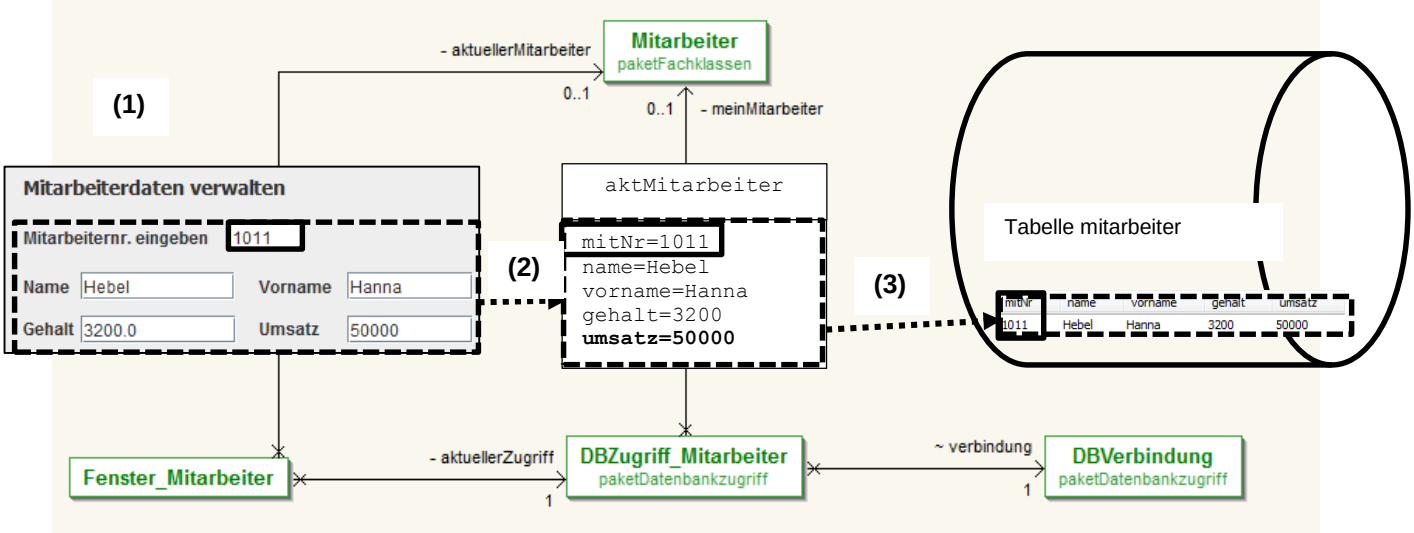
übersichtlicher und einfacher zu pflegen ist. Außerdem kann die Methode `anzeigenMitarbeiterdaten()` mehrmals aufgerufen werden.

2.2.3 Änderungen der Mitarbeiterdaten speichern

Problemstellung

Der Umsatz der neu erfassten Mitarbeiterin Hanna Hebel (Mitarbeiternummer 1011) beträgt nicht 30000,00 Euro, sondern 50000,00 Euro. Berichtigen Sie den Umsatz der Mitarbeiterin

Ablauf des Datenbankzugriffs:



(1) Die zu ändernden Daten des Mitarbeiters sind gesucht und werden in der GUI angezeigt (siehe Punkt 2.2.2). Änderungen können jetzt in der GUI vorgenommen werden.

(2) Wird die Schaltfläche **`[Mitarbeiter ändern]`** gedrückt, so wird der aktuelle Inhalt der Textfelder der GUI in das Objekt *aktuellerMitarbeiter* der Klasse «Fenster_Mitarbeiter» übertragen und

(3) mithilfe des Objekts *aktuellerZugriff* der Klasse «DBZugriff_Mitarbeiter» wird die Änderung festgelegt (SQL-Anweisung update) sowie mit der Methode *aendern(mSQL)* des Verbindungsobjekts *verbindung* der Klasse «DBVerbindung» auf der Datenbank ausgeführt (*executUpdate*).

Arbeitsaufträge

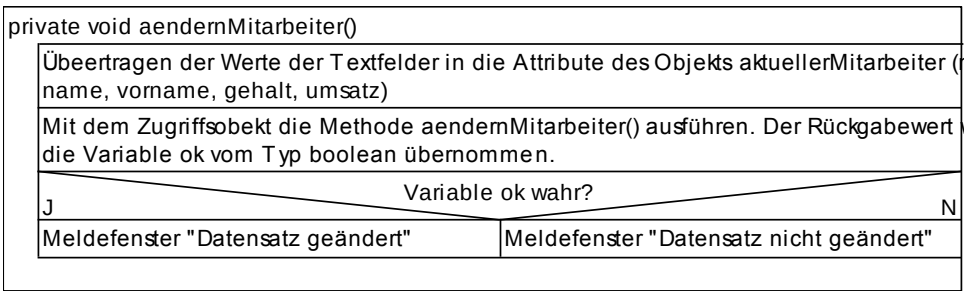
1) Die Änderungen des angezeigten Mitarbeiters sollen mit der Methode `public boolean aendereMitarbeiter()` der Klasse «DBZugriff_Mitarbeiter» beschrieben und dann mit dem Objekt *verbindung* der Klasse «DBVerbindung» auf der Datenbank ausgeführt werden.

   Erstellen Sie die Methode `public boolean aendereMitarbeiter()` in der Klasse «Fenster_Mitarbeiter» nach der folgenden Beschreibung:

   - Um Fehler abzufangen wird eine **`try…catch`**-Struktur verwendet.
   - Im *try*-Block wird die SQL-Anweisung zum Aktualisieren der Tabelle Mitarbeiter als *String* in die Variable *mSQL* gespeichert. Alle Felder der Tabelle sollen aktualisiert werden. Die Werte für die zu aktualisierenden Felder der Tabelle werden aus dem Objekt *meinMitarbeiter* ausgelesen.
   - Datenbankverbindung mit der Methode **`void schliesseDB()`** des Objekts *verbindung* der Klasse «DBVerbindung» öffnen.
   - Methode *aendern(pSQL)* mit dem Objekt *verbindung* ausführen. Die Methode fordert als Übergabewert die SQL-Anweisung, die in der Variablen *mSQL* gespeichert ist.

- Im *catch*-Block erhält die Variable *ok* den Wert *false*.
- Im Anschluss zur *try…catch*-Struktur wird die Methode *schliesseDB()* des Objekts *verbindung* ausgeführt und der Wert der Variablen *ok* zurückgegeben.

2) Mit dem nachfolgenden Struktogramm können die in der GUI angezeigten Daten des Mitarbeiters geändert werden. Wenn der Button **`[Mitarbeiter ändern]`** ausgelöst wird, wird die Methode **`private void aendernMitarbeiter()`** der ***`Klasse`*** «Fenster_Mitarbeiter» ausgeführt.
Codieren Sie das nachfolgende Struktogramm



Lösungshinweise

1) Methode `public boolean aendereMitarbeiter()`

```
public boolean aendereMitarbeiter() {
try {
mSQL = "UPDATE mitarbeiter SET name = '" + meinMitarbeiter.getName();
mSQL += "',vorname = '" + meinMitarbeiter.getVorname() + "',gehalt = '";
mSQL += meinMitarbeiter.getGehalt() + "', umsatz = '" + meinMitarbeiter.getUmsatz();
mSQL += "' WHERE mitNr = '" + meinMitarbeiter.getMitarbeiternummer() + "';";
verbindung.oeffneDB();
verbindung.aendern(mSQL);
}
catch (Exception e) {
       ok = false;
}
verbindung.schliesseDB();
return ok; //
}
```

2) Methode `private void aendernMitarbeiter()`

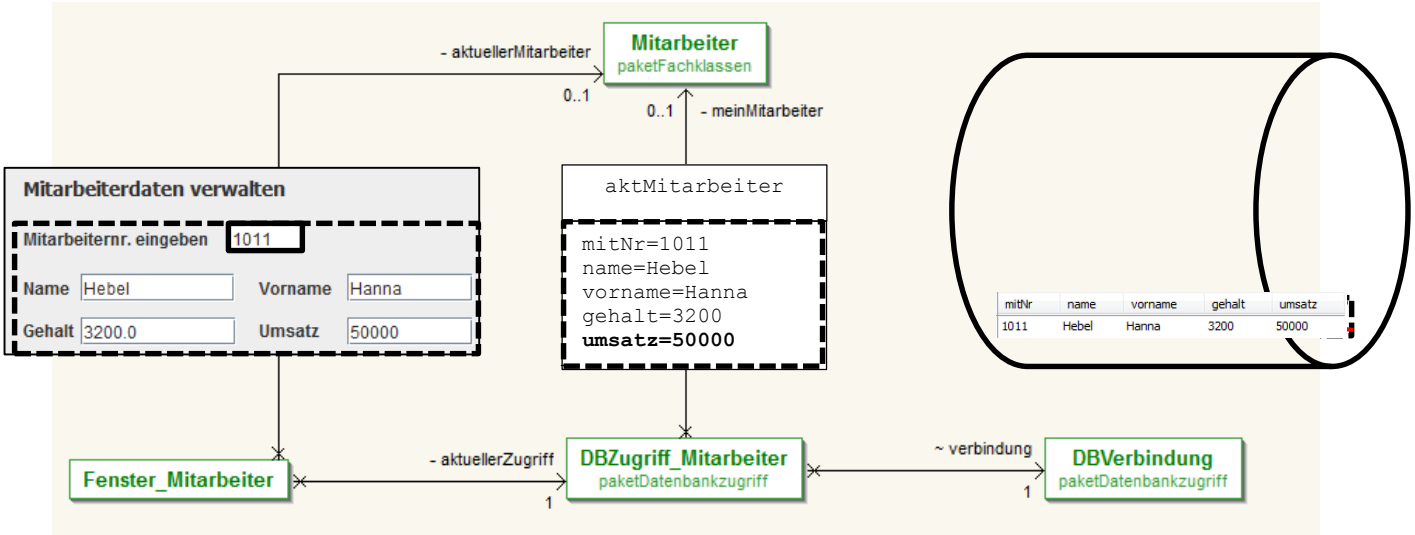
```
private void aendernMitarbeiter()   {
       // Übertragung der geänderten Daten aus den Textfeldern in das Objekt
       aktuellerMitarbeiter.setName(txtName.getText());
       aktuellerMitarbeiter.setVorname(txtVorname.getText());
       aktuellerMitarbeiter.setGehalt(Double.parseDouble(txtGehalt.getText()));
       aktuellerMitarbeiter.setUmsatz(Double.parseDouble(txtUmsatz.getText()));
       // Methode aendereMitarbeiter für das Objekt aktuellerZugriff ausführen und das
       //Ergebnis in die Variable ok (boolean) übernehmen
       boolean ok = aktuellerZugriff.aendereMitarbeiter();
       // Rückgabewert der methode boolean aendereMitarbeiter() auswerten
       if (ok)
       {
              JOptionPane.showMessageDialog(null,"Datensatz geändert!");
       }
       else
       {
              JOptionPane.showMessageDialog(null,"Datensatz nicht geändert!");
       }
   }
```

2.2.4 Einen Mitarbeiter löschen

Problemstellung

Nachdem die Mitarbeiterin Hanna Hebel (Mitarbeiternummer 1011) mit Hilfe der eingegebenen Artikelnummer gesucht und angezeigt wurde (Button **`[Mitarbeiter suchen/anzeigen])`** soll sie, falls gewünscht, gelöscht werden können.

Ablauf des Datenbankzugriffs:



(1) Die zu löschenden Daten der Mitarbeiterin sind gesucht und werden in der GUI angezeigt (siehe Punkt 2.2.2).

(2) Wird die Schaltfläche **`[Mitarbeiter löschen]`** gedrückt, wird die Mitarbeiternummer des zu löschenden Mitarbeiters mithilfe des Objekts *aktuellerZugriff* der Klasse «DBZugriff_Mitarbeiter» (SQL-Anweisung *delete*) und dem Verbindungsobjekt *verbindung* der Klasse «DBVerbindung» auf der Datenbank ausgeführt (*executUpdate*).

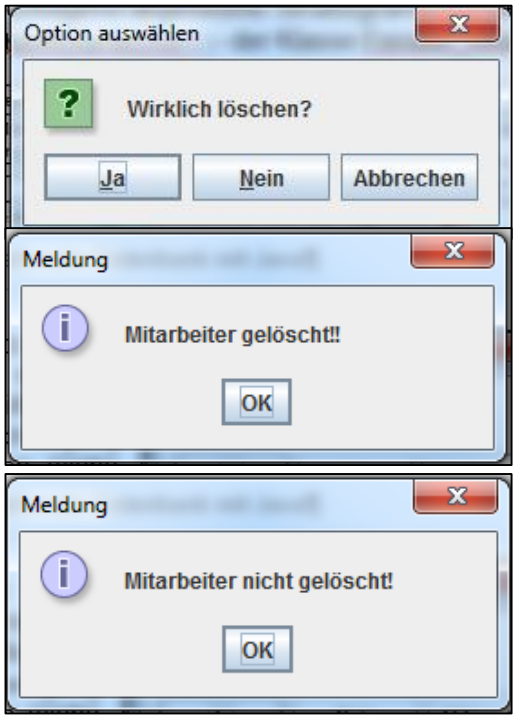
Arbeitsaufträge

1) Das nachfolgende Struktogramm beschreibt, wie mit der Methode **`public boolean loescheMitarbeiter()`** in der Klasse «DBZugriff_Mitarbeiter» die SQL-Anweisung zum Löschen des Artikels zur Datenbank gesandt und ausgeführt wird.

private void loeschenMitarbeiter()
Fehlerbearbeitung mit try...catch try-Block
SQL-Anweisung zum Löschen des Artikels mit der Mitarbeiternummer des angezeigten Mitarbeiters (Objekt meinMitarbeiter)in der Variablen mSQL speichern
Datenbankverbindung mit dem Verbindungsobjekt öffnen
Methode aendern(pSQL) mit dem Verbindungsobjekt ausführen. Als Übergabewert wird Methode der SQL-Befehl (mSQL) mitgegeben.
Variable ok erhält den Wert false
catch-Block Datenbankverbindung mit dem Objekt verbindung schließen
Wert der Variablen ok zurückgeben.

Codieren Sie das Struktogramm!

(3) Informieren Sie sich über den Einsatz eines Dialogfensters der Klasse «JOptionPane» mit der Klassenmethode **`showConfirmDialog`**. Mithilfe des Dialogfensters soll vor dem eigentlichen Löschvorgang eine „Sicherheitsabfrage“ vorgeschaltet werden (siehe nebenstehende Abbildung).



Wenn die Schaltfläche **`[Ja]`** gedrückt wird, soll die Methode **`loescheMitarbeiter()`** des Objekts *aktuellerZugriff* der Klasse «DBZugriff_Mitarbeiter» ausgeführt werden. Als Bestätigung soll dann ein Meldefenster angezeigt werden (siehe nebenstehende Abbildung):

Wenn die Schaltfläche **`[Nein]`** gedrückt wird, soll ein Meldefenster angezeigt werden (siehe nebenstehende Abbildung) und die Textfelder der GUI sollen geleert werden (Methode **`leeren()`**).

Wenn die Schaltfläche **`[Abbrechen]`** gedrückt wird, soll die Anzeige des Mitarbeiters erhalten bleiben und das Textfeld „Name“ den Cursor bekommen.

Codieren Sie das nachfolgend abgebildete Struktogramm für die Methode **`private void`** `loeschenMitarbeiter()` der Klasse «Fenster_Mitarbeiter»

<table>
<tr><td colspan="3">private void loescheMitarbeiter()</td></tr>
<tr><td colspan="3">Eingabedialog mit der Meldung "Wirklich löschen?" anzeigen und den Rückgabewert in die Variabl
(Typ int) übernehmen</td></tr>
<tr><td>Fall 0:</td><td colspan="2">Fallunterscheidung für mJN</td></tr>
<tr><td>Meldefenster "Mitarbeiter gelöscht" anzeigen</td><td>Fall 1:
Meldefenster "Mitarbeiter nicht gelöscht" anzeigen</td><td>Fall 2:
Cursor auf das Textfeld txtName setzen</td></tr>
<tr><td>Methode leeren() aufrufen</td><td>Methode leeren() aufrufen</td><td></td></tr>
</table>

Lösungshinweise

1) Methode boolean loescheMitarbeiter()

```
public boolean loescheMitarbeiter() {
try {
// (1) SQL-Anweisung zum Löschen des Artikels mit der eingegebenen Artikelnummer
mSQL = "DELETE FROM Mitarbeiter WHERE mitNr =
'"+meinMitarbeiter.getMitarbeiternummer()+"';";
verbindung.oeffneDB();       // (1) Datenbankverbindung öffnen
verbindung.aendern(mSQL);    // (2) Datenbank mit der Löschanweisung aktualisieren

ok = true;
}
catch (Exception e)
{
ok = false;
}
verbindung.schliesseDB();    // (3) Datenbankverbindung schließen
return ok;                   // (4)
}
```

2) Methode private void loeschenMitarbeiter()

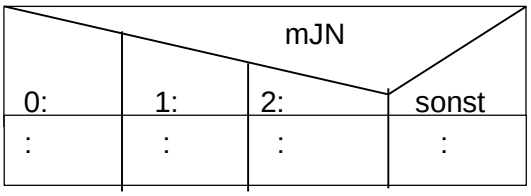
```
private void loeschenMitarbeiter()
{
// Eingabedialogfenster
int mJN = JOptionPane.showConfirmDialog(null, "Wirklich löschen?"); //(3)
System.out.println(mJN);
// Fallunterscheidung für die Variable mJN
switch (mJN)
{
case 0:
       aktuellerZugriff.loescheMitarbeiter();

       JOptionPane.showMessageDialog(
       null,
       "Mitarbeiter gelöscht!!");
       leeren();

       break;
case 1:
       JOptionPane.showMessageDialog(
       null,
       "Mitarbeiter nicht gelöscht!");
       leeren();
       break;

case 2:
       txtName.requestFocus();
       break;
   }
```

Erläuterung zur Fallunterscheidung:

Das Dialogfenster übergibt an die Variable *mJN* den Wert 0, 1 oder 2. Für die Auswertung eignet sich die Kontrollstruktur der Fallunterscheidung (Mehrfachauswahl).

Für den im *switch*-Block angegebenen Ausdruck (hier die Variable mJN) werden nacheinander die angegebenen Fälle geprüft. Da die Dialogbox keine anderen Werte als 0, 1 oder 2 zurück gibt, kann der „Sonst“-Block (default) entfallen.

Struktogramm:



Javacode:

```
switch (mJN) {
case 0:
       :
       break;

case 1:
       :
       break;
case 2:
       :
       break;

default:
        :
       break;
}
```

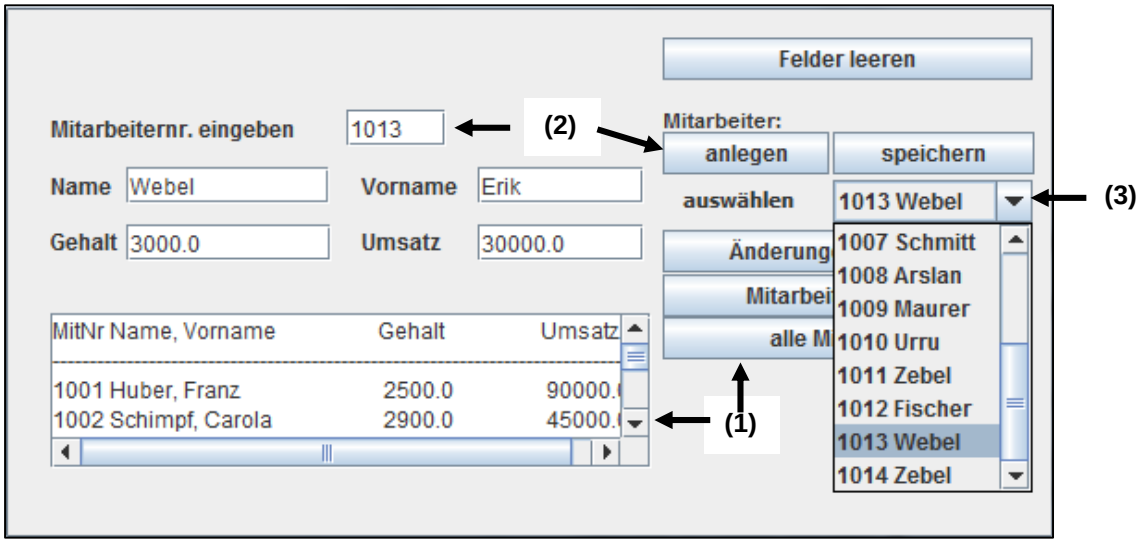
2.3 Listenbearbeitung

Problemstellung

Bisher wurde immer nur jeweils auf einzelne in der Datenbank gespeicherte Mitarbeiter zugegriffen, um diese zu bearbeiten.
Jetzt sollen mehrere/alle Mitarbeiter aus der Datenbank zur Bearbeitung zur Verfügung stehen.

Wie in der nachfolgenden Abbildung veranschaulicht wird, werden alle Datensätze der Tabelle Mitarbeiter benötigt, wenn

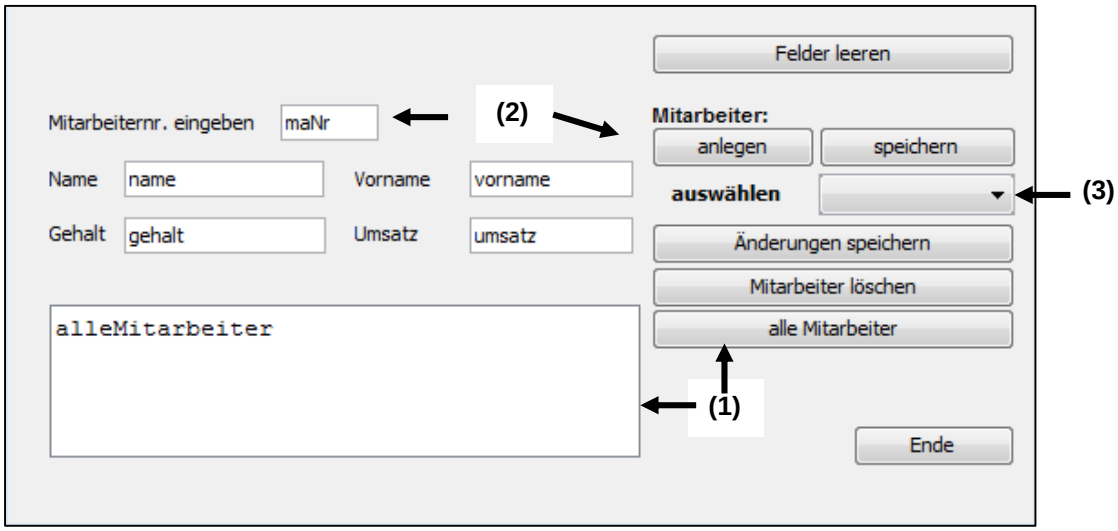
(1) alle Mitarbeiter der GeLa GmbH im Textbereich aufgelistet werden sollen
(2) ein neuer Mitarbeiter erfasst wird. Zur Ermittlung der Mitarbeiternummer des neu aufzunehmenden Mitarbeiters muss aus den vorhandenen Mitarbeiternummern die nächste ermittelt werden.
(3) der zu bearbeitende Mitarbeiter aus einem Kombinationsfeld, in dem alle Mitarbeiter aufgelistet sind, ausgewählt und angezeigt werden soll.



Für diesen Zweck wurden in die GUI bereits

(1) ein Textbereich sowie die Schaltfläche [**`alleMitarbeiter`**], mit der die Auflistung aller Mitarbeiter im Textbereich ausgelöst werden kann,
(2) die Schaltflächen **`[anlegen]`** und **`[speichern]`**
(3) ein Kombinationsfeld zum Auswählen des zu bearbeitenden Mitarbeiters

aufgenommen.



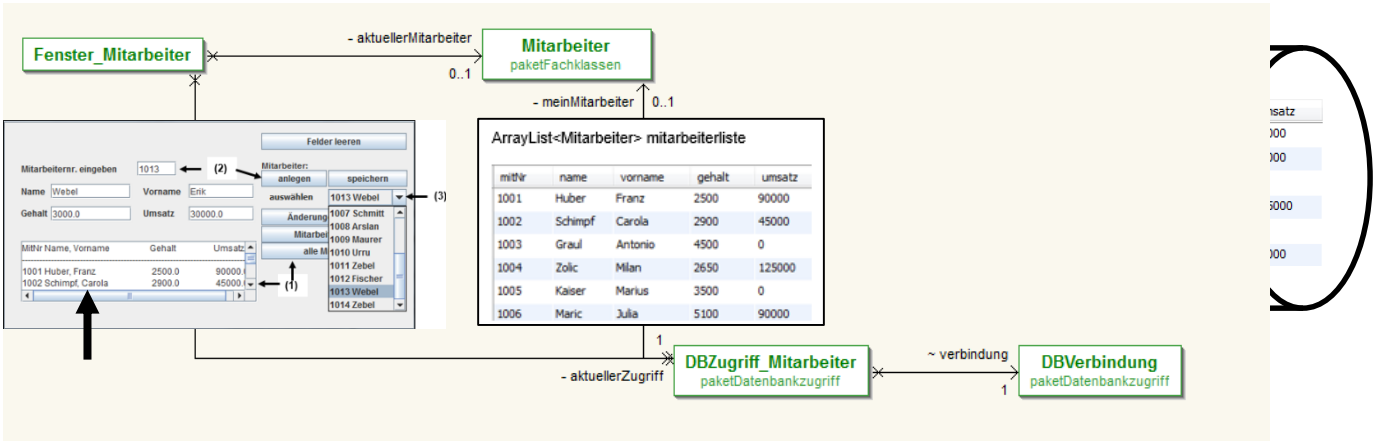
Alle Datensätze bereitstellen

Problemstellung

Für alle Aufgabenstellungen, die sich auf mehrere/alle Datensätze der Tabelle der Datenbank beziehen, werden alle Datensätze in einer Liste bereitgestellt.

Die untenstehende Abbildung veranschaulicht, wie

(1) die ausgewählten Daten der Tabelle in ein ResultSet-Objekt *rsM* übernommen und
(2) aus dem ResultSet-Objekt in eine Liste (*mitarbeiterliste* = Objekt der Klasse «ArrayList») übertragen werden.
(3) In der GUI wird dann diese übernommene Liste je nach Aufgabenstellung verarbeitet.



Aufgabe

Die Methode `public ArrayList<Mitarbeiter> mitarbeiterliste_db()` soll eine Liste aller Mitarbeiter als Ergebnistabelle zurückgeben.

Nach dem Aufruf der Methode sind folgende Schritte zu codieren

(1) Das Objekt mitarbeiterliste der Klasse «ArrayList» mit dem Standardkonstruktor muss erzeugt werden. Das Objekt mitarbeiterliste darf nur Objekte vom Typ «Mitarbeiter» *aufnehmen.*
(2) Das Objekt *rsM* der Klasse «ResultSet» zur Aufnahme der Abfrageergebnisse ist zu deklarieren.

(3) Die SQL-Anweisung, mit der alle Mitarbeiter der Tabelle *mitarbeiter* ausgewählt werden, ist als String in die Variable *mSQL* (Typ String) zu speichern
(4) Mit der Methode *oeffneDB()* des Objekts *verbindung* der Klasse «DBVerbindung» ist die Datenbankverbindung herzustellen.
(5) Mit dem Objekt *verbindung* ist die Methode *lesen(mSQL)* auszuführen. Die Methode verlangt als Übergabewert die in der Variablen *mSQL* gespeicherte SQL-Abfrage. Das Ergebnis der Methode ist in das *ResultSet*-Objekt *rsM* aufzunehmen.
(6) Eine **`try`**…**`catch`**-Struktur für auftretende Fehler ist zu erstellen. Im *try*-Block
(7) muss in einer *while*-Schleife immer jeweils die nächste Zeile des *ResultSet*-Objektes *rsM* aktiviert werden.
(8) Für die aus dem *ResultSet*-Objekt auszulesenden Mitarbeiterdaten muss das Objekt *meinMitarbeiter* der Klasse «Mitarbeiter» mit Hilfe des Standardkonstruktors erzeugt werden.
(9) Alle Mitarbeiterdaten sind aus den Spalten des *ResultSet*-Objekts *rsM* auszulesen und in die Attribute des Objekts *meinMitarbeiter* zu übertragen.
(10) Das Objekt *meinMitarbeiter* kann in der Mitarbeiterliste (Objekt *mitarbeiterliste* der Klasse «ArrayList» hinzugefügt werden.
(11) Im *catch*-Block der *try…catch*-Struktur soll lediglich eine Meldung im Konsolenfenster bei auftretenden Fehlern erscheinen.
(12) Die erstellte Mitarbeiterliste (Objekt *mitarbeiterliste* der Klasse «ArrayList» wird zurückgegeben.

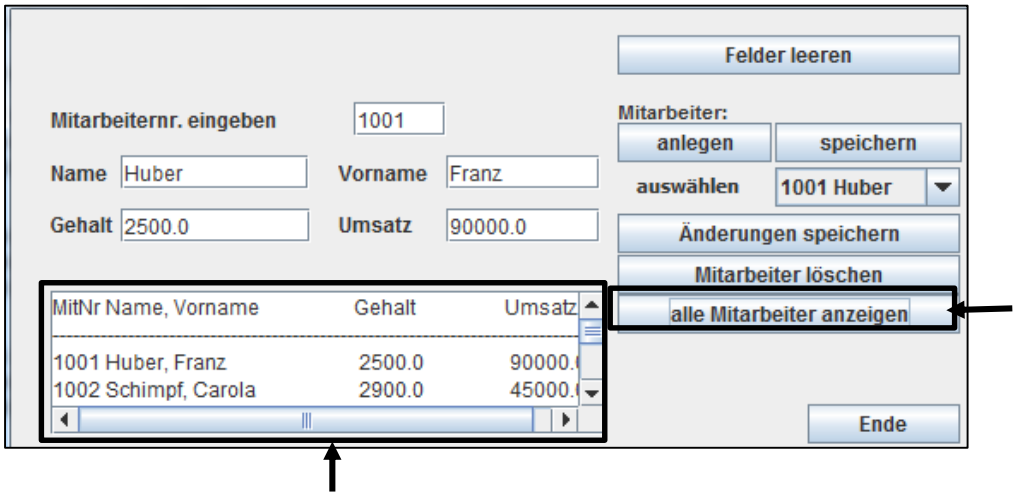
Lösungsvorschlag für die Methode `public ArrayList<Mitarbeiter> mitarbeiterliste_db()` der Klasse DBZugriff_Mitarbeiter

```
public ArrayList<Mitarbeiter> mitarbeiterliste_db()
{
ArrayList<Mitarbeiter> mitarbeiterliste = new ArrayList<Mitarbeiter>();  //(1)
ResultSet rsM;                                                           //(2)
mSQL = "SELECT * FROM mitarbeiter";                                      //(3)
verbindung.oeffneDB();                                                   //(4)
rsM = verbindung.lesen(mSQL);                                            //(5)
try {                                                                    //(6)
      while (rsM.next()) {                                               //(7)
            meinMitarbeiter = new Mitarbeiter();                         //(8)
            meinMitarbeiter.setMitarbeiternummer(rsM.getInt("mitNr"));   //(9)
            meinMitarbeiter.setName(rsM.getString("name"));
            meinMitarbeiter.setVorname(rsM.getString("vorname"));
            meinMitarbeiter.setGehalt(rsM.getDouble("gehalt"));
            meinMitarbeiter.setUmsatz(rsM.getDouble("umsatz"));
            mitarbeiterliste.add(meinMitarbeiter);                       //(10)
      }
} catch (SQLException e) {                                               //(11)
      System.out.println("keine Mitarbeiter gefunden");
}
return mitarbeiterliste;                                                 //(12)
}
```

2.3.1 Alle Mitarbeiter im Textbereich anzeigen

Problemstellung

Wird die Schaltfläche **`[alle Mitarbeiter anzeigen]`** gedrückt, so wird der Textbereich *alleMitarbeiter* sichtbar gemacht und die Mitarbeiter der GeLa GmbH werden im Textbereich aufgelistet (siehe nachfolgende Abbildung).



Arbeitsaufträge

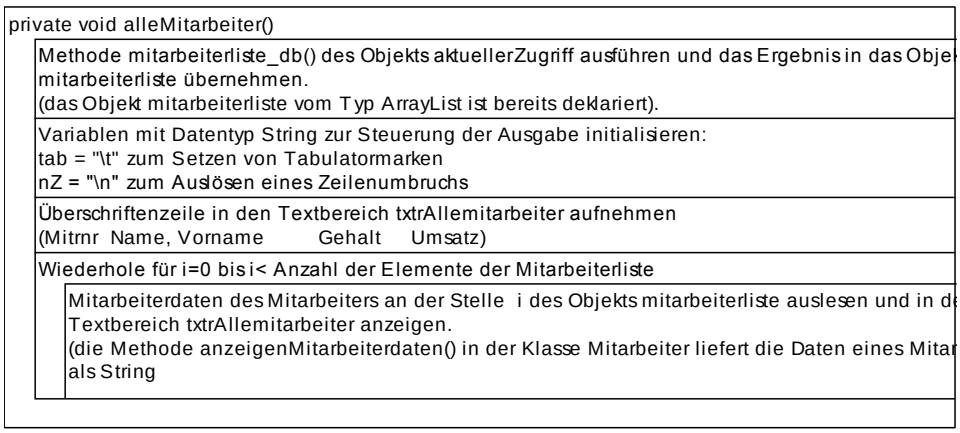
1) Wie erreichen Sie, dass Scrollbalken erscheinen, sobald der Inhalt des Textbereichs dessen Größe übersteigt?

2) Wenn die Schaltfläche **`[alleMitarbeter anzeigen]`** das Standardereignis erhält, sollen
   - der Textbereich (Objekt *txtrAllemitarbeiter* der Klasse «JTextArea») sowie das Objekt *scrollPane* der Klasse «JScrollPane» sichbar gemacht und
   - die Methode **`alleMitarbeiter()`** ausgeführt werden.

   Nehmen Sie das Standardereignis mit den beschriebenen Reaktionen in die GUI auf und erläutern Sie dessen Steuerung.

3) Mit der Methode **`private void alleMitarbeiter()`** werden alle Mitarbeiter aus der Tabelle *mitarbeiter* der Datenbank *mitarbeiterdb* im Textbereich *txtrAllemitarbeiter* aufgelistet.

   Folgendes Struktogramm wurde dafür entworfen:

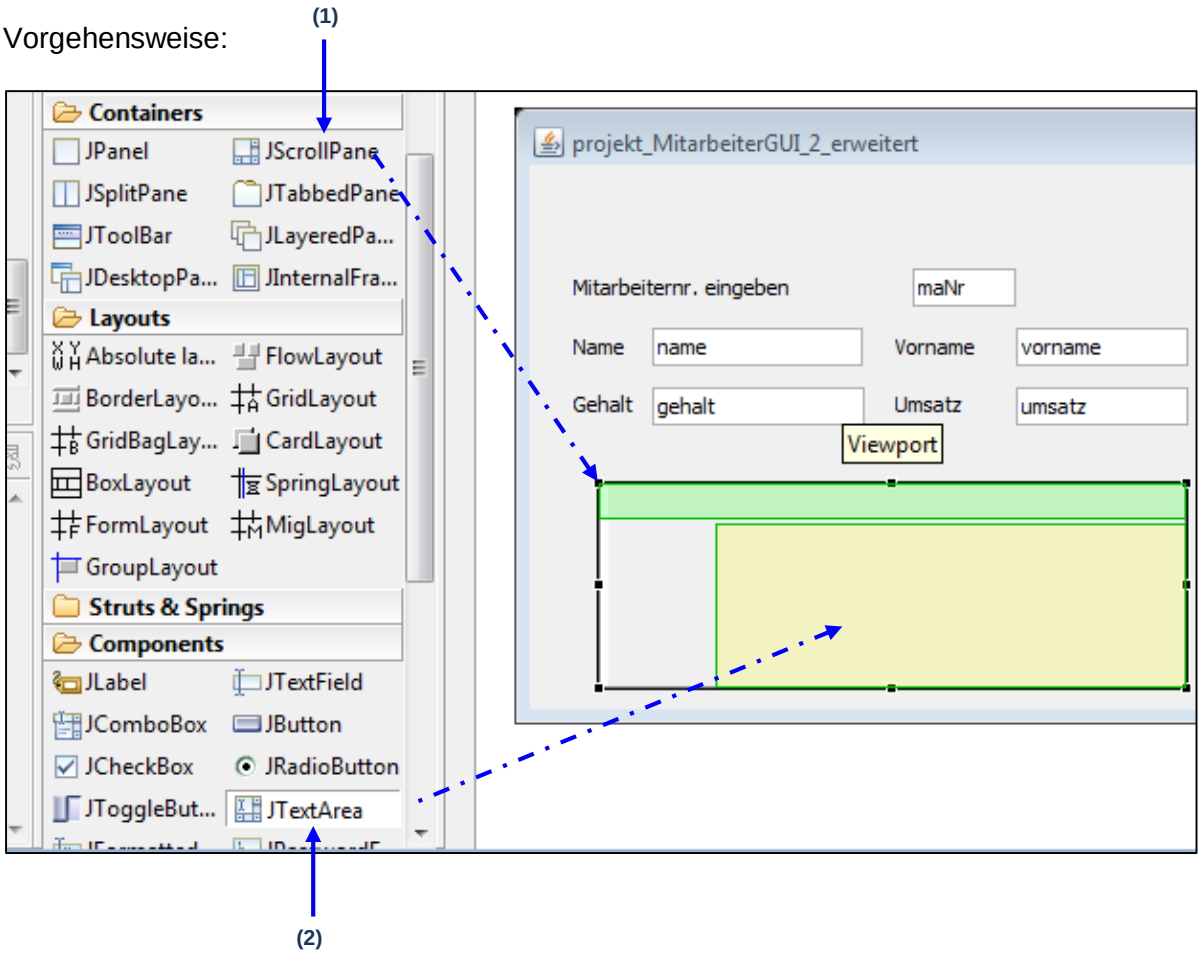


   Codieren Sie die Methode **`private void alleMitarbeiter()`**

Lösungshinweise

1) Das Objekt *txtrAllemitarbeiter* der Klasse «JTextArea» wird in ein Objekt der Klasse «JScrollpane» eingebettet.

Vorgehensweise:



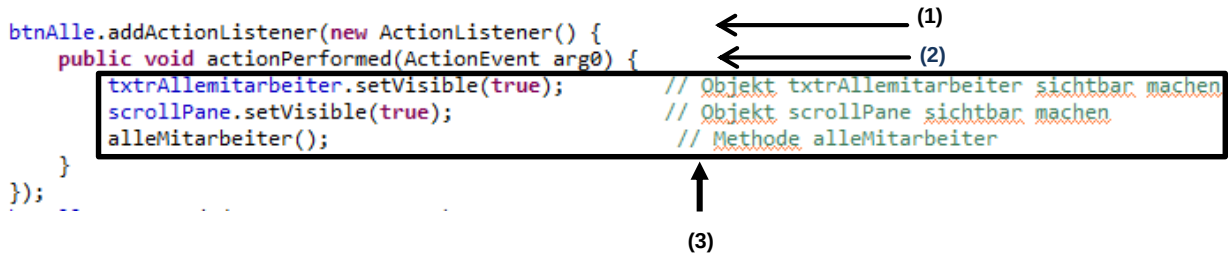
(1) Container-Objekt der Klasse «JScrollPane» anklicken und in der Zeichenfläche in der gewünschten Größe aufziehen. In dieses *ScrollPane*-Objekt wird dann

(2) das *TextArea*-Objekt eingebettet. Dazu wird die Komponente angeklickt und in den *Viewport-Bereich* des *ScrollPanes* gezogen. Der Textbereich erhält den Variablennamen *txtrAlleMitarbeiter.*

Damit das Objekt *txtrAllemitarbeiter* in allen Methoden der Klasse «Fenster_Mitarbeiter» verfügbar ist, muss der Deklarationsteil in den übergeordneten Teil der Klasse verlagert werden.

```
public class Fenster_Mitarbeiter extends JFrame
{
    // Eigene Deklarationen
        private JTextArea txtrAllemitarbeiter;
:
:

txtrAllemitarbeiter = new JTextArea();
scrollPane.setViewportView(txtrAllemitarbeiter);
txtrAllemitarbeiter.setText("alleMitarbeiter");
```

2) Ereignissteuerung für die Schaltfläche [alleMitarbeiter]



(1) Für die Schaltfläche wird ein „Lauscher“, der das Eintreten des Standardereignisses überwacht, eingerichtet.

(2) Tritt das Standardereignis ein, wird die „ereignisgesteuerte Methode“ ausgeführt.

(3) Im Methodenkörper der „ereignisgesteuerten Methode“ befinden sich die Anweisungen, die als Reaktion auf das eingetretene Ereignis auszuführen sind.

3) Quellcode der Methode `private void alleMitarbeiter()`

```
private void alleMitarbeiter()
{
mitarbeiterliste = aktuellerZugriff.mitarbeiterliste_db();
String tab = "\t"; String nZ = "\n";
txtrAllemitarbeiter.append("MitNr Name, Vorname" +tab+ "Gehalt"+tab+"Umsatz"+nZ);
txtrAllemitarbeiter.append("-------------------------------------------------"+nZ);

for (int i = 0; i < mitarbeiterliste.size(); i++)
      {
      txtrAllemitarbeiter.append(mitarbeiterliste.get(i).anzeigenMitarbeiterdaten()+nZ);
      }
}
}
```

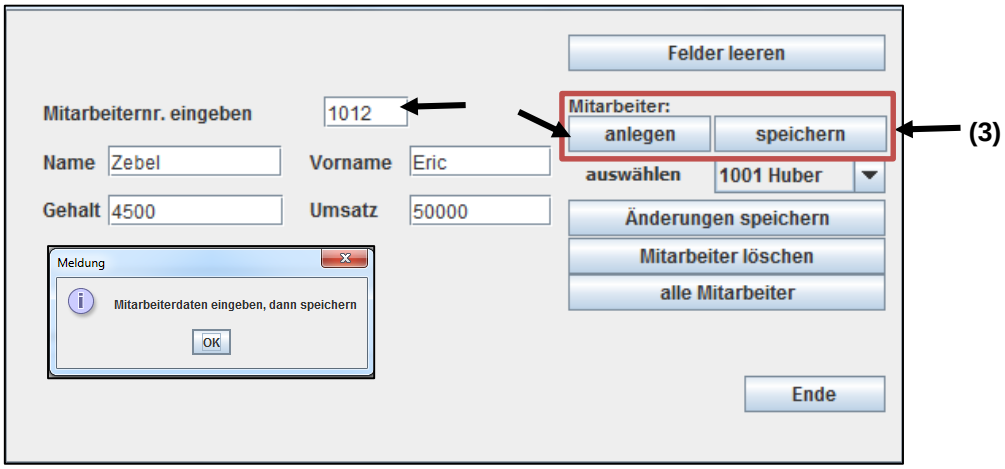
2.3.2 Die nächste Mitarbeiternummer ermitteln

Problemstellung:
Der neu eingestellte Mitarbeiter Eric Zebel mit einem Monatsgehalt von 4500,00 Euro und einem Umsatz von 50000,00 Euro soll in die Tabelle *mitarbeiter* aufgenommen werden.

Beim Erfassen des neuen Mitarbeiters soll die neue Mitarbeiternummer (Primärschlüsselattribut in der Tabelle *mitarbeiter*) vom Programm ermittelt werden. Dazu ist die GUI für die Mitarbeiterverwaltung anzupassen (siehe nachfolgende Abbildung.

Die Mitarbeiterverwaltung erhält für die Aufnahme neuer Mitarbeiter zwei Schaltflächen:

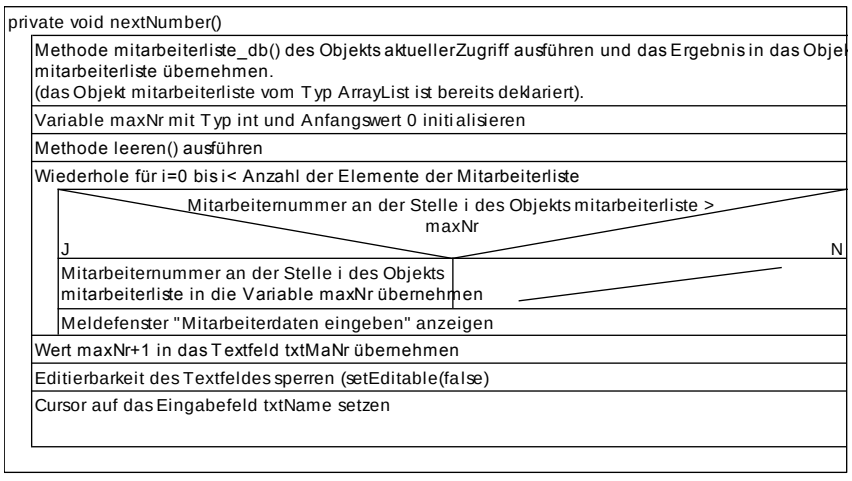
(1) Die Schaltfläche **`[anlegen]`**. Wird sie gedrückt, wird die neue Mitarbeiternummer ermittelt und im Textfeld für die Mitarbeiternummer eingetragen. Das Textfeld für die Mitarbeiternummer darf nicht mehr verändert werden können. Dies wird mit der Methode **`setEditable(false)`** für das Textfeld *txtMaxnr* erreicht.
(2) Gleichzeitig erscheint ein Meldefenster, mit dem Hinweis, dass jetzt die Daten erfasst werden können.
(3) Der Erfassungsvorgang wird mit der Schaltfläche **`[speichern]`** abgeschlossen.



Arbeitsaufträge

1) Verändern Sie die GUI. Die Schaltflächen **`[anlegen]`** und **`[speichern]`** (siehe obenstehende Abbildung) sind anzulegen.

2) Erhält die Schaltfläche **`[anlegen]`** das Standardereignis (*ActionPerformed*), wird die Methode **`private void nextNumber()`**, mit der die nächste Mitarbeiternummer ermittelt wird, ausgeführt. Codieren Sie dazu das dafür entwickelte Struktogramm:



3) Wenn die Schaltfläche **`[speichern]`** gedrückt wird, wird die Methode **`private void anlegenMitarbeiter()`** ausgeführt. Die eingegebenen Mitarbeiterdaten werden dann in der Datenbank gespeichert.

Lösungsvorschläge

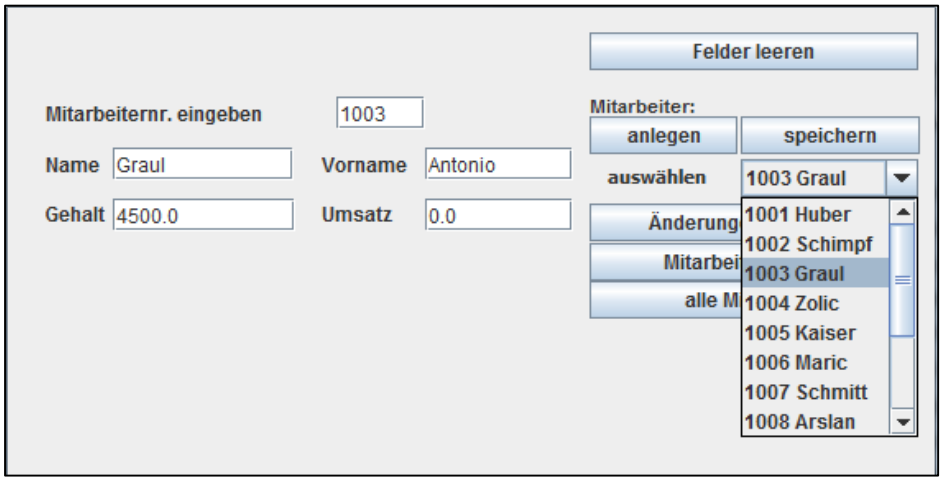
1) Javacode der Methode `private void nextNumber()`

```
private void nextNumber()
{
 mitarbeiterliste = aktuellerZugriff.mitarbeiterliste_db();
int maxNr=0;
leeren();
for (int i = 0; i < mitarbeiterliste.size(); i++)
{
      if (mitarbeiterliste.get(i).getMitarbeiternummer()> maxNr)
      {
            maxNr = mitarbeiterliste.get(i).getMitarbeiternummer();
      }
}
txtMaNr.setText(Integer.toString(maxNr+1)); txtMaNr.setEditable(false);
JOptionPane.showMessageDialog(null, "Mitarbeiterdaten eingeben, dann speichern");
txtName.requestFocus();
}
```

2.3.3 Mitarbeiter aus ComboBox auswählen

Problemstellung

Die Auswahl des zu ändernden/löschenden Mitarbeiters soll mithilfe eines Kombinationsfeldes, in dem alle Mitarbeiter aufgelistet sind, erfolgen (siehe nachfolgende Abbildung).

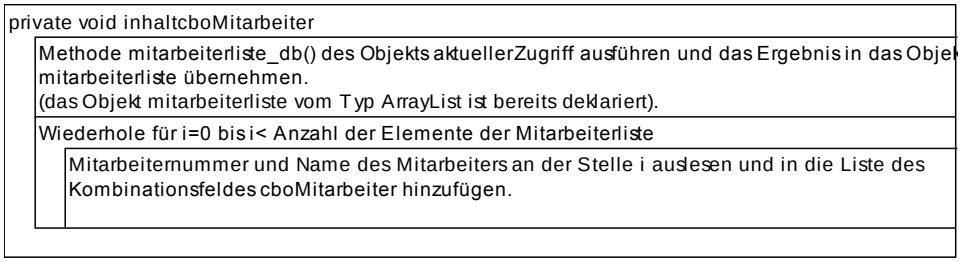


Wenn der Mitarbeiter in der Liste ausgewählt wurde (Mausklick), sollen die Attribute in den Textfeldern angezeigt werden.

Arbeitsaufträge

1) Ersetzen Sie in der GUI-Klasse «Fenster_Mitarbeiter» die Schaltfläche **`[Mitarbeiter suchen/anzeigen]`** durch das Kombinationsfeld *cboMitarbeiter* der Klasse «JComboBox». Achten Sie darauf, dass das Objekt *cboMitarbeiter* in allen Methoden der Klasse «Fenster_Mitarbeiter» verfügbar ist.

2) Im Konstruktor der GUI-Klasse «Fenster_Mitarbeiter» soll mit dem Aufruf der Methode **`private void inhaltcboMitarbeiter()`** das Kombinationsfeld *cboMitarbeiter* mit den Daten (Mitarbeiternummer, Name) aus der Tabelle *mitarbeiter* der Datenbank *mitarbeiterDB* gefüllt werden. Für die Methode wurde das folgende Struktogramm entworfen



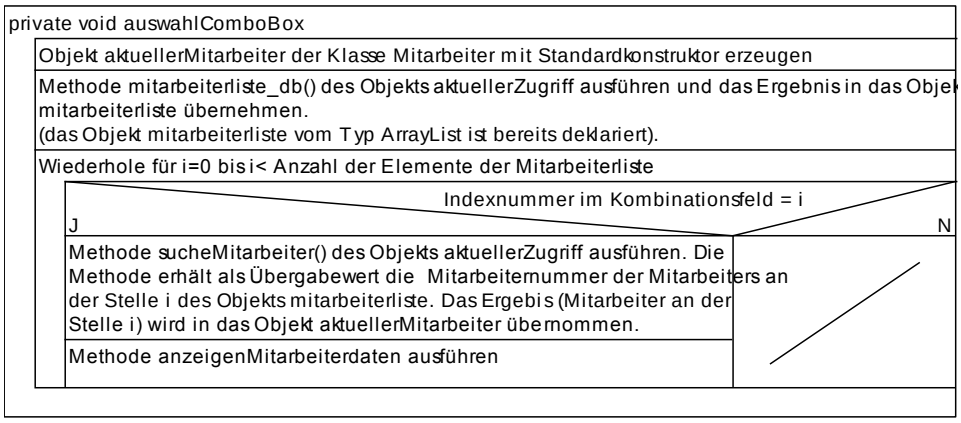
Hinweis: Mit der Methode **`addItem()`** können Elemente im Kombinationsfeld hinzugefügt werden.

Nehmen Sie den Methodenaufruf für die Methode **`private void inhaltcboMitarbeiter`** in den Konstruktor des Hauptfensters auf und codieren Sie anschließend die Methode wie im oben stehenden Struktogramm beschrieben.

3) Richten Sie für das Kombinationsfeld *cboMitarbeiter* eine Ereignissteuerung für das Standardereignis **`ActionPerformed`** ein. Wenn das Standardereignis eintritt, soll die Methode **`private void auswahlComboBox`** (siehe 4.) aufgerufen werden.

4) Mit der Methode **`private void auswahlComboBox()`** wird aus der Liste aller Mitarbeiter der gerade aktivierte Mitarbeiter anhand der Mitarbeiternummer ausgewählt. Die Mitarbeiterdaten werden dann in den Textfeldern der GUI angezeigt und stehen für weitere Tätigkeiten (ändern/löschen) zur Verfügung.

Codieren Sie dazu das nachfolgende Struktogramm:

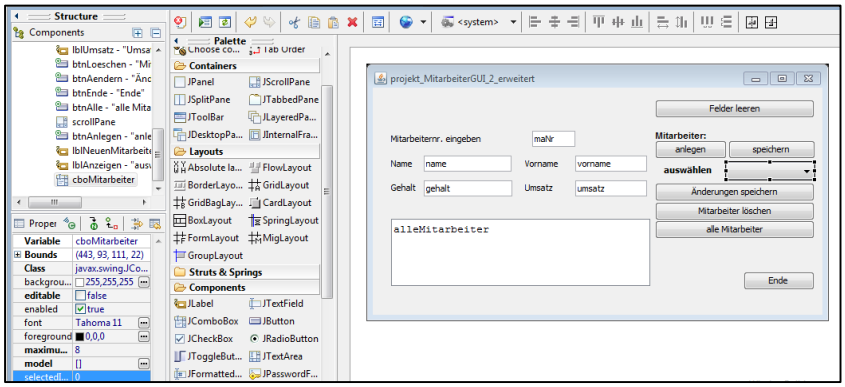


Hinweis: der Index des Eintrags aus dem Objekt cboMitarbeiter wird mit der Methode **`cboMitarbeiter.getSelectedIndex()`** ausgelesen.

5) In welchen Methoden zur Bearbeitung der angezeigten Mitarbeiterdaten muss der Inhalt im Kombinationsfeld *cboMitarbeiter* durch Ausführen der Methode **`inhaltcboMitarbeiter()`** aktualisiert werden?

Lösungshinweise

1) Kombinationsfeld *cboMitarbeiter* in die GUI aufnehmen



Deklaration im Kopf der Klasse «Fenster_Mitarbeiter»

```
public class Fenster_Mitarbeiter extends JFrame
{
	:
	// Eigene Deklarationen
	private JTextArea txtrAllemitarbeiter;
	private JComboBox<String> cboMitarbeiter;
	private ArrayList<Mitarbeiter> mitarbeiterliste;
	:
```

Quellcode des angelegten Kombinationsfeldes

```
	cboMitarbeiter = new JComboBox<String>();
	cboMitarbeiter.addActionListener(new ActionListener() {
	// Methode inhaltcboMitarbeiter
	inhaltcboMitarbeiter();
	cboMitarbeiter.setBounds(446, 92, 108, 23);
	contentPane.add(cboMitarbeiter);
```

2) Methode `private void inhaltcboMitarbeiter()`

```
private void inhaltcboMitarbeiter()
{
	mitarbeiterliste = aktuellerZugriff.mitarbeiterliste_db();
	cboMitarbeiter.removeAllItems();
	for (int i = 0; i < mitarbeiterliste.size(); i++)
	{
	cboMitarbeiter.addItem(mitarbeiterliste.get(i).getMitarbeiternummer()+"
"+mitarbeiterliste.get(i).getName());
	}
	cboMitarbeiter.setSelectedIndex(0);
}
```

3) Ereignissteuerung für das Kombinationsfeld cboMitarbeiter

```
cboMitarbeiter.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
	auswahlCobo();
	}
});
```

4) Methode private void auswahlCobo() für die Auswahl eines Elements der ComboBox

```
private void auswahlCobo()
{
	aktuellerMitarbeiter = new Mitarbeiter();
	mitarbeiterliste = aktuellerZugriff.mitarbeiterliste_db();
	for (int i = 0; i < mitarbeiterliste.size(); i++)
	{
		if (cboMitarbeiter.getSelectedIndex() == i)
		{
		aktuellerMitarbeiter =
aktuellerZugriff.sucheMitarbeiter(Integer.toString(mitarbeiterliste.get(i).getMitarbeiternummer()));
		anzeigenMitarbeiterdaten();
		}
	}
}
```

5) Aktualisierung des Inhaltes des Kombinationsfeldes

Der Inhalt des Kombinationsfeldes muss nach dem Einfügen, Ändern und Löschen eines Datensatzes erfolgen.

Aktualisierung am Beispiel des Löschen eines Mitarbeiters:

```
private void loeschenMitarbeiter()
{
	int mJN = JOptionPane.showConfirmDialog(null, "Wirklich löschen?"); //(3)
	System.out.println(mJN);
		if (mJN==0) {
		boolean ok= aktuellerZugriff.loescheMitarbeiter();
		// Inhalt der Combo-Box aktualisieren
		inhaltcboMitarbeiter();
		if (ok) {
			JOptionPane.showMessageDialog(
					null,
					"Mitarbeiter gelöscht!!");
			leeren();
		} else {
			JOptionPane.showMessageDialog(
					null,
					"Mitarbeiter nicht gelöscht!");
		}
	}
}
```