Objektorientierte Systementwicklung:

Grafische Benutzeroberflächen mit Java

Grafische Benutzeroberflächen mit Java

1 Einführung

1.1 GUI-Programmierung mit Java

Die bisher realisierten objektorientierten Applikationen wurden mithilfe von Eclipse, einer integrierte Entwicklungsumgebung für die Programmiersprache Java, entwickelt. Eine unmittelbare Kommunikation des Anwenders mit der entwickelten Software fand nicht statt. Eingaben mussten mithilfe von Anweisungen in einer Klasse mit main-Methode (Startklasse) formuliert werden. Ausgaben erfolgten im Konsolenfenster der Entwicklungssoftware, die ebenfalls durch Anweisungen in der Startklasse angestoßen wurden.

Im Gegensatz zu Konsolenanwendungen erleichtern grafische Benutzeroberflächen (Graphical User Interface – GUI) die Interaktion zwischen Anwender und Software. Sie bestehen aus Fenstern, in denen sogenannte Steuerelemente wie bspw. Schaltflächen (Button), Schriftfelder (Label) oder Eingabefelder (Textfeld) platziert werden. Diese grafischen Symbole werden mithilfe der Eingabegräte Tastatur oder Maus bedient, können aber auch durch einen Touchscreen zum Einsatz kommen.
Die Programmiersprache Java als plattformunabhängige Softwareentwicklungsumgebung stellt mit den Packages *Abstract Window Toolkid* (AWT) und Swing eine Sammlung von Bibliotheken zur Programmierung von grafischen Benutzeroberflächen bereit. Diese Bibliotheken sind Bestandteil der Java-Runtime (jre) und stellen die Elemente zur Verfügung, die am häufigsten zur GUI-Programmierung benötigte werden. Hierzu zählen der Einsatz von Steuerelementen (Textfelder, Schalflächen, Optionsfelder etc.) sowie die Behandlung von Ereignissen (Mausbewegung).

Die Entwicklung von grafischen Benutzeroberflächen in Java setzt voraus, dass die Gestaltung einer Oberfläche mit Java-Code erfolgen muss. Jede GUI-Komponente muss also mit dem *new*-Operator erzeugt und mithilfe eines Layouts angeordnet werden. Schon eine kleine Oberfläche mit wenigen GUI-Elementen verlangt somit einen umfangreichen Programmieraufwand. Mit der Entwicklung von speziellen GUI-Buildern werden Werkzeuge zur Verfügung gestellt, die diese Arbeit erheblich vereinfachen.

Die integrierte Entwicklungsumgebung Eclipse stellt keinen speziellen GUI-Builder zur Verfügung. Er kann jedoch aus einer Vielzahl von verfügbaren Plugins ausgewählt und in Eclipse eingebunden werden. Die Bearbeitung der im weiteren Verlauf dieser Unterlagen beschriebenen Projekte erfolgt unter dem Einsatz des Plug-Ins "WindowBuilder".

1.3 WindowBuilder

Das GUI-Entwickler-Werkzeug "WindowBuilder" wurde von dem amerikanischen Softwarehaus "Instantiations" für kommerzielle Zwecke entwickelt. Seit der Übernahme durch Google im Jahre 2010 wird das Tool kostenlos zur Verfügung gestellt. Mit dem WindowBuilder können Java-GUI-Anwendungen auf Basis der Grafikbiblithek Swing entwickelt werden. Bei Swing handelt es sich um eine von Sun Microsystems für die Programmiersprache Java entwickelte Programmierschnittstelle zum Programmieren grafischer Benutzeroberflächen.

Swing basiert auf dem Toolkit AWT (Abstract Window Toolkit) und erweitert dieses um zusätzliche Komponenten. So können mit Swing beispielsweise GUI-Elemente mit Tastenkombinationen oder mittels Drag and Drop gesteuert werden. Darüber hinaus stehen neue Layout-Manager zur Anordnung der GUI-Elemente sowie zusätzliche Komponenten zur Darstellung von Passworteingaben, Tabellen und Baumstrukturen zur Verfügung.

Zu beachten ist aber auch, dass AWT Klassen und Schnittstellen definiert, für die es in Swing keine entsprechende Alternative gibt. Für die Entwicklung von grafischen Benutzeroberflächen wird somit der Rückgriff auf beiden Bibliotheken erforderlich sein.

Im Gegensatz zu AWT, das auf Systemkomponenten basiert, werden Swing-Komponenten unabhängig vom Betriebssystem "gezeichnet". Das bedeutet, dass grafische Benutzeroberflächen, die mit dem AWT erstellt wurden, mit einem für das jeweilige Betriebssystem typischen Aussehen angezeigt werden. Demgegenüber haben mit Swing entwickelte grafische Oberflächen, unabhängig vom zugrunde liegenden Betriebssystem immer das gleiche Aussehen.

Die zur Entwicklung einer grafischen Benutzeroberfläche benötigten Klassen befinden sich in den Bibliotheken java.awt.* und javax.swing.*.

2 Entwicklung einer GUI-Anwendung

2.1 Problemstellung

Die Firma GeLa GmbH verwaltet ihre Mitarbeiterdaten mithilfe einer objektorientierten Software. Die bisher entwickelten Anwendungen beschränkten sich bei der Ausgabe von Daten ausschließlich auf die Darstellung auf der Konsole. Um die Programme modern und grafisch ansprechend zu gestalten sowie eine intuitive Bedienung der Anwendungen zu ermögliche, ist für die Kommunikation zwischen Nutzer und Software in Zukunft der Einsatz grafischer Benutzeroberflächen geplant.

In einer ersten Entwicklungsphase sollen die Mitarbeiterstammdaten erfasst und wieder am Bildschirm angezeigt werden können.
(Auf die permanente Speicherung der Mitarbeiterdaten wird in dieser Projektphase verzichtet.)

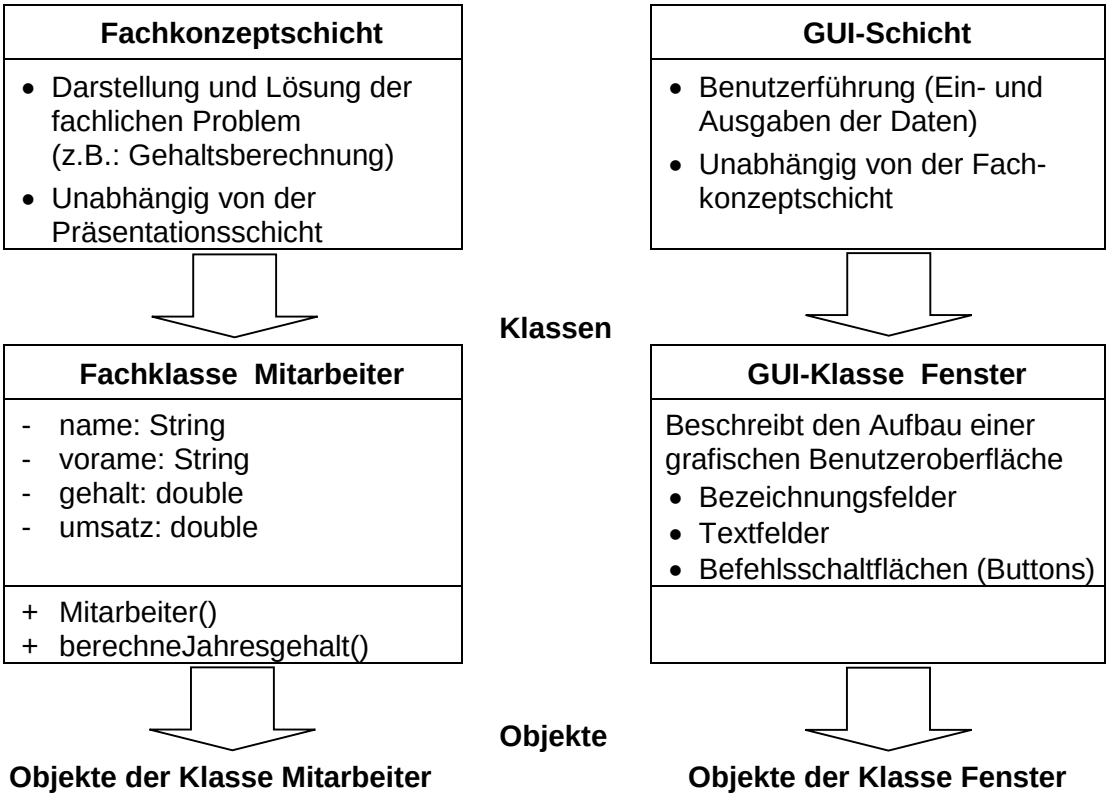
Hierzu wurde bereits die Fachklasse »Mitarbeiter« mit den Attributen *name*, *vorname*, *gehalt* und *umsatz* sowie den zugehörigen Set- und Get-Methoden erstellt ("projekt_MitarbeiterGUI_1").

Die gewünschte grafische Benutzeroberfläche soll folgenden Aufbau haben:

2.2 Schichtenmodell

Ein zentrales Prinzip bei der Entwicklung von Softwaresystemen ist die Trennung von fachlicher Problemlösung, grafischer Darstellung und Datenhaltung. Mit der Realisierung des Schichtenmodells, dem grundsätzliches Architekturprinzip der objektorientierten Systementwicklung, wird dieses Prinzip eingehalten. Es verfolgt das Ziel, die Komplexität von Softwaresystemen besser zu strukturieren und zu minimieren.

Das im Folgenden verwendete Zwei-Schichten-Modell unterscheidet in eine Präsentationsschicht (GUI-Schicht) und eine Logikschicht (Fachkonzeptschicht). Damit besteht die Möglichkeit, grafische Benutzeroberflächen auszutauschen, ohne das Fachkonzept ändern zu müssen.



2.3 Erstellen einer Fensterklasse

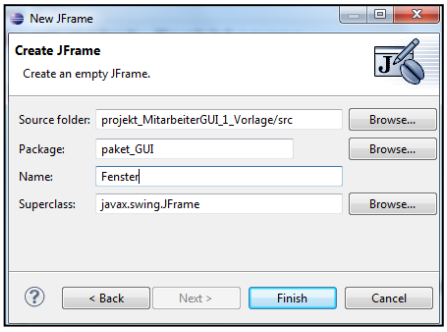
2.3.1 WindowBuilder als Entwicklungswerkzeug

Das Eclipse-Plugin WindowBuilder stellt Werkzeuge für GUI-Entwickler zur Verfügung, mit deren Hilfe Java-basierte grafische Benutzeroberflächen erstellt werden können.

Innerhalb des Pakets **paket_GUI** wird mit dem Eclipse-Menübefehl

File ► New ► Other… ► WindowBuilder ► Swing Designer ► JFrame

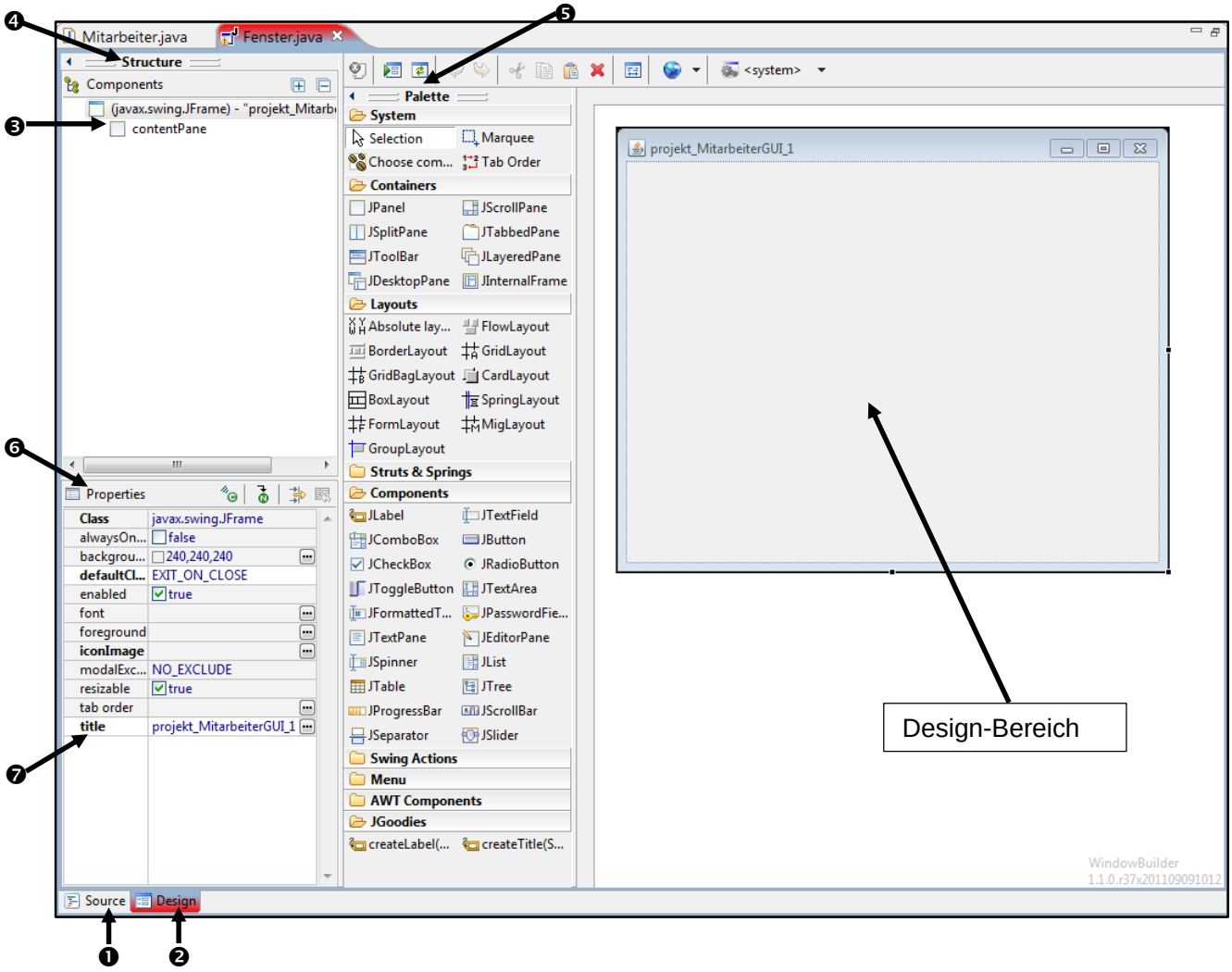
die WindowBuilder-Oberfläche zum Erstellen einer Fensterklasse gestartet. Alternativ kann das Symbol '*create new visuell classes*' verwendet werden.



Zunächst muss ein Klassenname vergeben werden (z.B. »Fenster«).

Wie die Abb. zeigt, erbt die Klasse »Fenster« Methoden der Superklasse »JFrame« (importiert aus der Klassenbibliothek javax.swing).

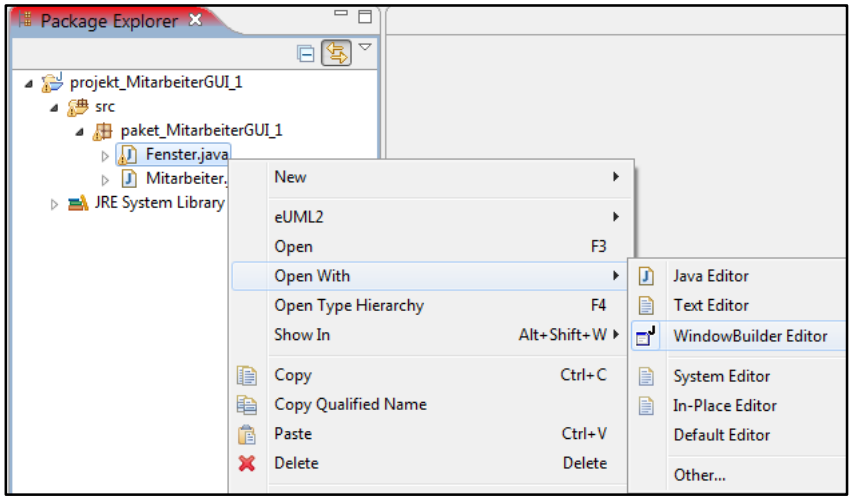
Nach der Bestätigung der Eingabe öffnet Eclipse ein Fenster mit den beiden Sichten ‘Source‘ (❶) und ‘Design‘ (❷). In der Source-Sicht wird der Quellcode, in der Design-Sicht die Entwurfsansicht der erzeugten Fensterklasse angezeigt.



Eine bereits erstellte Fensterklasse kann mit einem Doppelklick im Package Explorer in der Quellcode-Ansicht geöffnet werden.

Soll auch die Entwurfsansicht der GUI sichtbar sein, muss die Klasse mit dem Editor des WindowBuilders geöffnet werden:

Kontextmenü zur Klasse »Fenster«
- ► Open With
  - ► WindowBuilder Editor



2.3.2 Bildschirmaufbau des WindowBuilders

Alle Objekte, die ein Programm am Bildschirm anzeigen soll, müssen in einem sogenannten Top-Level-Fenster liegen, das vom Betriebssystem verwaltet wird. In Java heißt dieses Fenster **Frame**. Die Bibliothek *Swing* bietet mit der Klasse »JFrame« Methoden zur Entwicklung eines Top-Level-Fensters.

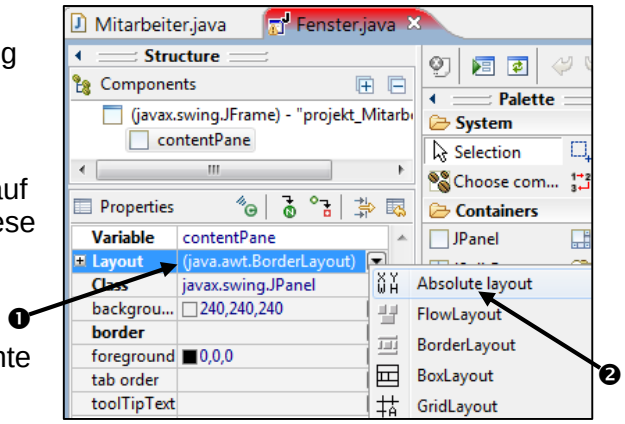
Die Fensterklasse verfügt über einen **Panel**-Container (contentPane ❸), der die Aufnahme und Anordnung von grafischen Komponenten (Beschriftungen, Textfelder, Schaltflächen etc.) am Bildschirm ermöglicht. Die Bibliothek *Swing* stellt hierzu Methoden in der Klasse »JPanel« zur Verfügung. Auf der WindowBuilder-Oberfläche wird dieser Container und seine Komponenten im Unterfenster "**Structure**" (❹) angezeigt.

Das Unterfenster "**Palette**" (❺) listet die verschiedenen Elemente auf, mit denen die grafische Benutzeroberfläche gestaltet werden kann. Das jeweilige Element wird per Drag & Drop in den "**Design-Bereich**" gezogen.

Zu jedem Element können im Unterfenster "**Properties**" (❻)die Eigenschaften angepasst werden.

Zunächst soll mithilfe der Eigenschaft '*title*' (❼) die Beschriftung der Titelzeile des Fensters mit dem Text "projekt_MitarbeiterGUI_1" festgelegt werden.

Der WindowBuilder öffnet das contentPane in der Design-Sicht standardmäßig mit der Layouteinstellung "BorderLayout" (❶). Hierbei handelt es sich um eine vordefinierte Aufteilung des Bildschirms in fünf Bereiche, in denen die GUI-Komponenten platziert werden können. Um die einzelnen Elemente individuell auf der Designoberfläche platzieren zu können, muss diese Eigenschaft des contentPanes auf die Einstellung "Absolute layout" (❷) geändert werden.



Anschließend können die gewünschten Grafikelemente in den Design-Bereich gezogen werden. Über sie erfolgt die Interaktion mit dem Anwender.

Quellcode der generierten GUI-Klasse »Fenster«

```
   package paket_GUI;
   //Import der benötigten swing- und awt-Klassen
   import javax.swing.JFrame;
   import java.awt.BorderLayout;
   import java.awt.EventQueue;
   import javax.swing.JPanel;
   import javax.swing.border.EmptyBorder;
①  public class Fenster extends Jframe  {
②  private JPanel contentPane;

③     public static void main(String[] args)  {
         EventQueue.invokeLater(new Runnable()  {
            public void run()  {
               try  {
④                Fenster frame = new Fenster();
⑤                frame.setVisible(true);
               }
               catch(Exception e)  {
                 e.printStackTrace();
               }
            }
         });
      }

⑥     public Fenster()  {
⑦        setTitle("projekt_MitarbeiterGUI_1");
⑧        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
⑨        setBounds(100, 100, 540, 400);
⑩        contentPane = new JPanel();
         contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
         setContentPane(contentPane);
         contentPane.setLayout(null);
      }
   }
```

Erläuterungen:

① Deklaration der Sub-Klasse »Fenster«. Sie erbt Methoder der Super-Klasse »JFrame«.

② Deklaration eines Panel-Containers (contentPane) vom Typ der Klasse »JPanel«.

③ Methode zur Ausführung der Klasse (`main`-Methode).
Eine eigene Klasse mit main-Methode (z.B. Startklasse) zum Starten der Anwendung ist nicht mehr erforderlich.

④ Ein Objekt (*frame*) vom Typ der Klasse »Fenster« wird instanziiert.

⑤ Das Objekt *frame* wird sichtbar gemacht.

⑥ Konstruktor der Klasse »Fenster«.

⑦ Text für die Titelzeile des Fensters wird festgelegt.

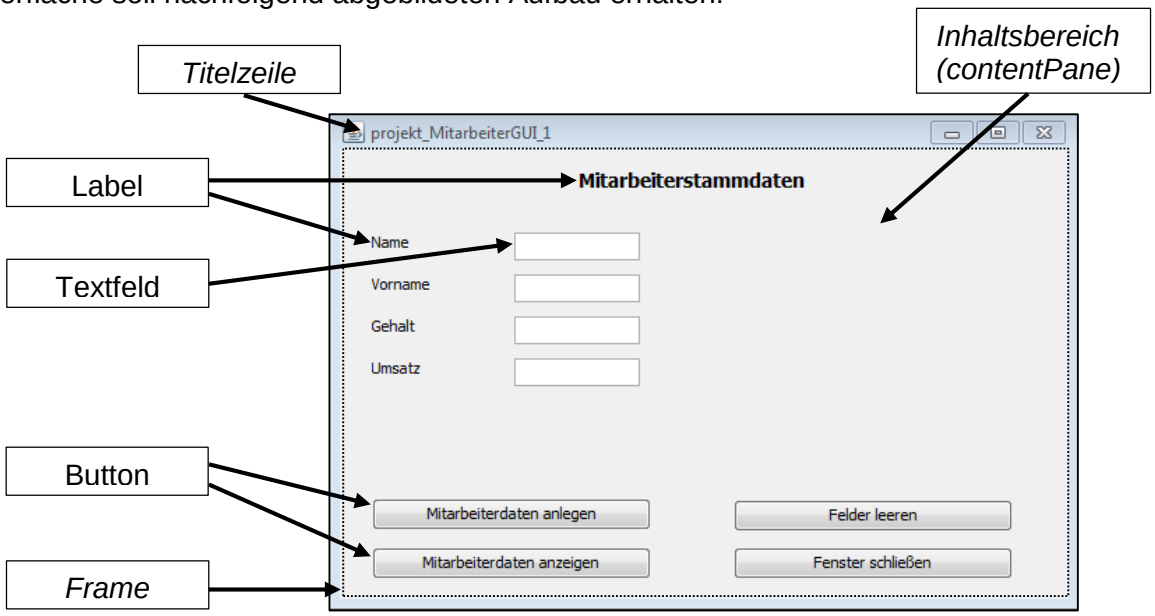
⑧ Ermöglicht das Beenden der Anwendung mithilfe des Schließen Buttons ()

⑨ Koordinaten für die Platzierung der GUI am Bildschirm werden festgelegt.

⑩ - Ein Objekts (contentPane) vom Typ der Klasse »JPanel« wird instanziiert.
- Für diesen Container wird ein unsichtbarer Rand festgelegt, der den Abstand zu den GUI-Inhalten definiert.
- Das Objekt *contentPane* wird als aktueller Panel-Container (ContentPane) festgelegt.
- Der Layout-Manager für diesen Container wird auf null (absolute) gesetzt, d.h., die GUI-Komponenten müssen absolut positioniert werden.

2.3.3 Grafische Komponenten der Benutzeroberfläche

Die für die Mitarbeiterdatenverwaltung der Firma GeLa GmbH zu entwickelnde grafische Benutzeroberfläche soll nachfolgend abgebildeten Aufbau erhalten:

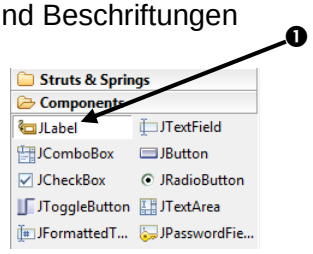


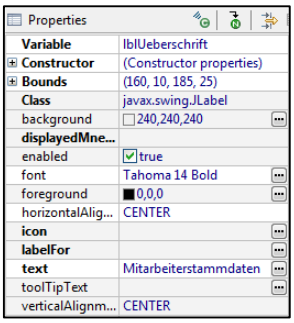
(Abb.: Fenster GUI_1)

Label

Ein Label ist eine GUI-Komponente, die eine Zeichenkette oder ein Icon repräsentiert. Sie/es ist vom Benutzer nicht editierbar und wird für die Darstellung von Überschriften und Beschriftungen verwendet. Labels sind Objekte der Swing-Klasse »JLabel«.

Um ein Label auf der contentPane zu platzieren, muss in der *Design-Sicht* des Fensters das entsprechende Symbol (❶) in der Rubrik '*Components*' aktiviert werden. Anschließend kann das Label in gewünschter Größe auf die Inhaltsfläche gezogen werden.





Labels erhalten mehrere Eigenschaften, die im Unterfenster '*Properties*' festgelegt werden können:

Einen Namen, unter dem sie "ansprechbar" sind (Eigenschaft: *Variable*), Angaben zur Positionierung auf der Inhaltsfläche (Eigenschaft: *Bounds*), Zuweisung einer Schriftart und einer Schriftgröße und eines Schriftgewichts (Eigenschaft: *font*), eine Schriftfarbe (Eigenschaft: *foreground*), eine horizontale Ausrichtung (Eigenschaft: *horizontalAlignment*), eine Beschriftung (Eigenschaft: *text*) sowie eine vertikale Ausrichtung (Eigenschaft: *vertikalAlignment*).

Mit dem Einfügen des Labels *lblUeberschrift* auf dem contantPane wird folgender Quellcode generiert (Source-Sicht):

Importbereich:

```
import javax.swing.JLabel;
import java.awt.Font;
```

Deklarationsbereich:

❶ `public final JLabel lblUeberschrift = new JLabel("Mitarbeiterstamm-daten");`

Konstruktor:

❷ `contentPane.add(lblUeberschrift);`
❸ `lblUeberschrift.setHorizontalAlignment(SwingConstants.CENTER);`
❹ `lblUeberschrift.setFont(new Font("Tahoma", Font.BOLD, 14));`
❺ `lblUeberschrift.setBounds(160, 10, 185, 25);`

Erläuterungen:

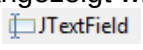
❶ Instanziierung des Objekts *lblUeberschrift* vom Typ der Klasse »JLabel« mit Parameterübergabe (Beschriftung des Objekts).
❷ Das Objekt *lblUeberschrift* wird dem Containerobjekt *contentPane* übergeben.
❸ Aufruf der Methode *setHorizontalAlignment* des Label-Objekts *lblUeberschrift* mit Übergabe der ausgewählten Ausrichtungseigenschaft.
❹ Aufruf der Methode *setFont* des Label-Objekts *lblUeberschrift* mit Übergabe der Schriftart, des Schriftschnitts und der Schriftgröße
❺ Aufruf der Methode *setBounds* des Label -Objekts *lblUeberschrift* mit Übergabe der Positionsdaten (Abstand vom linken Fensterrand, Abstand vom oberen Fensterrand, Breite des Labels, Höhe des Labels).

Verwendete Labels der Klasse »Fenster« des Projekts "projekt_MitarbeiterGUI_1":

Überschrift	lblUeberschrift	"Mitarbeiterstammdaten"
Beschriftung für Ein-/Ausgabefelder	lblName	"Name"
	lblVorname	"Vorname"
	lblGehalt	"Gehalt"
	lblUmsatz	"Umsatz"

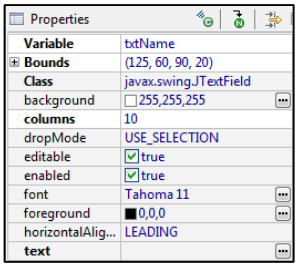
(Die weiteren Labels sind entsprechend der Abbildung "Fenster GUI_1" zu platzieren.)

Textfeld

Ein Textfeld ist eine GUI-Komponente zur Eingabe eines einzeiligen Textes. Textfelder sind Objekte der Swing-Klasse »JText«. Die Methode `getText()` ermöglicht den Zugriff auf den eingegebenen Text. Mit Hilfe der Methode `setText(String x)` kann dem Objekt ein Text übergeben werden, der im Textfeld angezeigt wird.
Sie werden mithilfe der Komponente 

ausgewählt und auf der Inhaltsfläche platziert.

Textfelder haben einen Namen, unter dem sie "ansprechbar" sind (Eigenschaft: *Variable*) und Angaben zur Positionierung auf der Inhaltsfläche (Eigenschaft: *Bounds*).
Die weiteren Eigenschaften können mit ihren Defaultwerten übernommen werden.



Mit dem Einfügen des Textfeldes *txtName* auf dem contantPane wird folgender Quellcode generiert (Source-Sicht):

Importbereich:

```
import javax.swing.JTextField;
```

Deklarationsbereich:

```
❶ public final JTextField txtName = new JTextField();
```

Konstruktor:

```
❷ contentPane.add(txtName);
❸ txtName.setBounds(125, 60, 90, 20);
❹ txtName.setColumns(10);
```

Erläuterungen:

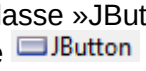
❶ Instanziierung des Objekts *txtName* vom Typ der Klasse »JTextField« .
❷ Das Objekt *txtName* wird dem Containerobjekt *contentPane* übergeben.
❸ Aufruf der Methode *setBounds* des Textfield-Objekts *txtName* mit Übergabe der Positionsdaten.
❹ Aufruf der Methode *setColumns* des Textfield-Objekts *txtName* mit Übergabe der Länge des Ein-/ Ausgabefelds.

Verwendete Textfelder (Ein-/ Ausgabefelder) der Klasse »Fenster« des Projekts "projekt_MitarbeiterGUI_1":

txtName
txtGehalt
txtUmsatz
txtProvSatz

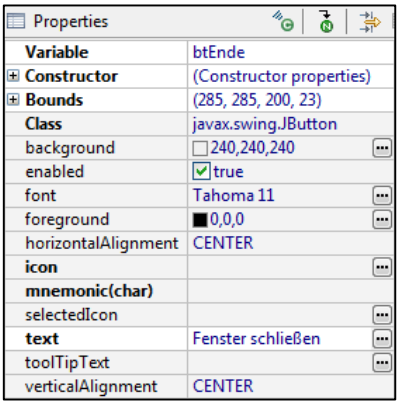
(Die weiteren Textfelder sind entsprechend der Abbildung "Fenster GUI_1" zu platzieren.)

Button (Schaltfläche)

Ein Button ist eine GUI-Komponente, die es einem Anwender ermöglicht, eine Aktion auszulösen. Buttons sind Instanzen der Swing-Klasse »JButton«.
Sie werden mithilfe der Komponente 

ausgewählt und auf der Inhaltsfläche platziert.

Buttons haben einen Namen, unter dem sie "ansprechbar" sind (Eigenschaft: *Variable*), Angaben zur Positionierung auf der Inhaltsfläche (Eigenschaft: *Bounds*) und eine Beschriftung (Eigenschaft: *text*).
Die weiteren Eigenschaften können mit ihren Default-Werten übernommen werden.



Die Funktionalität der Schaltflächen muss in der Source-Sicht des Fensters codiert werden.

Mit dem Einfügen des Buttons *btAnlegen* auf dem contantPane wird folgender Quellcode generiert (Source-Sicht):

Importbereich:

```
import javax.swing.JButton;
```

Deklarationsbereich:

```
❶ public final JButton btEnde = new JButton("Fenster schließen");
```

Konstruktor:

```
❷ contentPane.add(btEnde);
❸ btEnde.setBounds(280, 285, 200, 23);
```

Erläuterungen:

❶ Instanziierung des Objekts *btEnde* vom Typ der Klasse »JButton«.
❷ Das Objekt *btEnde* wird dem Containerobjekt *contentPane* übergeben.
❸ Aufruf der Methode *setBounds* des Button-Objekts *btEnde* mit Übergabe der Positionsdaten.

Verwendete Buttons (Schaltflächen) der Klasse »Fenster« des Projekts "projekt_MitarbeiterGUI_1":

btAnlegen	"Mitarbeiterdaten anlegen"
btZeigen	"Mitarbeiterdaten anzeigen"
btLeeren	"Fenster leeren"
btEnde	"Fenster schließen"

(Die weiteren Buttons sind entsprechend der Abbildung "Fenster GUI_1" zu platzieren.)

2.4 Funktionalität der Schaltflächen

Mithilfe der GUI-Komponenten soll der Anwender ein Programm steuern können. Die Programmsteuerung erfolgt durch Maus-Aktionen oder Tastatureingaben. Diese Aktionen veranlasst das Betriebssystem, eine Nachricht zu erzeugen und an die betroffene Anwendung zu senden.

Die Auslöser von Nachrichten sind sogenannte Ereignisquellen (Event Sources). Auf die vom Betriebssystem gesendeten Nachrichten reagiert die Anwendung mit Ereignisempfängern (EventListeners).

Die AWT-Bibliothek stellt mit dem Event-Package zahlreiche Unterklassen zur Verfügung, die spezielle Ereignisse beschreiben (»ActionEvent«, »MouseEvent« etc.). Sobald zu einer Schaltfläche ein Ereignis auftritt, wird ein entsprechendes AWT-Event-Objekt erzeugt (Event Handler). Es verfügt über Methoden, die beim Empfangen von Nachrichten reagieren (actionPerformed, mouseClicked).

Die Methode *actionPerformed* reagiert, wenn eine GUI-Komponente eine Aktion erfährt. Dies ist der Fall, wenn eine Schaltfläche gedrückt (Mausklick oder Leertaste), die Eingabe in ein Textfeld mit Enter abgeschlossen oder ein Element aus einem Listenfeld ausgewählt wird.

Die Methode *mouseClicked* reagiert ausschließlich auf einen Mausklick.

2.4.1 Schaltfläche "Fenster schließen"

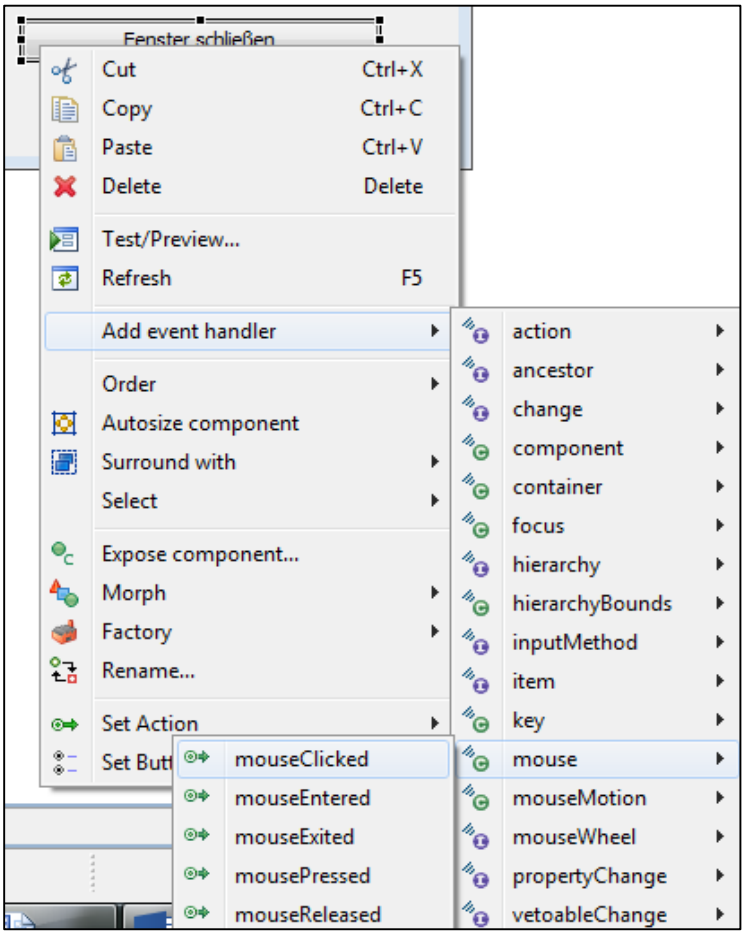
Mit einem Klick auf die Schaltfläche "Fenster schließen" ('btEnde') soll die GUI-Anwendung beendet werden. Hierzu muss ein entsprechender Event-Handler implementiert werden.

Im Kontextmenü zum GUI-Element btEnde ist die Methode *mouseClicked* auszuwählen

- ► Add event handler
  - ► mouse
    - ► mouseClicked

Alternativ kann auch die allgemeinere Methode *actionPerformed* gewählt werden

- ► Add event handler
  - ► action
    - ► actionPerformed



Der Button erhält ein Listener-Objekt vom Typ der Klasse »MouseAdapter«. Sobald das Objekt eine entsprechende Nachricht empfängt, wird die Methode *mouseClicked* aufgerufen.
Diese Methode muss die Anweisung zum Beenden der Applikation enthalten.

Im Quellcode der Klasse »Fenster« ergeben sich folgende Ergänzungen:

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
....
btEnde.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        System.exit(0);
    }
});
```

2.4.2 Schaltfläche "Felder leeren"

Mithilfe dieser Schaltfläche ('btLeeren') sollen vorhandene Einträge in den Textfeldern gelöscht und der Cursor im Textfeld txtName positioniert werden. Dazu muss die Methode *mouseClicked* die Methode *setText* der jeweiligen Objekte (txtName etc.) vom Typ der Klasse »JTextField« aufrufen und einen leeren Textstring übergeben.
Der Cursor wird mit der Methode *requestFocus* im Textfeld txtName platziert.

Die Schaltfläche 'btLeeren' erhält nach der oben beschriebenen Vorgehensweise die Ereignismethode *mouseclicked*. Die Funktionalität dieser Methode soll in eine eigenständige Methode *leeren* ausgelagert werden, die von der *mouseclicked*-Methode aufgerufen wird.

Quellcode Schaltfläche 'btLeeren':

```
btLeeren.addMouseListener(new MouseAdapter()  {
   public void mouseClicked(MouseEvent e) {
        leeren();
   }
});
btLeeren.setBounds(280, 240, 200, 23);
contentPane.add(btLeeren);
....
// selbst erstellte Methoden
// Textfelder leeren
public void leeren()
{
   txtName.setText("");
   txtVorname.setText("");
   txtGehalt.setText("");
   txtUmsatz.setText("");
   txtName.requestFocus();
}
```

2.4.3 Schaltfläche "Mitarbeiterdaten anlegen"

Mithilfe dieser Schaltfläche ('btAnlegen') sollen die in den Textfeldern eingetragenen Daten einem Objekt vom Typ »Mitarbeiter« übergeben und dort als Attributwerte zur Laufzeit der Applikation gespeichert werden.

Die Methode *mouseclicked* der Schaltfläche 'btAnlegen' hat folgende Funktionen auszuführen:

- Die Informationen Name, Vorname, usw. gehören zu einer konkreten Person, für die ein Objekt der Klasse »Mitarbeiter« instanziiert werden muss. (Konstruktor der Klasse »Mitarbeiter«).
- Auslesen der Inhalte der vier Textfelder (*getText*-Methode des jeweiligen Textfeld-Objekts).
- Umwandlung (parsen) der aus den Feldern txtGehalt und txtUmsatz ausgelesenen Werte (Texte) in ein Zahlenformat (Methode *parseDouble* der Klasse »Double«) (①).
- Übergabe der ausgelesenen Inhalte als Attributwerte an das Mitarbeiterobjekt (*set*-Methoden des Mitarbeiterobjekts).

Quellcode Schaltfläche 'btAnlegen':

<u>Deklarationsbereich:</u>

```
//Mitarbeiterobjekt instanziieren
private Mitarbeiter einMitarbeiter = new Mitarbeiter();
```

<u>Konstruktor:</u>

```
....
btAnlegen.addMouseListener(new MouseAdapter(){
   public void mouseClicked(MouseEvent e)
   {
      anlegen();
   }
});
btAnlegen.setBounds(23, 240, 200, 23);
contentPane.add(btAnlegen);
```

selbst erstellte Methode:

```
    //Mitarbeiterdaten zur Laufzeit speichern
    public void anlegen()
    {
       einMitarbeiter.setName(txtName.getText());
       einMitarbeiter.setVorname(txtVorname.getText());
①      einMitarbeiter.setGehalt(Double.parseDouble(txtGehalt.getText()));
       einMitarbeiter.setUmsatz(Double.parseDouble(txtUmsatz.getText()));
    }
```

2.4.4 Schaltfläche "Mitarbeiterdaten anzeigen"

Diese Schaltfläche ('btZeigen') soll es ermöglichen, die Attributwerte eines aktuellen Mitarbeiterobjekts in den Textfeldern der GUI auszugeben.

Die Methode *mouseclicked* der Schaltfläche 'btZeigen' hat folgende Funktionen auszuführen:

- Auslesen der Attributwerte des Mitarbeiterobjekts (*get*-Methoden des Mitarbeiterobjekts).
- Umwandlung der ausgelesenen Zahlenwerte (gehalt, umsatz) in Texte (Methode *toString* der Klasse »Double«). (❷)
- Übergabe der ausgelesenen Werte in die Textfelder der GUI (*setText*-Methode des jeweiligen Textfeld-Objekts).

Quellcode Schaltfläche 'btZeigen':

Konstruktor:

```
    btZeigen.addMouseListener(new MouseAdapter() {
       public void mouseClicked(MouseEvent e)
       {
          zeigen();
       }
    });
    btZeigen.setBounds(23, 285, 200, 23);
    contentPane.add(btZeigen);
```

selbst erstellte Methode:

```
    //Mitarbeiterdaten zeigen
    public void zeigen()
    {
       txtName.setText(einMitarbeiter.getName());
       txtVorname.setText(einMitarbeiter.getVorname());
❷      txtGehalt.setText(Double.toString(einMitarbeiter.getGehalt()));
       txtUmsatz.setText(Double.toString(einMitarbeiter.getUmsatz()));
    }
```

2.5 Erweiterung der Problemstellung

2.5.1 Eingabefehler prüfen

Die Fensterklasse des Projekts "projekt_MitarbeiterGUI_1" soll dahingehend erweitert werden, dass folgende Eingabefehler geprüft und dem Anwender entsprechende Hinweise gegeben werden:

1) Vor der Betätigung der Schaltfläche 'btAnlegen' wurden keine Eingaben in den Textfeldern gemacht.
2) In den Textfeldern txtGehalt bzw. txtUmsatz wurde Text anstatt Zahlen eingegeben.

Die Problemlösung der beschriebenen Situation soll im Projekt "projekt_MitarbeiterGUI_1_erweitert" realisiert werden. Hierzu kann im Package Explorer mithilfe des Kontextmenüs zum Projekt "projekt_MitarbeiterGUI_1" das Projekt kopiert (Copy – Paste) und entsprechend umbenannt werden.

Zu 1)

❶ Vor dem Aufruf der Methoden *setName(String pName)* und *setVorname(String pVorname)* des Mitarbeiterobjekts muss geprüft werden, ob die Methode *getText()* des jeweiligen Textfeldes einen leeren Textstring zurückliefert.

❷ Sofern dies der Fall ist, soll eine ausgelagerte Methode *fehlermeldung()* aufgerufen und die Meldung "Text eingeben" mitgeliefert werden. Diese Methode kann immer dann genutzt werden, wenn dem Anwender ein Hinweis in einer Message-Box angezeigt werden soll.

❸ Wenn die Methode *getText()* des jeweiligen Textfeldes einen Wert liefert, wird dieser der Methode *setName(String pName)* bzw. *setVorname(String pVorname)* des Mitarbeiterobjekts übergeben.

Zu 2)

Erwartet eine Methode beim Aufruf einen numerischen Übergabeparameter und erhält stattdessen einen Text, tritt ein sogenannter Ausnahmefehler (hier: NumberFormatException) auf. Die Behandlung von Ausnahmefehlern erfolgt mithilfe der try-catch-Anweisung. Der try-Block enthält dabei die Anweisung(en), bei deren Ausführung Fehler auftreten können. In diesem Fall wird der Programmablauf unterbrochen und nach der catch-Klausel fortgesetzt.

❹ Das bedeutet, dass sowohl die Anweisung zur Übergabe des Gehalts als auch die des Umsatzes in den try-Block einer try-catch-Anweisung eingebunden werden muss.

❺ Der catch-Block enthält den Aufruf der ausgelagerten Methode *fehlermeldung(String meldung).* Ihr wird der Text "Bitte Zahl eingeben" mitgeliefert.

❻ Ebenfalls im catch-Block wird die ausgelagerte Methode *leeren_Feld(JTextField pTextfeld)* aufgerufen und das zu leerende Textfeldobjekt mitgeliefert.

❼ Die ausgelagert Methode *fehlermeldung(String meldung)* empfängt beim Aufruf einen Textstring mit der anzuzeigenden Meldung.

❽ Die Ausgabe dieser Meldung soll mithilfe einer Messagebox erfolgen. Hierzu muss die Methode *showMessageDialog(null, "Nachricht")* der Klasse »JOptionPane« aufgerufen und der empfangene Textstring mitgeliefert werden.
Um diese Methode nutzen zu können muss die Klasse »JOptionPane« aus der Swing-Bibliothek importiert werden

```
import javax.swing.JOptionPane;
```

❾ Die ausgelagerte Methode *leeren_Feld(JTextField pTextfeld*) empfängt beim Aufruf ein Objekt der Klasse »JTextField«, dessen Inhalt gelöscht werden soll.

❿ Die Methode *setText()* des empfangenen Textfeldobjekts wird aufgerufen und ein leerer Textstring wird übergeben.

("projekt_MitarbeiterGUI_1_erweitert")

Quellcode der benötigten Methoden

```
   //Mitarbeiterdaten zur Laufzeit speichern
   //Eingabefehler prüfen
   public void anlegen()   {
❶  if(txtName.getText().equals("")) {
          fehlermeldung("Bitte Name eingeben");
      }
      else {
❸         einMitarbeiter.setName(txtName.getText());
      }
❶     if(txtVorname.getText().equals("")) {
❷         fehlermeldung("Bitte Vorname eingeben");
      }
      else {
❸         einMitarbeiter.setVorname(txtVorname.getText());
      }

❹     try
      {
        einMitarbeiter.setGehalt(Double.parseDouble(txtGehalt.
               getText()));
❹       try
        {
          einMitarbeiter.setUmsatz(Double.parseDouble(txtUmsatz.
                                                    getText()));
        }
        catch(NumberFormatException e)
        {
❺         fehlermeldung("Bitte Umsatz eingeben");
❻         leeren_Feld(txtUmsatz);
        }
      }
      catch(NumberFormatException e)
      {
❺       fehlermeldung("Bitte Gehalt eingeben");
❻       leeren_Feld(txtGehalt);
      }

   // Fehlermeldung
❼  public void fehlermeldung(String pMeldung)
   {
❽     JOptionPane.showMessageDialog(null, pMeldung);
   }


   // Leeren eines Textfeldes nach Fehlermeldung
❾  public void leeren_Feld(JTextField pTextfeld)
   {
❿     pTextfeld.setText("");
   }
```

2.5.2 Jahresgehalt und Jahresprämie ermitteln

Die grafische Benutzeroberfläche des vorhandenen Projekts soll so erweitert werden, dass nach dem Betätigen einer Schaltfläche ('btJahresgehalt') das Jahresgehalt und mit einer weiteren Schaltfläche ('btJahresprämie') die Jahresprämie des aktuellen Mitarbeiters ermittelt werden. Die Rechenergebnisse sollen in den beiden Textfeldern 'txtJahresgehalt' und 'txtJahrespraemie' angezeigt werden. Eingaben in diese Textfelder durch den Benutzer sind zu verhindern.

Das Jahresgehalt eines Mitarbeiters ergibt sich aus der Summe der zwölf Monatsgehälter.

Mitarbeiter, die einen Umsatz zwischen 100.000,00 Euro und 500.000,00 Euro (jeweils einschließlich) getätigt haben, erhalten eine Prämie in Höhe von 5 % ihres Umsatzes. Mitarbeiter, deren Umsatz unter 100.000,00 Euro liegt, erhalten keine Prämie. Für Mitarbeiter mit einem Umsatz von mehr als 500.000,00 Euro verdoppelt sich die Prämie.

Die zu gestaltende Bildschirmoberfläche soll folgenden Aufbau haben:
(Projekt: "projekt_MitarbeiterGUI_1_erweitert")

Aufgabenstellung

1) Bestimmen Sie die Funktionalitäten, um welche die Fachklasse »Mitarbeiter« erweitert werden muss.
2) Implementieren Sie diese Funktionalitäten in der Fachklasse »Mitarbeiter«.
3) Bestimmen Sie die Komponenten, um welche die Fensterklasse »Fenster« erweitert werden muss.
4) Erweitern Sie die Fensterklasse um diese Komponenten.
5) Entwickeln Sie den Quellcode der Funktionalität der Schaltfläche "Jahresgehalt ermitteln".
6) Entwickeln Sie den Quellcode der Funktionalität der Schaltfläche "Jahresprämie ermitteln".

Lösungshinweise

1) Funktionalitäten der Fachklasse »Mitarbeiter«:
   - Methode zur Berechnung des Jahresgehalts `berechneJahresgehalt()`
   - Statisches Attribut für den Prämiensatz
   - Methode zur Berechnung der Jahresprämie `berechneJahrespraemie()`

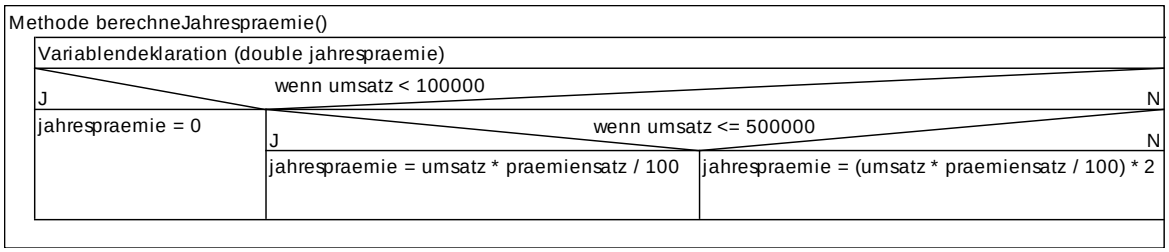
2) - Quellcode der Berechnung des Jahresgehalts (Klasse »Mitarbeiter«):

```
public double berechneJahresgehalt() {
    double jahresgehalt;
    jahresgehalt = gehalt * 12;
    return jahresgehalt;
}
```

   - Deklaration des statischen Attributs für den Prämiensatz (Klasse »Mitarbeiter«):

```
private static double praemiensatz=5;
```

   - Struktogramm zur Problemstellung "Berechnung der Jahresprämie":



   - Quellcode der Berechnung der Jahresprämie (Klasse »Mitarbeiter«):

```
public double berechneJahrespraemie() {
    double jahrespraemie;
    if (umsatz < 100000) {
      jahrespraemie = 0;
    }
    else if (umsatz <= 500000) {
      jahrespraemie = umsatz * praemiensatz / 100;
    }
    else {
      jahrespraemie = (umsatz * praemiensatz / 100) * 2;
    }
    return jahrespraemie;
}
```

3) - Labels für die Beschriftungen "Jahresgehalt" (*lblJahresgehalt*) und "Jahresprämie" (*lblJahrespraemie*)
   - Textfelder für die Ausgabe des Jahresgehalts (*txtJahresgehalt*) und der Jahresprämie (*txtJahrespraemie*)

     Da diese Textfelder vom Anwender nicht editierbar sein sollen, muss für diese Textfelder die Eigenschaft *editable* auf *false* gesetzt werden.

     (Hinweis: Die Eigenschaft *editable* = *false* führt dazu, dass die Feldinhalte nicht geändert, aber kopiert werden können.
     Die Eigenschaft *enabled* = *false* zeigt die Feldinhalte an, lässt aber keine weiteren Aktionen des Anwender zu.

   - Schaltflächen für die Ermittlung und Anzeige des Jahresgehalts (*btJahresgehalt*) und der Jahresprämie (*btJahrespraemie*).

5) Schaltfäche „Jahresgehalt ermitteln“

```
btJahresgehalt.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)  {
     berechneJahresgehalt();
   }
});
btJahresgehalt.setBounds(24, 349, 200, 23);
contentPane.add(btJahresgehalt);

//selbst erstellte Methoden
//Jahresgehalt ermitteln
public void berechneJahresgehalt() {
   txtJahresgehalt.setText(Double.toString(einMitarbeiter.
                                        berechneJahresgehalt()));
}
```

6) Schaltfäche „Jahresprämie ermitteln“

```
btJahrespraemie.addMouseListener(new MouseAdapter()
{
   public void mouseClicked(MouseEvent e)
   {
     berechneJahrespraemie();
   }
});
btJahrespraemie.setBounds(282, 349, 200, 23);
contentPane.add(btJahrespraemie);


//selbst erstellte Methoden
//Jahresprämie ermitteln
public void berechneJahrespraemie() {
   txtJahrespraemie.setText(Double.toString(einMitarbeiter.
                                        berechneJahrespraemie()));
}
```

Hinweis: Den Quellcode zum Projekt "projekt_MitarbeiterGUI_1_erweitert" finden Sie im Anhang.

3 GUI-Anwendung mit Testdaten

3.1 Projektbeschreibung

Die GeLa GmbH möchte ihre Mitarbeiterdaten abteilungsbezogen auswerten können. Zum einen sollen die Daten (Name, Vorname, Abteilung) eines Mitarbeiters anhand seiner Mitarbeiternummer aus einer zur Verfügung stehenden Liste von Mitarbeitern ausgewählt und am Bildschirm angezeigt werden. Zum anderen sollen mithilfe der Abteilungsnummer aus einer zur Verfügung stehenden Liste die zugehörige Abteilung sowie die Daten (Name, Vorname, Gehalt) der in dieser Abteilung beschäftigten Mitarbeiter ausgewählt und am Bildschirm aufgelistet werden. Darüber hinaus soll die Summe aller in der Abteilung aufgewendeten Gehälter ermittelt und am Bildschirm angezeigt werden.
Hierzu wurde bereits ein unvollständiges Javaprojekt "projekt_MitarbeiterGUI_2" entwickelt, das folgende Komponenten enthält:

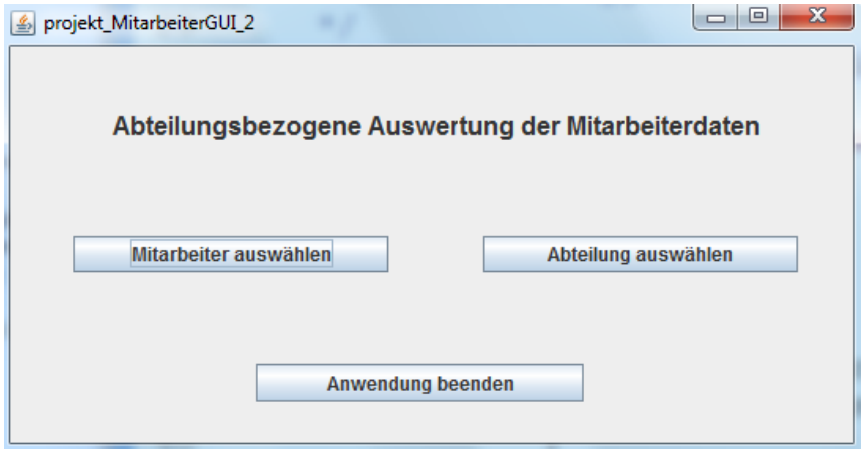
- Das Paket *paket_Fachklassen* mit den Klassen »Abteilung« und »Mitarbeiter«.
  (Siehe Quellcode der Klassen Klassen »Mitarbeiter« und »Abteilung«; Anhang Seite 50ff.)

UML-Klassendiagramm

- Das Paket *paket_Testdaten* mit der Klasse »Testdaten«. Sie stellt zum einen den Container *datenMitarbeiter* vom Typ ArrayList zur Verfügung, der zehn Mitarbeiterobjekte enthält. Zum anderen wird der Container (ArrayList) *datenAbteilungen* zur Verfügung gestellt, der drei Abteilungsobjekte mit den jeweils zugehörigen Mitarbeiterobjekten enthält.
  (Siehe Quellcode der Klasse »Testdaten« des Projekts "projekt_MitarbeiterGUI_2" sowie Anmerkungen zum Quellcode der Klasse »Testdaten«; Anhang Seite 53ff.)

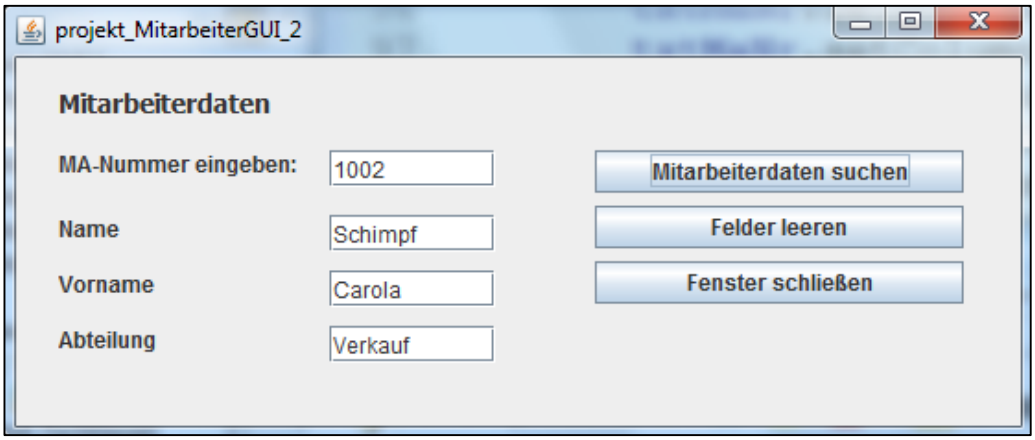
- Das Paket *paket_GUI* mit den GUI-Klassen »Fenster_Abteilung«, »Fenster_Mitarbeiter« und »Hauptfenster«.
  (Siehe Quellcode der unvollständigen Klassen Klassen »Fenster_Abteilung«, »Fenster_Mitarbeiter« und »Hauptfenster«; Anhang Seite 55ff.)

  Die Klasse »Hauptfenster« hat folgenden Aufbau:



Anmerkungen:
- Die Schaltfläche "Mitarbeiter auswählen" bewirkt, dass ein Objekt der Klasse »Fenster_Mitarbeiter« erzeugt und am Bildschirm angezeigt wird.
- Die Schaltfläche "Abteilung auswählen" bewirkt, dass ein Objekt der Klasse »Fenster_Abteilung« erzeugt und am Bildschirm angezeigt wird.
- Die Schaltfläche "Anwendung beenden" beendet die aktuelle Anwendung.

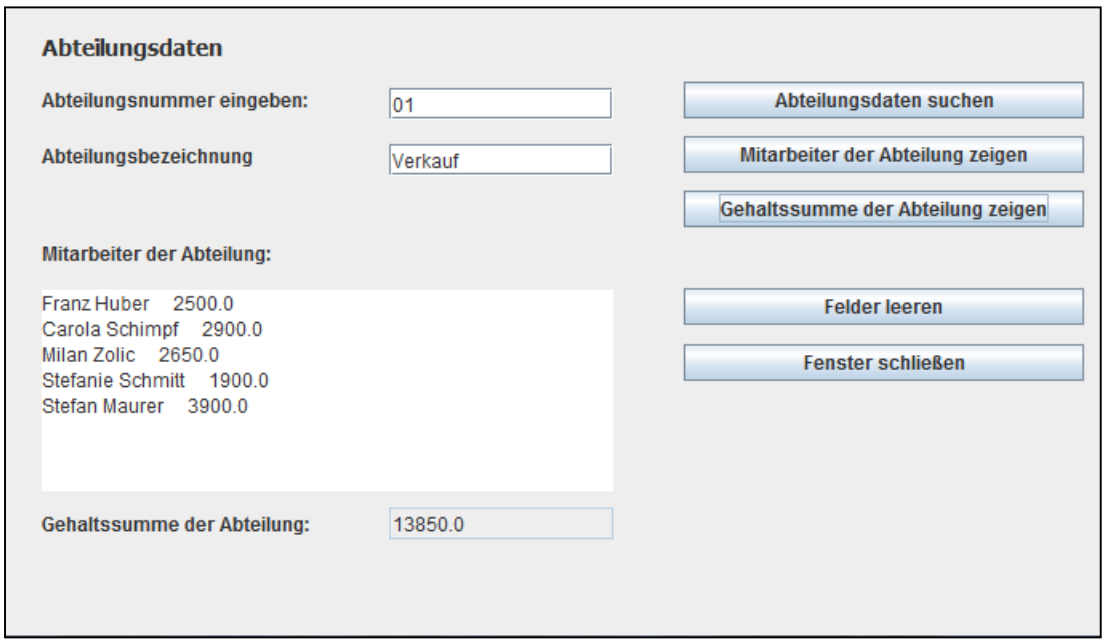
Die Klasse »Fenster_Mitarbeiter« hat folgenden Aufbau:



Anmerkungen:
- Die Schaltfläche "Mitarbeiterdaten suchen" bewirkt, dass aus den in der Klasse »Testdaten« zur Verfügung gestellten Daten die gewünschten Informationen gesucht und diese in den entsprechenden Textfeldern angezeigt werden.
- Die Funktion der Schaltfläche "Felder leeren" ist selbsterklärend.
- Die Schaltfläche "Fenster schließen" entfernt die Einträge in den Textfeldern und beendet die Ansicht des Fensters 'Mitarbeiterdaten'.

(Die Funktionalitäten der Schaltflächen "Mitarbeiterdaten suchen" und "Fenster schließen" sind noch nicht entwickelt.)

Die Klasse »Fenster_Abteilung« hat folgenden Aufbau:



Anmerkungen:
- Die Schaltfläche ''Abteilungsdaten suchen'' bewirkt, dass aus den in der Klasse »Testdaten« zur Verfügung gestellten Daten die Bezeichnung zur gesuchten Abteilung gesucht und diese in dem entsprechenden Textfeld angezeigt wird.
- Die Schaltflächen ''Mitarbeiter der Abteilung zeigen'' bewirkt, dass zu der ausgewählten Abteilung die in der Klasse »Testdaten« zugeordneten und zur Verfügung gestellten Daten ausgelesen und im entsprechenden Textfeld bzw. in der Textarea angezeigt werden.
- Die Schaltflächen ''Gehaltssumme der Abteilung zeigen'' bewirkt, dass die Gehälter aller Mitarbeiter der aktuellen Abteilung aufsummiert und im entsprechenden Textfeld angezeigt wird.
- Die Funktionen der Schaltflächen ''Felder leeren'' und ''Fenster schließen'' entsprechen denen der Klasse »Fenster_Mitarbeiter«.

(Die Funktionalitäten der Schaltflächen ''Abteilungsdaten suchen'', 'Mitarbeiter der Abteilung zeigen'', ''Gehaltssumme der Abteilung zeigen'' und ''Fenster schließen'' sind noch nicht entwickelt.)

Arbeitsaufträge

1 Entwickeln Sie ein Struktogramm für die Suche und die Anzeige der Mitarbeiterdaten (Schaltfläche ''Mitarbeiterdaten suchen'' - Klasse »Fenster_Mitarbeiter«).

2 Entwickeln Sie den Quellcode einer Methode für die Suche und die Anzeige der Mitarbeiterdaten und implementieren Sie Ihre Lösung.
(Schaltfläche ''Mitarbeiterdaten suchen'' - Klasse »Fenster_Mitarbeiter«).

3 Ergänzen Sie den Quellcode der Klasse »Fenster_Mitarbeiter« um weitere erforderliche Anweisungen.

4 Entwickeln Sie ein Struktogramm für die Suche und die Anzeige der Abteilungsdaten (Schaltfläche ''Abteilungsdaten suchen'' - Klasse »Fenster_Abteilung«).

5 Entwickeln Sie ein Struktogramm für die Anzeige der Mitarbeiterdaten einer Abteilung (Schaltfläche ''Mitarbeiter der Abteilung zeigen'' - Klasse »Fenster_Mitarbeiter«).

6 Entwickeln Sie ein Struktogramm für die Berechnung der Summe aller Gehälter einer Abteilung und begründen Sie, in welcher Klasse die entsprechende Methode implementiert werden soll.

7 Entwickeln Sie den Quellcode einer Methode für die Suche und die Anzeige der Abteilungsdaten und implementieren Sie Ihre Lösung.
(Schaltfläche ''Abteilungsdaten suchen'' - Klasse »Fenster_Abteilung«).

8 Entwickeln Sie den Quellcode einer Methode für die Anzeige der Mitarbeiterdaten einer Abteilung und implementieren Sie Ihre Lösung.
(Schaltfläche ''Mitarbeiter der Abteilung zeigen'' - Klasse »Fenster_Mitarbeiter«).

9 Entwickeln Sie den Quellcode einer Methode für die Berechnung der Summe aller Gehälter einer Abteilung. Bestimmen Sie die Klasse, in der die Methode zu implementieren ist und realisieren Sie Ihre Lösung.

10 Entwickeln Sie den Quellcode für die Anzeige der Summe aller Gehälter einer Abteilung und implementieren Sie Ihre Lösung.

11 Ergänzen Sie den Quellcode der Klasse »Fenster_Abteilung« um weitere erforderliche Anweisungen.

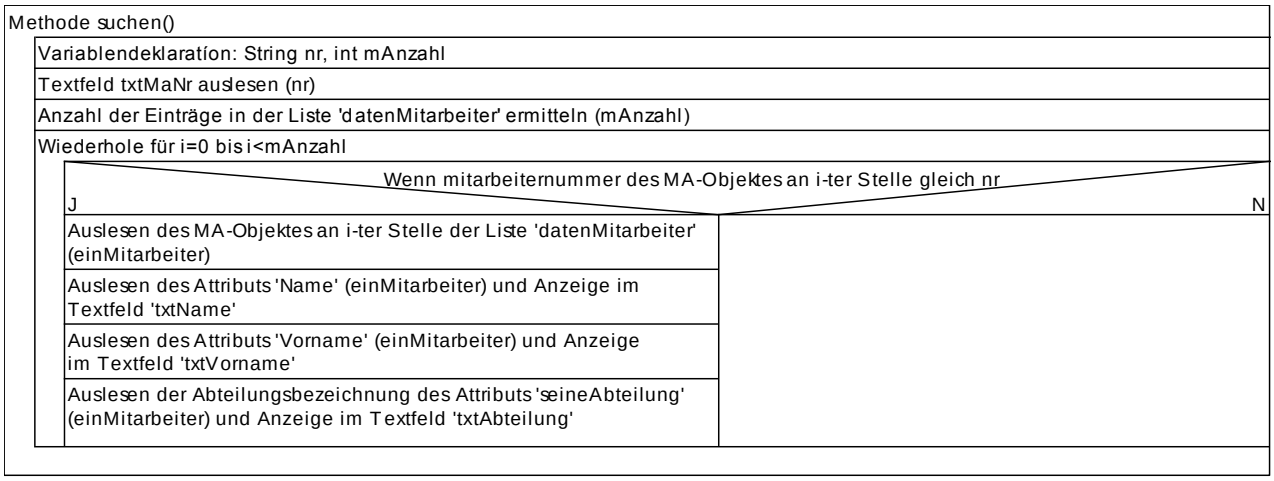
12 Die Klasse »Hauptfenster« soll dem Anwender die Auswahl zum Aufruf des Fensters „Mitarbeiterdaten“ bzw. „Abteilungsdaten“ bieten. Ergänzen Sie den Quellcode, um den die Klasse »Hauptfenster« zu erweitern ist und implementieren Sie Ihre Lösung.

3.2 Funktionalitäten der Schaltflächen im Fenster "Mitarbeiterdaten"

Mithilfe des Fensters 'Mitarbeiterdaten' sollen die in den Testdaten zur Verfügung gestellten Informationen (Vorname, Nachname, Abteilung) eines über die Mitarbeiternummer selektierten beliebigen Mitarbeiters am Bildschirm angezeigt werden.

3.2.1 Schaltfläche "Mitarbeiterdaten suchen"

Struktogramm 'Suche und Anzeige der Mitarbeiterdaten' (zu Arbeitsauftrag 1)



Das UML-Klassendiagramm zeigt eine bidirektionale Assoziation zwischen den Klassen »Mitarbeiter« und »Abteilung«. Daraus folgt, dass jedes Mitarbeiterobjekt über eine Referenz zu seiner Abteilung verfügt. Um die Mitarbeiterdaten in der Klasse »Fenster_Mitarbeiter« anzeigen zu können, muss in dieser Klasse ein Mitarbeiterobjekt vorhanden sein, das die entsprechenden Daten zur Verfügung stellt. Es wird im Deklarationsbereich der Klasse »Fenster_Mitarbeiter« angelegt. (①)

Die Informationen zu einem bestimmten Mitarbeiter sollen aus der in der Klasse »Testdaten« zur Verfügung gestellten Liste von Mitarbeiterobjekten ausgewählt werden. Die Klasse »Fenster_Mitarbeiter« benötigt somit auch ein Testdatenobjekt. (②)

Beide Objekte werden in der Variablendeklaration der Fensterklasse deklariert.

Da die Klasse »Mitarbeiter« im Paket *paket_Fachklasse* und die Klasse »Testdaten« im Paket *paket_Daten* gespeichert sind, müssen diese Klassen zunächst importiert werden. (③)

Ansonsten kann aus der Klasse »Fenster_Mitarbeiter« heraus nicht auf deren Methoden zugegriffen werden.

Im Konstruktor werden anschließend die beiden Objekt instanziiert (④). Darüber hinaus sorgt der Konstruktur mit dem Aufruf der Methode *generiere Daten()* der Testdatenklasse, dass das Testdatenobjekt eine Liste mit Mitarbeiterdaten (ArrayList *datenMitarbeiter*) und eine Liste mit Abteilungsdaten (ArrayList *datenAbteilungenr*) enthält (⑤).

Mit einem Klick auf die Schaltfläche "Mitarbeiterdaten suchen" ('btZeigen') soll zunächst die vom Anwender im Textfeld 'txtMaNr' eingegebene Mitarbeiternummer ausgelesen und das gewünschte Mitarbeiterobjekt in der ArrayList *datenMitarbeiter* gesucht werden.
Aus diesem Objekt sind die Informationen Name und Vorname auszulesen und in den jeweiligen Textfeldern ('txtName', 'txtVorname') anzuzeigen.
Über die Referenz zum zugehörigen Abteilungsobjekt wird die Abteilungsbezeichnung ausgelesen und ebenfalls am Bildschirm angezeigt ('txtAbteilung').

Quellcodeergänzung Fensterklasse »Fenster_Mitarbeiter« (zu Arbeitsauftrag 2)

Importbereich:

```
③//Import der Klassen Testdaten und Mitarbeiter
  import paket_Daten.Testdaten;
  import paket_Fachklassen.Mitarbeiter;
```

Deklarationsbereich:

```
① //Mitarbeiterobjekt deklarieren
  private Mitarbeiter einMitarbeiter;
② //Testdatenobjekt deklarieren
  private Testdaten daten = new Testdaten();
```

Konstruktor:

```
④ //Mitarbeiter- und Testdatenobjekt instanziieren
  einMitarbeiter = new Mitarbeiter();
  daten = new Testdaten();
⑤ //Testdaten erzeugen
  daten.generiereDaten();
```

Selbst erstellte Methode *suchen()*:

```
  //Mitarbeiterdaten suchen und anzeigen
  private void suchen()
  {
❶   String nr = txtMaNr.getText();
❷   int mAnzahl = daten.getMitarbeiter().size();
❸   for (int i = 0; i < mAnzahl; i++)
    {
❹     if(daten.getMitarbeiter().get(i).getMitarbeiternummer().equals(nr))
      {
❺     einMitarbeiter = daten.getMitarbeiter().get(i);
❻     txtName.setText(einMitarbeiter.getName());
      txtVorname.setText(einMitarbeiter.getVorname());
❼     txtAbteilung.setText(einMitarbeiter.getSeineAbteilung().
                                                  getBezeichnung());
      }
    }
  }
```

Der Aufruf dieser Methode erfolgt mit dem mouseClicked-Event der Schaltfläche "Mitarbeiter-daten suchen".

Erläuterungen zum Quellcode der Methode *suchen()*:

❶ Auslesen des Textfeldes 'txtMaNr'. Der empfangene Wert wird der lokalen Variablen *nr* vom Typ String übergeben.

❷ Aufruf der Methode *getMitarbeiter()* des Objekts *daten* vom Typ »Testdaten». Diese Methode liefert die ArrayList *datenMitarbeiter* zurück. Zu diesem Objekt wird die Methode *size()* aufgerufen, welche die Länge der Liste (= Anzahl Einträge) zurückliefert. Der zurückgelieferte Wert wird in der lokalen Variablen mAnzahl vom Typ int gespeichert.

❸ Kopf der Zählerschleife. Die Schleife wird solange durchlaufen, solange der Zählerwert *i* kleiner als die ermittelten Anzahl der Listeneinträge in der ArrayList (mAnzahl) ist. Der Zählerwert *i* erhöht sich bei jedem Schleifendurchlauf um den Wert 1.

❹ Überprüfung des Inhalts der lokalen Variablen *nr* mit dem Attributwert '*mitarbeiternummer*' des aktuellen Mitarbeiterobjekts der zurückgelieferten ArrayList *datenMitarbeiter* (String-Vergleich).

❺ Übergabe der Referenz auf das aktuellen Mitarbeiterobjekt an das in der Fensterklasse instanziierte Objekt *einMitarbeiter*.

❻ Auslesen der Attributwerte '*name*' und '*vorname*' und Ausgabe dieser Werte in den Textfeldern 'txtName' und 'txtVorname'.

❼ Auslesen des Attributwerts '*bezeichnung*' aus dem vom aktuellen Mitarbeiterobjekt aus referenzierten Abteilungsobjekt und Ausgabe im Textfeld 'txtAbeilung'.

3.2.2 Schaltfläche "Felder leeren" (zu Arbeitsauftrag 3)

Die Funktion der Schaltfläche "Felder leeren" ist es, die Inhalte der Textfelder zu entfernen und den Fokus auf das Textfeld zur Eingabe der Mitarbeiternummer (*txtMaNr*) zu setzen.

Selbst erstellte Methode *leeren()*:

```
//Textfelder leeren
public void leeren()
{
  txtMaNr.setText("");
  txtName.setText("");
  txtVorname.setText("");
  txtAbteilung.setText("");
  txtMaNr.requestFocus();
}
```

3.2.3 Schaltfläche "Fenster schließen" (zu Arbeitsauftrag 3)

Die Schaltfläche "Fenster schließen" soll bewirken, dass alle Textfelder der GUI geleert werden und das Fenster am Bildschirm nicht mehr sichtbar ist. Hierzu wird die Methode *leeren()* aufgerufen und die Sichtbarkeit des Fensters auf *false* gesetzt.

Selbst erstellte Methode *zurueck()*:

```
// Fenster schließen
private void zurueck()
{
  leeren();
  this.setVisible(false);
}
```

3.3 Funktionalitäten der Schaltflächen im Fenster 'Abteilungsdaten'

Das Fenster 'Abteilungsdaten' soll mehrere Funktionen erfüllen Eine Aufgabe ist es, die Bezeichnung einer beliebigen Abteilung am Bildschirm anzuzeigen, die anhand der vom Anwender eingegebenen Abteilungsnummer aus den zur Verfügung gestellten Testdaten selektiert wird.

Daneben soll es möglich sein, die Mitarbeiter, die in der ausgewählten Abteilung beschäftigt sind, mit den Informationen Vorname, Nachname und Gehalt am Bildschirm anzuzeigen. Darüber hinaus soll die Summe aller Gehälter der in der ausgewählten Abteilung beschäftigten Mitarbeiter ermittelt und am Bildschirm angezeigt werden können.

Um die Abteilungs- und Mitarbeiterdaten in der Klasse »Fenster_Abteilung« anzeigen zu können, muss ein Abteilungsobjekt erzeugt werden, welches die jeweiligen Daten der Abteilung bzw. des Mitarbeiters zur Verfügung stellen. (①)

Die Informationen zu einer bestimmten Abteilung bzw. zu einem bestimmten Mitarbeiter sollen aus den in der Klasse »Testdaten« zur Verfügung gestellten Listen mit Abteilungs- und Mitarbeiterobjekten ausgewählt werden. Die Fensterklasse »Fenster_Abteilung« benötigt somit auch ein Testdatenobjekt. (②)

Diese beiden Objekte werden in der Variablendeklaration der Fensterklasse deklariert.

Da die Klasse »Abteilung« im Paket *paket_Fachklasse* und die Klasse »Testdaten« im Paket *paket_Daten* gespeichert sind, müssen diese Klassen zunächst importiert werden. (③) Ansonsten kann aus der Klasse »Fenster_Abteilung« heraus nicht auf deren Methoden zugegriffen werden.

Im Konstruktor werden anschließend die beiden Objekte instanziiert (④). Darüber hinaus sorgt der Konstruktur mit dem Aufruf der Methode *generiere Daten()* der Testdatenklasse, dass das Testdatenobjekt eine Liste mit Mitarbeiterdaten (ArrayList *datenMitarbeiter*) und eine Liste mit Abteilungsdaten (ArrayList *datenAbteilungenr*) enthält (⑤).

Quellcodeergänzungen Fensterklasse »Fenster_Abteilung«

Importbereich:

```
③ //Import der Klassen Testdaten, Abteilung
  import paket_Daten.Testdaten;
  import paket_Fachklassen.Abteilung;
```

Deklarationsbereich:

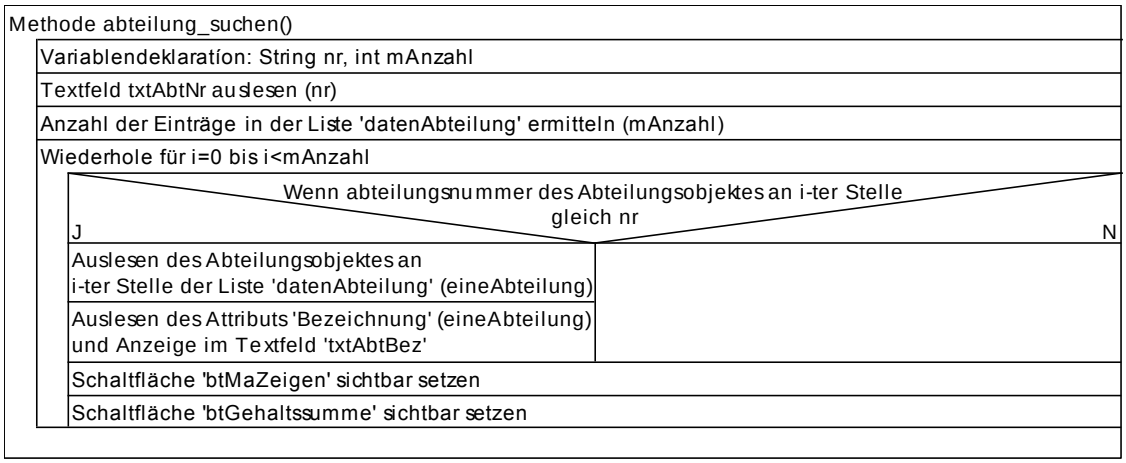
```
① //Abteilungsobjekt deklarieren
  private Abteilung eineAbteilung;
② //Testdatenobjekt deklarieren
  private Testdaten daten;
```

Konstruktor:

```
④ //Abteilungsobjekt instanziieren
  eineAbteilung = new Abteilung();
  //Testdatenobjekt instanziieren
  daten = new Testdaten();
  //Testdaten erzeugen
    daten.generiereDaten();
```

3.3.1 Schaltfläche "Abteilungsdaten suchen"

Struktogramm 'Suche und Anzeige der Abteilungsdaten' (zu Arbeitsauftrag 4)



Mit dem Klick auf die Schaltfläche "Abteilungsdaten suchen" (btAbteilungSuchen) soll zunächst die vom Anwender im Textfeld 'txtAbtNr' eingegebene Abteilungsnummer ausgelesen und das gewünschte Abteilungsobjekt in der ArrayList *datenAbteilung* gesucht werden. Aus diesem Objekt ist die Abteilungsbezeichnung auszulesen und im Textfeld ('txtAbtBez') anzuzeigen. Außerdem sollen nun die beiden Schaltflächen "Mitarbeiter der Abteilung zeigen" (btMAZeigen) und 'Gehaltssumme der Abteilung zeigen' (btGehaltssumme) am Bildschirm sichtbar werden.

Quellcodeergänzungen Fensterklasse »Fenster_Abteilung«

Selbst erstellte Methode *abteilung_suchen()*: (zu Arbeitsauftrag 7)

```
//Abteilungsdaten zeigen
public void abteilung_suchen()
{
  String nr = txtAbtNr.getText();
  int mAnzahl;
  mAnzahl = daten.getAbteilungen().size();
  for (int i = 0; i < mAnzahl; i++)
  {
    if(daten.getAbteilungen().get(i).getAbteilungsnummer().equals(nr))
    {
      eineAbteilung = daten.getAbteilungen().get(i);
      txtAbtBez.setText(eineAbteilung.getBezeichnung());
    }
  }
❶ btMaZeigen.setVisible(true);
  btGehaltssumme.setVisible(true);
}
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Abteilungsdaten suchen".

Erläuterungen zum Quellcode der Methode *abteilung_suchen()*:

Die Anweisungen dieser Methode entsprechen weitgehend denen der Methode *suchen()* aus der Klasse »Fenster_Mitarbeiter«. (Vergleiche Erläuterungen zum Quellcode der Methode *suchen())*

❶ Die gewünschten GUI-Elemente werden mit der Methode *setVisible()* sichtbar gemacht.

3.3.2 Schaltfläche "Mitarbeiter der Abteilung zeigen"

Struktogramm 'Anzeige der Mitarbeiterdaten einer Abteilung' (zu Arbeitsauftrag 5)

Methode ma_zeigen()
Label 'lblHead2' sichtbar setzen
TextArea 'taListe' sichtbar setzen
Anzahl der Einträge in der Liste 'maListe' des Objekts 'eineAbteilung' ermitteln (mAnzahl)
Wiederhole für i=0 bis i<mAnzahl
Auslesen des Attributwerts 'Vorname' des Mitarbeiterobjekts an der i-ten Stelle der Mitarbeiterliste des Objektes 'eineAbteilung' und anfügen an TextArea
Auslesen des Attributwerts 'Name' des Mitarbeiterobjekts an der i-ten Stelle der Mitarbeiterliste des Objektes 'eineAbteilung' und anfügen an TextArea
Auslesen des Attributwerts 'Gehalt' des Mitarbeiterobjekts an der i-ten Stelle der Mitarbeiterliste des Objektes 'eineAbteilung' und anfügen an TextArea

Damit die Mitarbeiterdaten der aktuellen Abteilung am Bildschirm angezeigt werden können, müssen zunächst das Label 'Mitarbeiter der Abteilung' sowie der Textbereich (TextArea) *'taListe'* sichtbar gemacht werden.

Der Zugriff auf die Mitarbeiterdaten erfolgt über das Abteilungsobjekt *eineAbteilung*, das die aus der Liste '*datenAbteilungen*' des Testdatenobjektes ausgewählte Abteilung enthält.
Jedes Abteilungsobjekt verfügt über eine Liste mit zugeordneten Mitarbeiterobjekten (ArrayList '*maListe*'). Aus dieser Liste werden die Informationen jedes Mitarbeiters ausgelesen und am Bildschirm angezeigt.

Quellcodeergänzungen Fensterklasse »Fenster_Abteilung«

Selbst erstellte Methode *ma_zeigen()*: (zu Arbeitsauftrag 8)

```
//Mitarbeiterdaten der Abteilung zeigen
public void ma_zeigen()
{
❶ lblHead2.setVisible(true);
  taListe.setVisible(true);
❷ int mAnzahl_Ma = eineAbteilung.getMaListe().size();
  for (int i = 0; i < mAnzahl_Ma; i++)
  {
❸   taListe.append(eineAbteilung.getMaListe().get(i).getVorname()+" ");
    taListe.append(eineAbteilung.getMaListe().get(i).getName()+"    ");
    taListe.append(eineAbteilung.getMaListe().get(i).getGehalt()+"\n");
  }
}
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Mitarbeiter der Abteilung zeigen".

Erläuterungen zum Quellcode der Methode *ma_zeigen()*:

❶ Die gewünschten GUI-Elemente werden mit der Methode *setVisible()* sichtbar gemacht.

❷ Aufruf der Methode *getMaListe()* des Objekts *eineAbteilung* vom Typ »Abteilung«, die eine Liste von Mitarbeiterobjekten vom Typ »ArrayList« zurückliefert. Zu diesem Objekt wird die Methode *size()* aufgerufen, welche die Länge der Liste (= Anzahl Einträge) zurückliefert. Der zurückgelieferte Wert wird in der lokalen Variablen mAnzahl_Ma vom Typ int gespeichert.

❸ Aus der Liste von Mitarbeiterobjekten vom Typ »ArrayList« des aktuellen Abteilungsobjektes wird das Mitarbeiterobjekt an der i-ten Stelle aufgerufen und mithilfe der get-Methoden die gewünschten Mitarbeiterinformationen abgefragt. Die Methode *append()* der TextArea *'taListe'* fügt dieser Liste die Informationen Vorname, Name und Gehalt an.

(Der Ausdruck `"\n"` fügt einen Zeilenumbruch in die TextArea ein.)

3.3.3 Schaltfläche "Gehaltssumme der Abteilung zeigen"

Struktogramm 'Gehaltssumme einer Abteilung berechnen' (zu Arbeitsauftrag 6)

Methode gehaltssumme()
Variablendeklaration summe = 0
Anzahl der Einträge in der Liste 'maListe' des aktuellen Objekts ermitteln (mAnzahl)
Wiederhole für i=0 bis i<mAnzahl
Auslesen des Attributs 'Gehalt' an der i-ten Stelle der Liste 'maListe' (mBetrag)
Wert der Variablen summe um den Wert der Variablen mBetrag erhöhen (summe=summe+mBetrag)
Rückgabe des Variablenwertes 'summe'

Die Ermittlung der Summe aller Gehälter einer Abteilung stellt ein fachliches Problem der Klasse Abteilung dar. Die entsprechende Methode zur Berechnung der Gehaltssumme (*gehaltssumme()*) muss somit in der Fachklasse »Abteilung« implementiert werden.

Die Schaltfläche "Gehaltssumme der Abteilung zeigen" in der Fensterklasse **»**Fenster_Abteilung« hat die Funktion, die beiden GUI-Elemente *lblGehaltssumme* und *txtGehaltssumme* sichtbar zu setzen, die Methode zur Berechnung der Gehaltssumme (*gehaltssumme()*) der Fachklasse Abteilung aufrufen und den zurückgelieferten Wert im entsprechenden Textfeld anzeigen.

Quellcodeergänzungen der Fachklasse »Abteilung«

Selbst erstellte Methode *gehaltssumme()*: (zu Arbeitsauftrag 9)

```
//Summe der Gehälter einer Abteilung ermitteln
public double gehaltssumme()
{
  double summe=0;
  for (int i = 0; i < maListe.size(); i++)
  {
    double mBetrag = maListe.get(i).getGehalt();
    summe = summe + mBetrag;
  }
  return summe;
}
```

Quellcodeergänzungen der Fensterklasse »Fenster_Abteilung«

Selbst erstellte Methode *gehaltssumme_zeigen()*: (zu Arbeitsauftrag 10)

```
//Summe der Gehälter einer Abteilung zeigen
public void gehaltsumme_zeigen()
{
  lblGehaltssumme.setVisible(true);
  txtGehaltssumme.setVisible(true);
❶ double summe = eineAbteilung.gehaltssumme();
❷ txtGehaltssumme.setText(Double.toString(summe));
}
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Gehaltssumme der Abteilung zeigen".

Erläuterungen zum Quellcode der Methode *gehaltssumme_zeigen()*:

❶ Aufruf der Methode gehaltssumme() der Fachklasse mit Übergabe an die lokale Variable *summe*.

❷ Der Rückgabewert der Methode gehaltssumme() ist vom Typ double. Bevor dieser Wert im Textfeld txtGehaltssumme angezeigt werden kann, muss es in einen Text umgewandelt werden. Hierzu wird die Methode *toString()* der Klasse »Double« verwendet.

Die Anweisungen der beiden Zeilen können auch zu einem Befehl zusammengefasst werden:

```
txtGehaltssumme.setText(Double.toString(eineAbteilung.gehaltssumme()));
```

3.3.4 Schaltfläche "Felder leeren" (zu Arbeitsauftrag 11)

Die Funktion der Schaltfläche "Felder leeren" ist es, die Inhalte der Textfelder zu entfernen, die zuvor sichtbar gesetzten GUI-Elemente wieder auszublenden und den Fokus auf das Textfeld zur Eingabe der Mitarbeiternummer (*txtAbtNr*) zu setzen.

Selbst erstellte Methode *leeren()*:

```
  public void leeren()     {
    txtAbtNr.setText("");
    txtAbtBez.setText("");
    taListe.setText("");
    lblHead2.setVisible(false);
    lblGehaltssumme.setVisible(false);
    txtGehaltssumme.setVisible(false);
    taListe.setVisible(false);
    btMaZeigen.setVisible(false);
    btGehaltssumme.setVisible(false);
    txtAbtNr.requestFocus();
❶   eineAbteilung = new Abteilung();
  }
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Felder leeren".

Erläuterungen zum Quellcode der Methode *leeren()*:

❶ Das Abteilungsobjekt *eineAbteilung* muss neu instanziiert werden, da sonst das vorhandene Objekt weiter existiert und auch nach dem Leeren aller Felder ein Klick auf die Schaltflächen 'Mitarbeiter der Abteilung zeigen' und 'Gehaltssumme der Abteilung zeigen' die zuvor ausgewählten Daten am Bildschirm angezeigt würden.

3.3.5 Schaltfläche "Fenster schließen" (zu Arbeitsauftrag 11)

Die Schaltfläche "Fenster schließen" soll bewirken, dass alle Textfelder der GUI geleert werden und das Fenster am Bildschirm nicht mehr sichtbar ist. Hierzu wird die Methode *leeren()* aufgerufen und die Sichtbarkeit des Fensters auf *false* gesetzt.

```
  public void zurueck()    {
    leeren();
    this.setVisible(false);
  }
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Fenster schließen".

3.4 Funktionalitäten der Schaltflächen im Fenster 'Abteilungsbezogene Auswertung der Mitarbeiterdaten'

Diese Bildschirmoberfläche soll dazu dienen, dem Anwender die verschiedenen zur Verfügung stehenden Fenster zur Auswahl anzubieten:

Die Schaltfläche ''Mitarbeiter auswählen'' soll ein Fensterobjekt vom Typ »Fenster_Mitarbeiter« erzeugen und dieses am Bildschirm sichtbar machen. (①)

Die Schaltfläche ''Mitarbeiter auswählen'' soll ein Fensterobjekt vom Typ »Fenster_Abteilung« erzeugen und dieses am Bildschirm sichtbar machen. (②)

Die beiden Fensterobjekte werden im Deklarationsbereich des Hauptfensters deklariert. (③)

Zu beachten ist, dass das Fenster 'Abteilungsbezogene Auswertung der Mitarbeiterdaten' weiterhin am Bildschirm angezeigt wird. Es wird lediglich von dem zweiten Fenster ('Mitarbeiterdaten' bzw. 'Abteilungsdaten') überdeckt.
Die Methoden `zurueck()` der beiden Fensterklassen »Fenster_Mitarbeiter« und »Fenster_Abteilung« setzen jeweils die Sichtbarkeit des Fensterobjekts auf *false*, so dass das Auswahlfenster wieder in den Vordergrund rückt.

Quellcodeergänzungen der Fensterklasse »Hauptfenster« (zu Arbeitsauftrag 12)

Deklarationsbereich:

```
③ private Fenster_Mitarbeiter erstesFenster;
  private Fenster_Abteilung zweitesFenster;
```

MouseClicked-Event der Schaltfläche ''Mitarbeiter auswählen''

```
① //Objekt der Klasse Fenster_Mitarbeiter erzeugen
  erstesFenster = new Fenster_Mitarbeiter();
  // erzeugtes Objekt am Bildschirm anzeigen
  erstesFenster.setVisible(true);
```

MouseClicked-Event der Schaltfläche ''Abteilung auswählen''

```
② //Objekt der Klasse Fenster_Abteilung erzeugen
  zweitesFenster = new Fenster_Abteilung();
  //erzeugtes Objekt am Bildschirm anzeigen
  zweitesFenster.setVisible(true);
```

3.5 Erweiterung der Problemstellung

Das entwickelte Projekt "projekt_MitarbeiterGUI_2" soll dahingehend erweitert werden, dass die Daten eines neuen Mitarbeiters mithilfe der grapfischen Benutzeroberfläche erfasst werden können. Dazu sind die Mitarbeiternummer, der Vor- und Zuname des Mitarbeiters sowie die Abteilung, in welcher der Mitarbeiter eingesetzt ist, in die entsprechenden Textfelder einzugeben. Die Auswahl der zuzuordnenden Abteilung soll dabei über eine Drop-Down-Liste (Combo-Box) erfolgen, die alle mit dem Container *datenAbteilungen* der Klasse »Testdaten« bereitgestellten Abteilungen enthält. (Projekt "projekt_MitarbeiterGUI_2_erweitert")

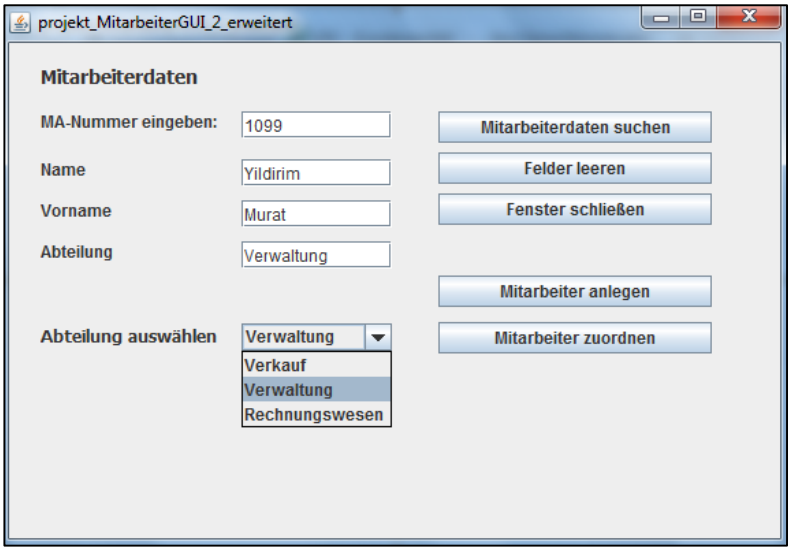
Zur Realisierung dieser Aufgabe erhält das Fenster 'Mitarbeiterdaten' weitere GUI-Elemente:

- Schaltfläche ''Mitarbeiter anlegen'' ('btNeu'),
- Schaltfläche ''Mitarbeiter zuordnen'' ('btZuordnen'),
- Drop-Down-Liste (cboAbteilung - Klasse »JComboBox«),
- Label 'Abteilung auswählen' (lblHead2)

Beim Starten des Fensters sind zunächst die Schaltfläche ''Mitarbeiter zuordnen'', die Drop-Down-Liste und der Label nicht sichtbar.

Funktion der Schaltfläche ''Mitarbeiter anlegen'' soll es sein, diese drei zusätzlichen GUI-Elemente am Bildschirm sichtbar zu machen und die Drop-Down-Liste mit Inhalt zu füllen.

Die Schaltfläche ''Mitarbeiter zuordnen'' hat die Aufgabe, die Bezeichnung der in der Drop-Down-Liste ausgewählten Abteilung in dem entsprechenden Textfeld anzuzeigen. Darüber hinaus sollen mit einem Klick auf diesen Button die Mitarbeiterinformationen in einem Objekt der Klasse »Mitarbeiter« erfasst und während der Laufzeit der Anwendung in der bereitgestellten Liste *datenMitarbeiter* der Klasse »Testdaten« gespeichert werden.



Arbeitsaufträge

1 Ergänzen Sie den Quellcode der bereits vorhandenen Methode leeren().

2 Entwickeln Sie den Quellcode für die Funktion der Schaltfläche ''Mitarbeiter anlegen'' und implementieren Sie Ihre Lösung.

3 Entwickeln Sie ein Struktogramm für die Zuordnung der ausgewählten Abteilung zu einem Mitarbeiterobjekt (Schaltfläche ''Mitarbeiter zuordnen'').

4 Entwickeln Sie den Quellcode einer Methode für die Zuordnung der ausgewählten Abteilung zu einem Mitarbeiterobjekt.

3.6 Funktionalitäten der zusätzlichen Schaltflächen

3.6.1 Schaltfläche "Mitarbeiter anlegen"

Das Fenster Mitarbeiterdaten dient einerseits der Auswahl zur Auflistung vorhandener, andererseits aber auch der Erfassung noch nicht vorhandener Mitarbeiterdaten. Die GUI-Elemente, die nur zur Erfassung von Daten benötigt werden, sollen auch erst dann am Bildschirm sichtbar sein, wenn der Anwender mit einem Klick auf die Schaltfläche "Mitarbeiter anlegen" diesen Wunsch zum Ausdruck bringt.
Deshalb muss zunächst die Methode leeren(), die mit dem Konstruktoraufruf der Klasse »Fenster_Abteilung« ausgeführt wird, um die Anweisungen erweitert werden, welche die Sichtbarkeit der zusätzlichen GUI-Elemente auf *false* setzt.

Ergänzung der selbst erstellte Methode *leeren()*: (zu Arbeitsauftrag 1)

```
public void leeren()
{
  ....
  cboAbteilung.setVisible(false);
  lblHead2.setVisible(false);
  btZuordnen.setVisible(false);
}
```

Die Aufgabe der Schaltfläche "Mitarbeiter anlegen" ist es, die Abteilungsbezeichnungen in die Drop-Down-Liste (Combo-Box) einzutragen. Zunächst sollen evtl. vorhandene Einträge aus dieser Liste entfernt werden. Im Anschluss daran müssen die Abteilungsbezeichnungen aus der vorhandenen Liste der Abteilungsdaten (*datenAbteilungen*) ausgelesen und in die Drop-Down-Liste eingetragen werden. Abschließend wird die Sichtbarkeit der drei zusätzlichen GUI-Elemente auf *true* gesetzt.

Quellcodeergänzungen Fensterklasse »Fenster_Mitarbeiter«

Selbst erstellte Methode *elementeZeigen ()* : (zu Arbeitsauftrag 2)

```
//Combobox füllen
public void elementeZeigen()
{
❶   cboAbteilung.removeAllItems();
    for (int i = 0; i < daten.getAbteilungen().size(); i++)
    {
❷     cboAbteilung.addItem(daten.getAbteilungen().get(i).
                                                  getBezeichnung());
    }
    cboAbteilung.setVisible(true);
    lblHead2.setVisible(true);
    btZuordnen.setVisible(true);
}
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Mitarbeiter anlegen".

Erläuterungen zum Quellcode der Methode *elementeZeigen()*:

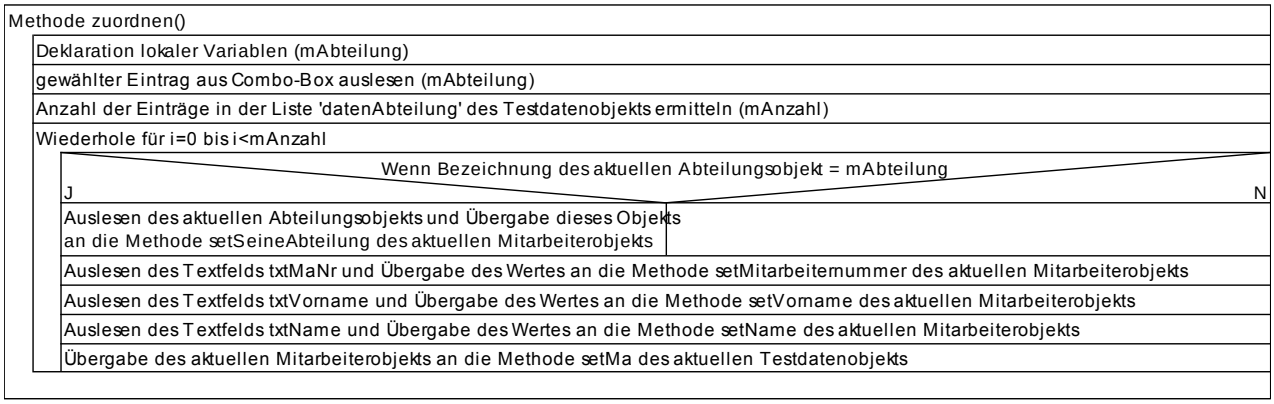
❶ `removeAllItems()` entfernt alle Einträge aus der Combo-Box '*cboAbteilung*'

❷ `addItem()` fügt der Combo-Box '*cboAbteilung*' den übergebenen Wert an.

3.6.2 Schaltfläche "Mitarbeiter zuordnen"

Die Schaltfläche "Mitarbeiter zuordnen" hat die Funktion, die in der Combo-Box ausgewählte Abteilung dem aktuellen Mitarbeiterobjekt zuzuordnen. Dazu wird der selektierte Eintrag der Combo-Box ausgelesen und in das Textfeld txtAbteilung eingetragen.
Daneben werden die Einträge der Liste der Abteilungen (*datenAbteilungen*) überprüft, ob deren Abteilungsbezeichnung mit dem selektierten Wert übereinstimmt. Trifft dies zu, so wird das gefundene Abteilungsobjekt an der i-ten Stelle der Liste datenAbteilungen ausgelesen und dem aktuellen Mitarbeiterobjekt übergeben.
Anschließend werden die verbleibenden Einträge der Textfelder (Mitarbeiternr, Name, Vorname) ausgelesen und dem aktuellen Mitarbeiterobjekt übergeben.
Zur Speicherung der Daten muss das aktuelle Mitarbeiterobjekt der Mitarbeiterliste der Klasse »Testdaten« (*datenMitarbeiter*) angefügt werden.

Struktogramm 'Zuordnung Abteilung - Mitarbeiter' (zu Arbeitsauftrag 3)



Quellcodeergänzungen Fensterklasse »Fenster_Mitarbeiter« (zu Arbeitsauftrag 4)

Selbst erstellte Methode *zuordnen()* :

```
  //ausgewählte Abteilung zuweisen
  public void zuordnen()
  {
❶   String mAbteilung;
    mAbteilung = cboAbteilung.getSelectedItem().toString();
    txtAbteilung.setText(mAbteilung);
    int mAnzahl = daten.getAbteilungen().size();
    for (int i = 0; i < mAnzahl; i++)
    {
❷     if(daten.getAbteilungen().get(i).getBezeichnung().
                                              equals(mAbteilung))
      {
❸       einMitarbeiter.setSeineAbteilung(daten.getAbteilungen().get(i));
      }
    }
    einMitarbeiter.setMitarbeiternummer(txtMaNr.getText());
    einMitarbeiter.setVorname(txtVorname.getText());
    einMitarbeiter.setName(txtName.getText());
❹   daten.setMa(einMitarbeiter);
  }
```

Der Aufruf dieser Methode erfolgt aus dem mouseClicked-Event der Schaltfläche "Mitarbeiter zuordnen".

Erläuterungen zum Quellcode der Methode *zuordnen()*:

❶ `getSelectedItem().toString();` liest den selektierten Eintrag der Combo-Box '*cboAbteilung*' aus und wandelt ihn einen String um.

❷ Überprüfung der Abteilungsbezeichnung des in der Liste *datenAbteilungen* zur Verfügung gestellten Abteilungsobjekts an der i-ten Stelle mit dem Eintrag der lokalen Variablen '*nr*'.

❸ Das gefundene Abteilungsobjekt an der i-ten Stelle der Liste *datenAbteilungen* wird ausgelesen und der Methode '*setSeineAbteilung()*' der Klasse »Mitarbeiter« übergeben.

❹ Übergabe des aktuellen Mitarbeiterobjekts an die Methode '*setMa()*' der Klasse »Testdaten«. Sie fügt das mitgelieferte Mitarbeiterobjekt der Mitarbeiterliste *datenMitarbeiter* an.

Hinweis: Den Quellcode der GUI-Klasse »Fenster_Mitarbeiter« zum Projekt "projekt_MitarbeiterGUI_2_erweitert" finden Sie im Anhang.

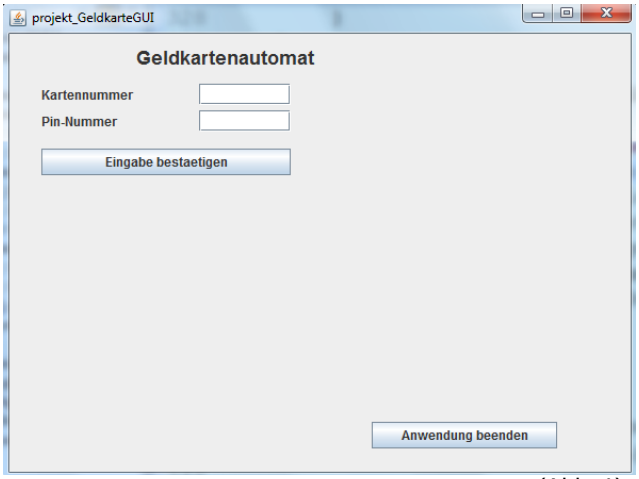
4 Projekt Geldkarte

4.1 Sachverhalt:

Ein Kreditinstitut bietet seinen Kunden eine Geldkarte zur bargeldlosen Bezahlung an. Alle Kunden, die über ein Girokonto verfügen, können Geldkarten beantragen. Von jedem Girokonto benötigt das Kreditinstitut die Kontonummer, den Inhaber des Kontos sowie den Kontostand. Für jede Geldkarte sind die Kartennummer, eine Pin-Nummer und das Kartenguthaben zu verwalten. Darüber hinaus soll jeder Geldkarte das zugehörige Girokonto zugewiesen werden können.

Für die Abwicklung der Zahlungsvorgänge (ausbezahlen, aufladen) soll eine objektorientierte Software entwickelt werden, wobei die Funktionen der Geldkarte mithilfe einer grafischen Benutzeroberfläche abrufbar sein sollen.

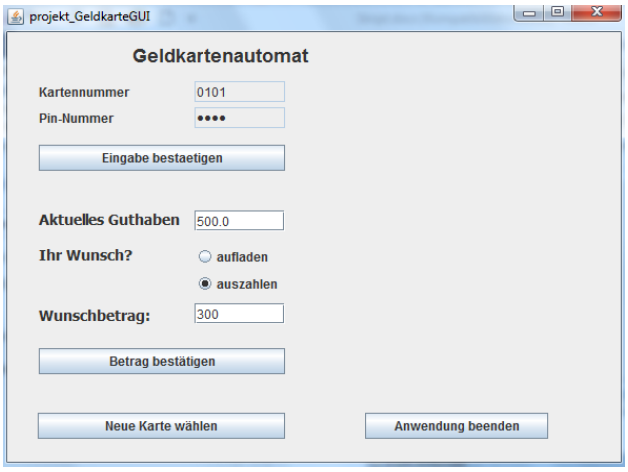
Nach dem Starten der grafischen Benutzeroberfläche sollen zunächst nur die GUI-Elemente sichtbar sein, welche zur Eingabe von Kartennummer und Pin-Nummer benötigt werden (siehe Abb. 1).



(Abb. 1)

Zuerst wird überprüft, ob die eingegebene Kartennummer mit der Nummer einer existierenden Geldkarte übereinstimmt. Ist dies nicht der Fall, so soll die Ausgabe der Meldung 'Karte gibt es nicht!' in einer Message-Box veranlasst werden. Stimmt die eingegebene Nummer mit der einer Geldkarte überein, wird überprüft, ob die eingegebene Pin-Nummer mit der hinterlegten Nummer übereinstimmt. Ist dies nicht der Fall, so soll die Ausgabe der Meldung 'Pin falsch!' in einer Message-Box veranlasst werden. In beiden Fällen hat der Anwender drei Eingabeversuche bevor das Programm abgebrochen wird. (Schaltfläche "Eingabe bestätigen")

Sind die eingegebenen Werte korrekt, so werden weitere GUI-Elemente am Bildschirm sichtbar, die zur Bedienung des „Geldkartenautomaten“ benötigt werden. Außerdem wird das aktuelle Kartenguthaben in einem Textfeld angezeigt (siehe Abb. 2).



(Abb. 2)

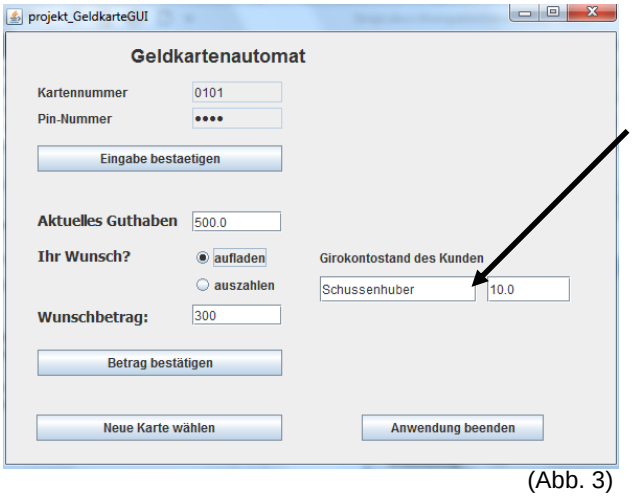
Mithilfe der Optionsfelder (RadioButtons) kann zwischen den Funktionen 'aufladen' und 'auszahlen' gewählt werden. Beim Anzeigen der Elemente soll immer das Optionsfeld 'auszahlen' selektiert werden.

Nachdem der gewünschte Betrag eingegeben und die Schaltfläche "Betrag bestätigen" aktiviert wurde, wird zuerst geprüft, ob eine positive Zahl angegeben wurde. Ist dies nicht der Fall, so soll die Meldung 'Ihr Wunschbetrag ist negativ!' in einer Message-Box erscheinen. Erst nach dieser Eingabeprüfung wird die zuvor gewählte Funktion ausgeführt.

Im Falle einer Auszahlung wird geprüft, ob das Guthaben der aktuellen Geldkarte für die gewünschte Auszahlung ausreicht. Ist dies nicht der Fall, so soll die Meldung 'Ihr Guthaben ist zu gering!' in einer Message-Box erscheinen. Ansonsten wird das Guthaben der aktuellen Geldkarte um den eingegebenen Wunschbetrag verringert, die Meldung 'Auszahlung gebucht!'

in einer Message-Box ausgegeben und das korrigierte Guthaben als aktuelles Guthaben am Bildschirm angezeigt.

Im Falle des Aufladens der Geldkarte werden weitere GUI-Elemente am Bildschirm angezeigt, die dazu dienen, den Inhaber und den Kontostand des Girokontos auszugeben, von dem der aufzuladende Betrag abgebucht werden soll (siehe Abb. 3).



(Abb. 3)

Nachdem der gewünschte Betrag eingegeben und die Schaltfläche "Betrag bestätigen" aktiviert wurde, soll geprüft werden, ob eine positive Zahl als Wunschbetrag eingegeben wurde (siehe Funktion 'auszahlen'). Unabhängig vom Kontostand des Girokontos wird das Guthaben der aktuellen Geldkarte um den gewünschten Betrag erhöht und der Kontostand des zugehörigen Girokontos entsprechend korrigiert. Sowohl das neue Guthaben der aktuellen Geldkarte als auch der korrigierte Kontostand des zugehörigen Girokontos werden in den entsprechenden Textfeldern angezeigt.

(Abb. 4)

Sollte der Kontostand des Girokontos nun negativ sein, soll ein Hinweis am Bildschirm erfolgen (siehe Abb. 4).

Arbeitsaufträge

1 Analysieren Sie den Sachverhalt und erstellen Sie ein UML-Klassendiagramm.

Ein Software-Entwickler-Team hat bereits mit der Realisierung des Geldkartenprojekts begonnen und ein unvollständiges Java-Projekt "projekt_GeldkarteGUI_Vorlage" mit folgenden Komponenten vorgelegt:

- Das Paket *paket_Fachklassen* mit den Klassen »Geldkarte« und »Girokonto« (siehe Quellcode im Anhang Seite 61ff).
- Das Paket *paket_Testdaten* mit der Klasse »Testdaten«. Sie stellt für Testzweicke Geldkarten- und Girokontenobjekte in den Containern (ArrayList) *girokontenliste* und *geldkartenliste* zur Verfügung (siehe Quellcode im Anhang Seite 62f).
- Das Paket *paket_GUI* mit der Klasse »Hauptfenster«, das den Anwender zur Kommunikation mit dem Programm dient.

2 Ergänzen Sie den Quellcode der GUI-Klasse »Hauptfenster«, so dass beim Öffnen des Fensters lediglich die Elemente erscheinen, die zur Eingabe von Karten- und Pin-Nummer erforderlich sind (siehe Abb. 1).

3 Entwickeln Sie ein Struktogramm zur Überprüfung der eingegebenen Pin-Nummer mit den Daten der vorhandenen Test-Geldkarten und begründen Sie, in welcher Klasse die entsprechende Methode zu implementieren ist.

4 Entwickeln Sie den Quellcode einer Methode für die Überprüfung der Pin-Nummer und implementieren Sie Ihre Lösung.

5 Entwickeln Sie ein Struktogramm für die Funktionalität der Schaltfläche "Eingabe bestaetigen".

6 Entwickeln Sie den Quellcode für eine Methode, welche die fehlenden GUI-Elemente am Bildschirm sichtbar macht (siehe Abb. 2). Zu beachten ist, dass bei den Optionsfeldern immer die Option „auszahlen“ selektiert sein soll.

7 Entwickeln Sie den Quellcode für eine Methode, welche die Funktionalität der Schaltfläche "Eingabe bestaetigen" realisiert.

8 Entwickeln Sie den Quellcode einer Methode, die den Namen des Inhabers und den Kontostand des Girokontos anzeigt, das einem aktuellen Geldkartenobjekt zugeordnet ist. Bestimmen Sie, durch welche Aktion diese Methode aufgerufen werden soll und implementieren Sie Ihre Lösung.

9 Entwickeln Sie ein Struktogramm zur Prüfung des eingegebenen Wunschbetrags hinsichtlich seiner Höhe und seiner Deckung durch das Kartenguthaben und begründen Sie, in welcher Klasse die entsprechende Methode zu implementieren ist.

10 Entwickeln Sie den Quellcode für eine Methode zur Prüfung des eingegebenen Wunschbetrags und implementieren Sie Ihre Lösung.

11 Entwickeln Sie den Quellcode der Methode zur Auszahlung des Wunschbetrags. Bestimmen Sie die Klasse, in der diese Methode zu implementieren ist und realisieren Sie Ihre Lösung.

12 Entwickeln Sie den Quellcode der Methode zum Aufladen der Geldkarte. Bestimmen Sie die Klasse, in der diese Methode zu implementieren ist und realisieren Sie Ihre Lösung.

13 Entwickeln Sie ein Struktogramm für die Funktionalität der Schaltfläche "Betrag bestaetigen".

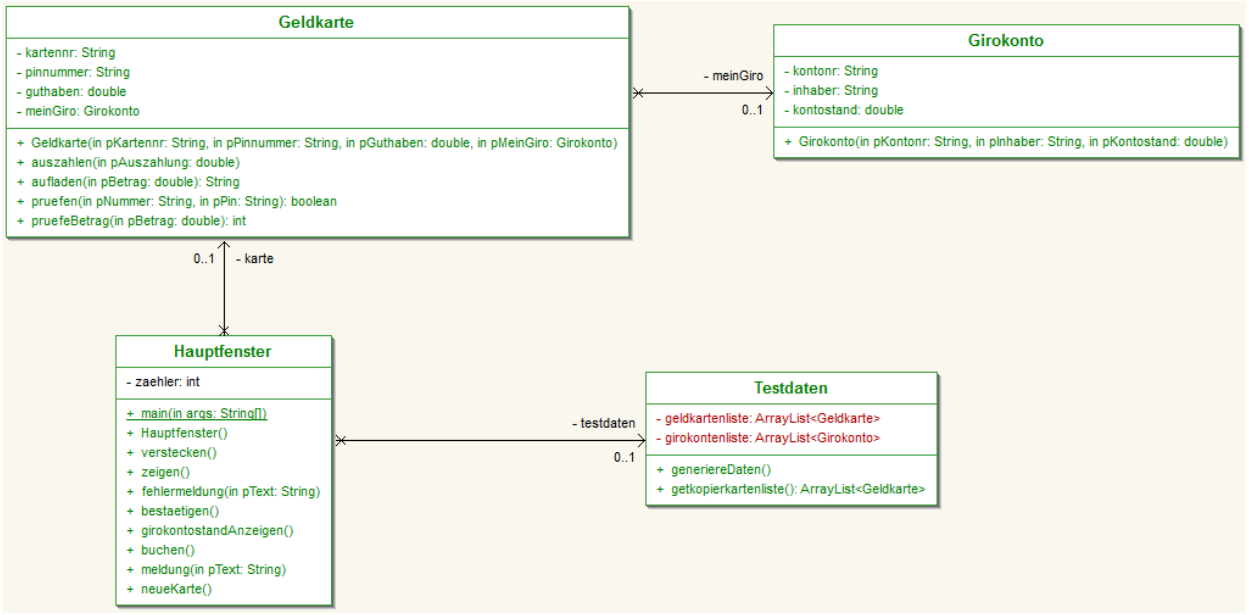
14 Entwickeln Sie den Quellcode für eine Methode, welche die Funktionalität der Schaltfläche "Betrag bestaetigen" realisiert.

15 Entwickeln Sie den Quellcode für eine Methode, welche die Funktionalität der Schaltfläche "Neue Karte wählen" realisiert.

4.2 Lösungshinweise zum Projekt "projekt_GeldkarteGUI"

4.2.1 UML-Klassendiagramm - (zu Arbeitsauftrag 1)

Auf Grundlage der Sachverhaltsanalyse ergibt sich folgendes UML-Klassendiagramm:



4.2.2 Ausblenden der nicht gewünschten GUI-Elemente

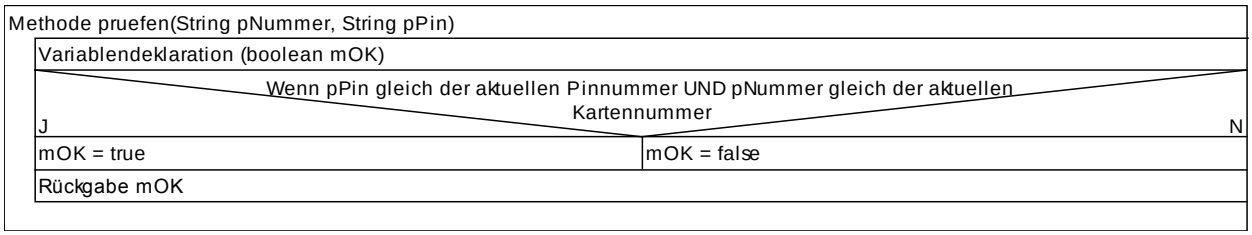
Quellcodeergänzung der GUI-Klasse »Hauptfenster« - (zu Arbeitsauftrag 2)

```
//selbst erstellte Methoden
public void verstecken()  {
  lblBetrag.setVisible(false);
  lblKontostand.setVisible(false);
  lblGiro.setVisible(false);
  lblBild.setVisible(false);
  lblWunsch.setVisible(false);
  tfBetrag.setVisible(false);
  tfGiroKunde.setVisible(false);
  tfGiroBetrag.setVisible(false);
  tfRest.setVisible(false);
  btnBestaetigen.setVisible(false);
  btNeueingabe.setVisible(false);
  rbAuszahlung.setVisible(false);
  rbAufladen.setVisible(false);
  rbAuszahlung.setSelected(true);
}
```

Der Aufruf dieser Methode erfolgt aus dem Konstruktor der GUI-Klasse »Hauptfenster«.

4.2.3 Überprüfung der eingegebenen Pin-Nummer

Struktogramm 'prüfen der Pin-Nummer' - (zu Arbeitsauftrag 3)



➔ Methode der Fachklasse »Geldkarte«

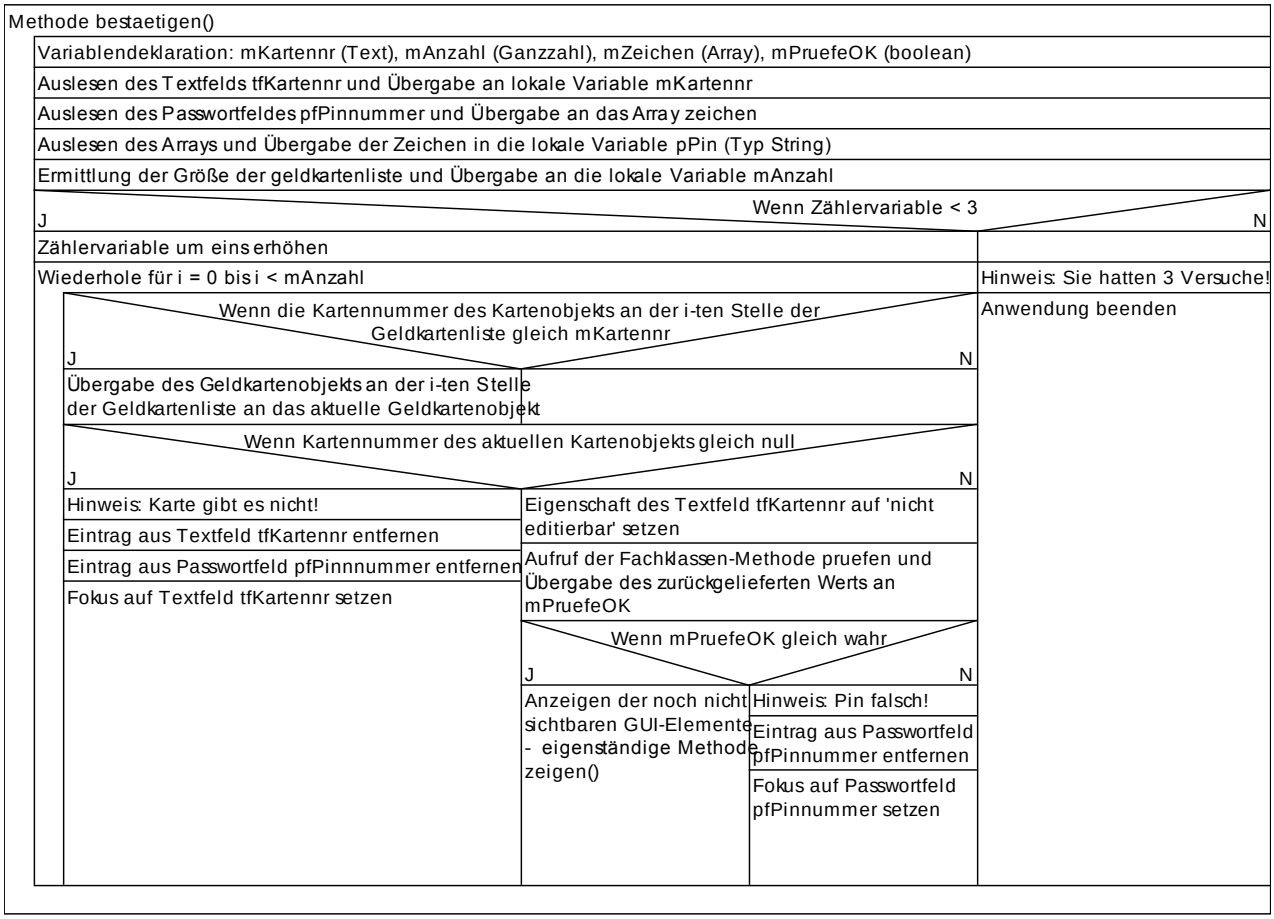
Quellcodeergänzung der Fachklasse »Geldkarte« - (zu Arbeitsauftrag 4)

```
//Methode zur Überprüfung der eingegebenen Pin-Nummer
public boolean pruefen(String pNummer, String pPin)  {
  boolean mOK;
  if((pPin.equals(pinnummer)) && pNummer.equals(kartennr))  {
    mOK = true;
  }
  else  {
    mOK = false;
  }
  return mOK;
}
```

Der Aufruf dieser Methode erfolgt über die Schaltfläche "Eingabe bestaetigen" der GUI-Klasse »Hauptfenster«.

4.2.4 Schaltfläche "Eingabe bestaetigen"

Struktogramm 'Eingabe bestaetigen' - (zu Arbeitsauftrag 5)



Quellcodeergänzung der GUI-Klasse »Hauptfenster« - (zu Arbeitsauftrag 6)

```
//Anzeige notwendiger GUI-Elemente
public void zeigen()
{
  tfBetrag.setVisible(true);
  lblBetrag.setVisible(true);
  btnBestaetigen.setVisible(true);
  btNeueingabe.setVisible(true);
  lblKontostand.setVisible(true);
  tfRest.setVisible(true);
  lblWunsch.setVisible(true);
  rbAuszahlung.setVisible(true);
  rbAufladen.setVisible(true);
  pfPinnummer.setEditable(false);
  tfBetrag.requestFocus();
  // Kontostand anzeigen
  tfRest.setText(Double.toString(karte.getGuthaben()));
}
```

Der Aufruf dieser Methode erfolgt über die Schaltfläche "Eingabe bestaetigen" der GUI-Klasse »Hauptfenster«.

Quellcodeergänzung der GUI-Klasse »Hauptfenster« - (zu Arbeitsauftrag 7)
(Schaltfläche "Eingabe bestaetigen")

Deklarationsbereich:

```
❶ int mZaehler = 0;
  private Geldkarte karte;
  private Testdaten testdaten;
```

selbst erstellte Methode:

```
  // Kartenidentifikation
  public void bestaetigen()  {
    String mKartennr;
    int mAnzahl;
    char[] mZeichen;
    boolean mPruefeOK;
❷   mKartennr = tfKartennr.getText();
    mZeichen = pfPinnummer.getPassword();
    String mPin = new String(mZeichen);
    mAnzahl = testdaten.getGeldkartenliste().size();
❸   if(mZaehler < 3)
    {
      mZaehler = mZaehler + 1;
❹     for (int i = 0; i < mAnzahl; i++)
      {
❺       if(testdaten.getGeldkartenliste().get(i).
                              getKartennr().equals(mKartennr))
        {
          karte = testdaten.getGeldkartenliste().get(i);
        }
      }
❻     if(karte.getKartennr() == null)
      {
        meldung("Karte gibt es nicht!");
        tfKartennr.setText("");
        pfPinnummer.setText("");
        tfKartennr.requestFocus();
      }
❼     else
      {
        tfKartennr.setEditable(false);
        mPruefeOK = karte.pruefen(mKartennr, mPin);
        if(mPruefeOK == true)
        {
          zeigen();
        }
❽       else  {
          meldung("Pin falsch!");
          pfPinnummer.setText("");
          pfPinnummer.requestFocus();
        }
      }
    }
❾   else  {
      meldung("Tschüss - Sie hatten 3 Versuche!!!");
      System.exit(0);
    }
  }
```

```
  //Methode für alle Message-Box-Meldungen
⓾ public void meldung(String pText)
  {
    JOptionPane.showMessageDialog(this, pText);
  }
```

Erläuterungen zum Quellcode der Methode *bestaetigen()*:

❶ Zur Überprüfung von drei Falscheingaben muss eine lokale Variable mit dem Wert 0 deklariert werden. Die Deklaration kann nicht innerhalb der Methode erfolgen, da der Wert der Variablen sonst mit jedem Aufruf auf 0 zurückgesetzt würde.

❷ Die Eingabe der Pin-Nummer erfolgt in einem Passwort-Feld. Hierbei handelt es sich um ein Objekt der Swing-Klasse »JPasswordField«. Zum Auslesen der Inhalte aus einem PasswordField steht die Methode '*getPassword()*' zur Verfügung. Der Rückgabewert dieser Methode ist vom Typ Array, der eine Reihe einzelner Zeichen enthält (char[ ]).
Das bedeutet, dass der zurückgelieferte Wert in einen Text vom Typ String umgewandelt werden muss (String x = new String(y);).

❸ Überprüfung der Zählervariablen. Wenn der Wert kleiner drei ist, wird zunächst der Wert um eins erhöht. Anschließend erfolgt die Prüfung der eingegebenen Werte.

❹ Wiederholungsbedingung zum Durchlaufen der Testdatenliste *geldkartenliste*.

❺ Überprüfung der Kartennummer des Geldkartenobjekts an der i-ten Stelle der *geldkartenliste* mit der eingetragenen Kartennummer des Textfeldes *tfKartennr*. Stimmen die Werte überein, wird das gefundene Geldkartenobjekt an das aktuelle Geldkartenobjekt *'karte'* übergeben.

❻ Nach dem Durchlauf der *geldkartenliste* wird geprüft, ob ein Geldkartenobjekt übergeben wurde. Ist dies nicht der Fall, werden die Einträge im Textfeld *tfKartennr* und im Passwortfeld *pfPinnummer* entfernt, der Fokus auf das Textfeld *tfKartennr* gesetzt und die Meldung „Karte gibt es nicht!“ in einer Message-Box veranlasst.
Da in unterschiedlichen Situationen Meldungen in Message-Boxen erfolgen sollen, wird hierfür die eigenständige Methode *'meldung(String pText)* ' erstellt. Ihr wird an dieser Stelle der gewünschte Text übergeben.

❼ Sofern ein Geldkartenobjekt übergeben wurde, wird die Editiereigenschaft des Textfeldes *tfKartennr* auf *false* gesetzt und die Methode *'pruefen(String pNummer, String pPin)'* der Klasse »Geldkarte« zur Überprüfung der Pin-Nummer aufgerufen. Diese Methode liefert einen boolschen Wert zurück. Sofern der Wert *true* zurückgeliefert wird, werden die fehlenden GUI-Elemente am Bildschirm angezeigt.

❽ Im anderen Falle wird die Meldung „Pin falsch!“ in einer Message-Box veranlasst, der Eintrag im Passwortfeld *pfPinnummer* entfernt und der Fokus auf das Passwortfeld gesetzt.

❾ Wenn die Überprüfung der Zählervariablen *mZaehler* ergibt, dass eine Eingabe zum vierten Mal erfolgte (siehe ❸), wird die Meldung „Sie hatten drei Versuche!“ in einer Message-Box veranlasst und das Programm abgebrochen.

❿ Die Ausgaben der unterschiedlichen Hinweise werden beim Aufruf dieser Methode jeweils übergeben und in der Message-Box platziert.

4.2.5 Ausgabe der Daten des zugeordneten Girokontos

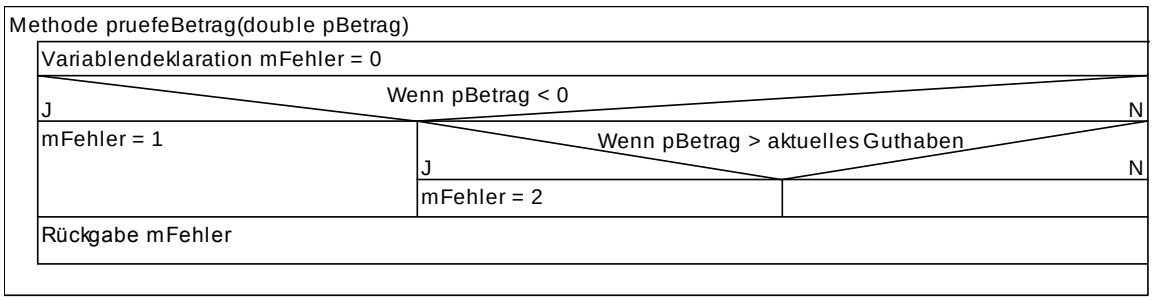
Quellcodeergänzung der GUI-Klasse »Hauptfenster« - (zu Arbeitsauftrag 8)

```
//Methode zum Anzeigen von Girokontodaten
public void girokontostandAnzeigen()
{
  lblGiro.setVisible(true);
  tfGiroKunde.setVisible(true);
  tfGiroBetrag.setVisible(true);
  String name = karte.getMeinGiro().getInhaber();
  double stand = karte.getMeinGiro().getKontostand();
  tfGiroKunde.setText(name);
  tfGiroBetrag.setText(Double.toString(stand));
}
```

Der Aufruf dieser Methode erfolgt über den RadioButton 'aufladen' der GUI-Klasse »Hauptfenster«.

4.2.6 Überprüfung der Höhe und der Deckung des Wunschbetrags

Struktogramm 'pruefeBetrag' - (zu Arbeitsauftrag 9)



Quellcodeergänzung der Fachklasse »Geldkarte« - (zu Arbeitsauftrag 10)

```
//Methode zur Überprüfung von Höhe und Deckung des Wunschbetrags
public int pruefeBetrag(double pBetrag)
{
  int mFehler=0;
  if (pBetrag<0)
  {
    mFehler=1;
  }
  else
  {
    if(pBetrag > guthaben)
    {
      mFehler=2;
    }
  }
  return mFehler;
}
```

4.2.7 Ausbezahlen eines gewünschten Betrags

Quellcodeergänzung der Fachklasse »Geldkarte« - (zu Arbeitsauftrag 11)

```
//ausbezahlen eines Wunschbetrags
public void auszahlen(double pAuszahlung)
{
  guthaben = guthaben - pAuszahlung;
}
```

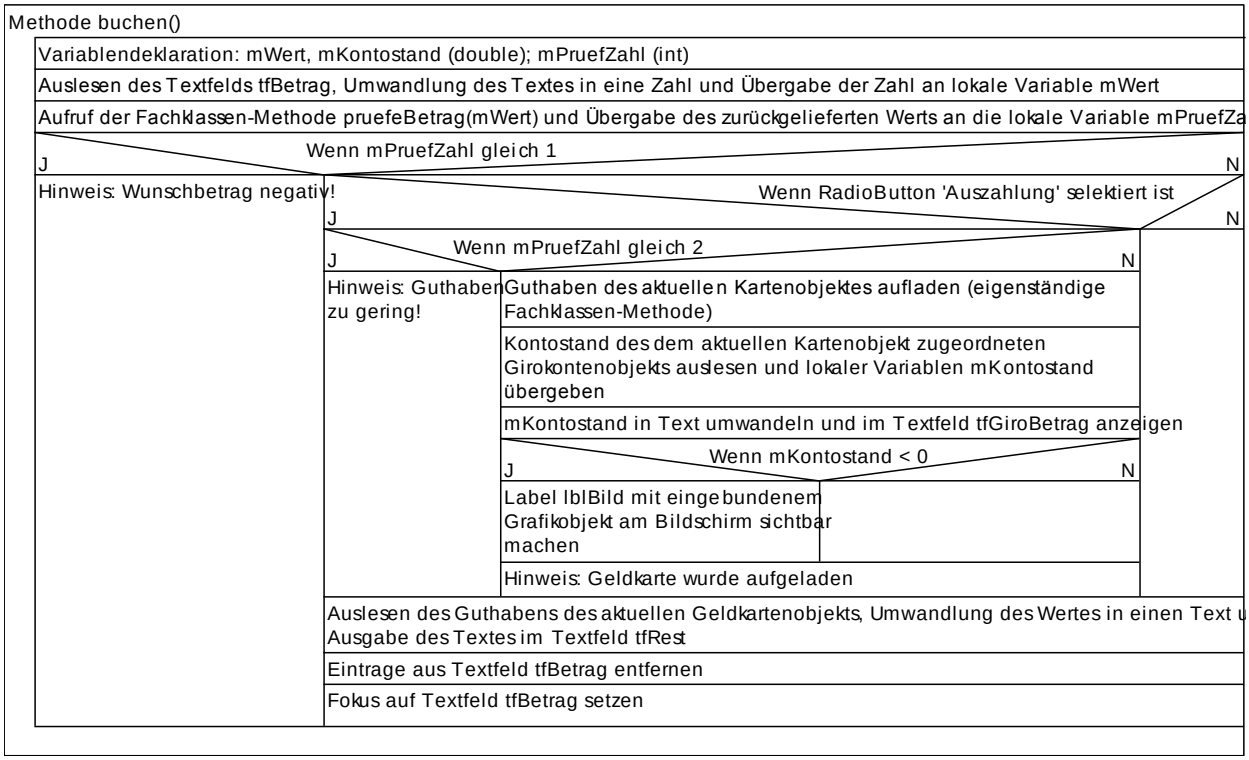
4.2.8 Aufladen einer Geldkarte

Quellcodeergänzung der Fachklasse »Geldkarte« - (zu Arbeitsauftrag 12)

```
//aufladen einer Geldkarte
public void aufladen(double pBetrag)
{
  guthaben = guthaben + pBetrag;
  meinGiro.setKontostand(meinGiro.getKontostand() - pBetrag);
}
```

4.2.9 Schaltfläche "Betrag bestaetigen"

Struktogramm 'buchen der gewählten Geldkartenaktion' - (zu Arbeitsauftrag 13)



Quellcodeergänzung der GUI-Klasse »Hauptfenster« - (zu Arbeitsauftrag 14)

```
//Buchen der gewählten Geldkartenaktion
public void buchen()     {
  double mWert, mKontostand;
  int mPruefZahl;
  mWert = Double.parseDouble(tfBetrag.getText());
  mPruefZahl = karte.pruefeBetrag(mWert);
  if(mPruefZahl == 1)
  {
    meldung("Ihr Wunschbetrag ist negativ!!!");
  }
  else
  {
    if(rbAuszahlung.isSelected())
    {
      if(mPruefZahl == 2)
      {
        meldung("Ihr Guthaben ist zu gering");
      }
      else
      {
        karte.auszahlen(mWert);
        meldung("Auszahlung gebucht!!");
      }
    }
    else
    {
      karte.aufladen(mWert);
      mKontostand = karte.getMeinGiro().getKontostand();
      tfGiroBetrag.setText(Double.toString(mKontostand));
      if(mKontostand < 0)
      {
        lblBild.setVisible(true);
      }
      meldung("Ihre Geldkarte wurde aufgeladen");
    }
  }
  tfRest.setText(Double.toString(karte.getGuthaben()));
  tfBetrag.setText("");
  tfBetrag.requestFocus();
}
```

4.2.10 Schaltfläche "Neue Karte wählen"

Quellcodeergänzung der GUI-Klasse »Hauptfenster« - (zu Arbeitsauftrag 15)

```
//leeren aller Textfelder und Ausblenden von GUI-Elementen
public void neueKarte()  {
  verstecken();
  pfPinnummer.setEditable(true);
  tfKartennr.setEditable(true);
  tfRest.setText("");
  pfPinnummer.setText("");
  tfKartennr.setText("");
  pfPinnummer.requestFocus();
}
```

Anhang

1 Quellcode zum Projekt "projekt_MitarbeiterGUI_1_erweitert"

1.1 Quellcode der Fachklasse »Mitarbeiter«

```
public class Mitarbeiter {
  private String name;
  private String vorname;
  private double gehalt;
  private double umsatz;
  private static double praemiensatz=5;

  public void setName(String pName)
  {
    name = pName;
  }

  public void setVorname(String pVorname)
  {
      vorname = pVorname;
  }

  public void setGehalt(double pGehalt)
  {
    gehalt = pGehalt;
  }

  public void setUmsatz(double pUmsatz)
  {
    umsatz = pUmsatz;
  }

  public String getName()
  {
    return name;
  }

  public String getVorname()
  {
    return vorname;
  }

  public double getGehalt()
  {
    return gehalt;
  }

  public double getUmsatz()
  {
    return umsatz;
  }

  // selbst erstellte Methoden
  // Jahresgehalt berechnen
  public double berechneJahresgehalt() {
    double jahresgehalt;
    jahresgehalt = gehalt * 12;
    return jahresgehalt;
  }
```

```
   //Jahresprämie berechnen
   public double berechneJahrespraemie() {
      double jahrespraemie;
      if (umsatz < 100000) {
         jahrespraemie = 0;
      }
      else if (umsatz <= 500000) {
         jahrespraemie = umsatz * praemiensatz / 100;
      }
      else {
         jahrespraemie = (umsatz * praemiensatz / 100) * 2;
      }
      return jahrespraemie;
   }
}
```

1.2 Quellcode der GUI-Klasse »Fenster«

Importbereich:

```
   import paket_Fachklasse.Mitarbeiter;
```

Deklarationsbereich:

```
   //Mitarbeiterobjekt deklarieren
   private Mitarbeiter einMitarbeiter;
```

Konstruktor:

```
   //Mitarbeiterobjekt instanziieren
   einMitarbeiter = new Mitarbeiter();
```

Befehlsschaltflächen:

```
   btLeeren.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent e)
      {
         leeren();
      }
   });
   btLeeren.setBounds(280, 240, 200, 23);
   contentPane.add(btLeeren);

   btAnlegen.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent e)
      {
         anlegen();
      }
   });
   btAnlegen.setBounds(23, 240, 200, 23);
   contentPane.add(btAnlegen);

   btZeigen.addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent e)
      {
         zeigen();
      }
   });
   btZeigen.setBounds(23, 285, 200, 23);
   contentPane.add(btZeigen);
```

```
btJahresgehalt.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)
   {
      berechneJahresgehalt();
   }
});
btJahresgehalt.setBounds(24, 349, 200, 23);
contentPane.add(btJahresgehalt);

btJahrespraemie.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)
   {
      berechneJahrespraemie();
   }
});
btJahrespraemie.setBounds(282, 349, 200, 23);
contentPane.add(btJahrespraemie);
```

Selbst erstellte Methoden:

```
//Textfelder leeren
public void leeren() {
   txtName.setText("");
   txtVorname.setText("");
   txtGehalt.setText("");
   txtUmsatz.setText("");
   txtJahresgehalt.setText("");
   txtJahrespraemie.setText("");
   txtName.requestFocus();
}

//Mitarbeiterdaten zur Laufzeit speichern; Eingabefehler prüfen
public void anlegen() {
   if(txtName.getText().equals(""))
   {
      fehlermeldung("Text eingeben");
   }
   else
   {
      einMitarbeiter.setName(txtName.getText());
   }
   if(txtVorname.getText().equals(""))
   {
      fehlermeldung("Text eingeben");
   }
   else
   {
      einMitarbeiter.setVorname(txtVorname.getText());
   }

   try
   {
      einMitarbeiter.setGehalt(Double.parseDouble(txtGehalt.getText()));
   }
   catch(NumberFormatException e)
   {
      fehlermeldung("Bitte Zahl eingeben");
      leeren_Feld(txtGehalt);
   }
```

```
    try
    {
      einMitarbeiter.setUmsatz(Double.parseDouble(txtUmsatz.getText()));
    }
    catch(NumberFormatException e)
    {
      fehlermeldung("Bitte Zahl eingeben");
      leeren_Feld(txtUmsatz);
    }
  }

  //Mitarbeiterdaten zeigen
  public void zeigen() {
    txtName.setText(einMitarbeiter.getName());
    txtVorname.setText(einMitarbeiter.getVorname());
    txtGehalt.setText(Double.toString(einMitarbeiter.getGehalt()));
    txtUmsatz.setText(Double.toString(einMitarbeiter.getUmsatz()));
  }

  //Fehlermeldung
  public void fehlermeldung(String pMeldung)
  {
    JOptionPane.showMessageDialog(null, pMeldung);
  }

  //Leeren eines Textfeldes nach Fehlermeldung
  public void leeren_Feld(JTextField pTextfeld)
  {
    pTextfeld.setText("");
  }

  // Jahresgehalt ermitteln
  public void berechneJahresgehalt()
  {
    txtJahresgehalt.setText(Double.toString(einMitarbeiter
         .berechneJahresgehalt()));
  }

  // Jahrespraemie berechnen
  public void berechneJahrespraemie()
  {
    txtJahrespraemie.setText(Double.toString(einMitarbeiter
         .berechneJahrespraemie()));
  }
```

2 Quellcode zum Projekt "projekt_MitarbeiterGUI_2"

Hinweis: Die in dieser Schriftart dargestellten Texte entsprechen den Lösungen zu den Arbeitsaufträgen.

2.1 Quellcode der Fachklasse »Mitarbeiter«

```
package paket_Fachklassen;
public class Mitarbeiter {
  private String mitarbeiternummer;
  private String name;
  private String vorname;
  private double gehalt;
  private double umsatz;
  private static double praemiensatz=5;
  private Abteilung seineAbteilung;

  public Mitarbeiter() {
  }

  public Mitarbeiter(String pMitarbeiternummer, String pName,
    String pVorname, double pGehalt, double pUmsatz)
  {
    mitarbeiternummer = pMitarbeiternummer;
    name = pName;
    vorname = pVorname;
    gehalt = pGehalt;
    umsatz = pUmsatz;
  }

  public void setMitarbeiternummer(String pMitarbeiternummer)
  {
    mitarbeiternummer = pMitarbeiternummer;
  }
  public void setName(String pName)
  {
    name = pName;
  }
  public void setVorname(String pVorname)
  {
      vorname = pVorname;
  }
  public void setGehalt(double pGehalt)
  {
    gehalt = pGehalt;
  }
  public void setUmsatz(double pUmsatz)
  {
    umsatz = pUmsatz;
  }
  public static void setPraemiensatz(double pPraemiensatz)
  {
    praemiensatz = pPraemiensatz;
  }
  public String getMitarbeiternummer()
  {
    return mitarbeiternummer;
  }
  public String getName()
  {
    return name;
  }
  public String getVorname()
  {
    return vorname;    }
```

```
  public double getGehalt()
  {
    return gehalt;
  }
  public double getUmsatz()
  {
    return umsatz;
  }
  public static double getPraemiensatz()
  {
    return praemiensatz;
  }

  public void setSeineAbteilung(Abteilung pSeineAbteilung)
  {
    seineAbteilung = pSeineAbteilung;
  }
  public Abteilung getSeineAbteilung()
  {
    return seineAbteilung;
  }

  //selbst erstellte Methoden
  //Jahresgehalt berechnen
  public double berechneJahresgehalt() {
    double jahresgehalt;
    jahresgehalt = gehalt * 12;
    return jahresgehalt;
  }

  //Jahresprämie berechnen
  public double berechneJahrespraemie() {
    double jahrespraemie;
    if (umsatz < 100000) {
      jahrespraemie = 0;
    }
    else if (umsatz <= 500000) {
      jahrespraemie = umsatz * praemiensatz / 100;
    }
    else {
      jahrespraemie = (umsatz * praemiensatz / 100) * 2;
    }
    return jahrespraemie;
  }
}
```

2.2 Quellcode der Fachklasse »Abteilung«

```
package paket_Fachklassen;
import java.util.ArrayList;

public class Abteilung {
  private String abteilungsnummer;
  private String bezeichnung;
  private ArrayList<Mitarbeiter> maListe = new ArrayList();

  public Abteilung()
  {
  }
```

```
   public Abteilung(String pAbteilungsnummer, String pBezeichnung)
   {
      abteilungsnummer = pAbteilungsnummer;
      bezeichnung = pBezeichnung;
   }

   public String getAbteilungsnummer()
   {
      return abteilungsnummer;
   }
   public void setAbteilungsnummer(String pAbteilungsnummer)
   {
      abteilungsnummer = pAbteilungsnummer;
   }
   public String getBezeichnung()
   {
      return bezeichnung;
   }
   public void setBezeichnung(String pBezeichnung)
   {
      this.bezeichnung = pBezeichnung;
   }
   public Mitarbeiter getMa(int pStelle)
   {
      return maListe.get(pStelle);
   }
   public void setMA(Mitarbeiter pMitarbeiter)
   {
      maListe.add(pMitarbeiter);
   }
   public int getAnzahlMitarbeiter()
   {
      return maListe.size();
   }
   public ArrayList<Mitarbeiter> getMaListe()
   {
      return maListe;
   }
   public void setMaListe(ArrayList<Mitarbeiter> pMaListe)
   {
      maListe = pMaListe;
   }
   //selbst erstellte Methode
   public double gehaltssumme() {
      double summe=0;
      for (int i = 0; i < maListe.size(); i++)
      {
      summe = summe + maListe.get(i).getGehalt();
      }
      return summe;
   }
}
```

2.3 Quellcode der Datenklasse »Testdaten«

```
package paket_Daten;

import java.util.ArrayList;
import paket_Fachklassen.Abteilung;
import paket_Fachklassen.Mitarbeiter;

public class Testdaten {
❶   private ArrayList<Mitarbeiter> datenMitarbeiter = new
                                                    ArrayList<Mitarbeiter>();
❷   private ArrayList<Abteilung> datenAbteilungen = new
                                                    ArrayList<Abteilung>();
    public void generiereDaten()
    {
❸     Mitarbeiter ma1,ma2,ma3,ma4,ma5,ma6,ma7,ma8,ma9,ma10;
❹     ma1=new Mitarbeiter("1001","Huber", "Franz", 2500, 90000);
❺     datenMitarbeiter.add(ma1);
      ma2=new Mitarbeiter("1002","Schimpf", "Carola", 2900, 45000);
      datenMitarbeiter.add(ma2);
      ma3=new Mitarbeiter("1003","Graul", "Antonio", 4500,0);
      datenMitarbeiter.add(ma3);
      ma4=new Mitarbeiter("1004","Zolic", "Milan", 2650, 125000);
      datenMitarbeiter.add(ma4);
      ma5=new Mitarbeiter("1005","Kaiser", "Marius", 3500, 0);
      datenMitarbeiter.add(ma5);
      ma6=new Mitarbeiter("1006","Maric", "Julia", 5100, 90000);
      datenMitarbeiter.add(ma6);
      ma7=new Mitarbeiter("1007","Schmitt", "Stefanie", 1900, 25000);
      datenMitarbeiter.add(ma7);
      ma8=new Mitarbeiter("1008","Arslan", "Hamit", 3750, 90000);
      datenMitarbeiter.add(ma8);
      ma9=new Mitarbeiter("1009","Maurer", "Stefan", 3900, 55000);
      datenMitarbeiter.add(ma9);
      ma10=new Mitarbeiter("1010","Urru", "Franziska", 3900, 0);
      datenMitarbeiter.add(ma10);

❻     Abteilung a1, a2, a3;
❼     a1 = new Abteilung("01", "Verkauf");
      a1.setMA(ma1);
      a1.setMA(ma2);
      a1.setMA(ma4);
      a1.setMA(ma7);
      a1.setMA(ma9);
❽     datenAbteilungen.add(a1);
      a2 = new Abteilung("02", "Verwaltung");
      a2.setMA(ma3);
      a2.setMA(ma6);
      a2.setMA(ma8);
      datenAbteilungen.add(a2);
      a3 = new Abteilung("03", "Rechnungswesen");
      a3.setMA(ma5);
      a3.setMA(ma10);
      datenAbteilungen.add(a3);

❾     ma1.setSeineAbteilung(a1);
      ma2.setSeineAbteilung(a1);
      ma3.setSeineAbteilung(a2);
      ma4.setSeineAbteilung(a1);
      ma5.setSeineAbteilung(a3);
      ma6.setSeineAbteilung(a2);
      ma7.setSeineAbteilung(a1);
      ma8.setSeineAbteilung(a2);
      ma9.setSeineAbteilung(a1);
```

```
      ma10.setSeineAbteilung(a3);
    }

❿   public ArrayList<Abteilung> getAbteilungen()
    {
      return datenAbteilungen;
    }

    public ArrayList<Mitarbeiter> getMitarbeiter()
    {
      return datenMitarbeiter;
    }

    public void setMa(Mitarbeiter pMa)
    {
      datenMitarbeiter.add(pMa);
    }
}
```

Erläuterungen zum Quellcode der Fachklasse »Testdaten«

❶ Instanziieren eines Objekts *datenMitarbeiter* vom Typ ArrayList (Liste der Mitarbeiter) zur Aufnahme von Mitarbeiterobjekten.

❷ Instanziieren eines Objekts *datenAbteilung* vom Typ ArrayList (Liste der Abteilungen) zur Aufnahme von Abteilungsobjekten.

❸ Deklaration von zehn Mitarbeiterobjekten.

❹ Erzeugung (Instanziierung) eines Mitarbeiterobjekts mit Parameterübergabe der Attributwerte.

❺ Hinzufügen des erzeugten Mitarbeiterobjekts (*ma1*) zur Liste der Mitarbeiter (ArrayList *datenMitarbeiter*).

❻ Deklaration von drei Abteilungsobjekten.

❼ Erzeugung (Instanziierung) des Abteilungsobjekts (*a1*) mit Parameterübergaben der Attributwerte.

❽ Hinzufügen des erzeugten Abteilungsobjekts (*a1*) zur Liste der Abteilungen (ArrayList *datenAbteilung*).

❾ Zuweisung des Abteilungsobjekts *a1* zum Attribut seineAbteilung des Mitarbeiterobjekts *ma1*.

❿ Methode zur Ausgabe der Mitarbeiter- bzw. Abteilungsliste.

2.4 Quellcode der GUI-Klasse »Hauptfenster«

Deklarationsbereich:

```
//Fensterobjekte deklarieren
private Fenster_Mitarbeiter erstesFenster;
private Fenster_Abteilung zweitesFenster;
```

Befehlsschaltflächen:

```
btMitarbeiter.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent arg0)
   {
      // Objekt der Klasse Fenster_Mitarbeiter erzeugen
      erstesFenster = new Fenster_Mitarbeiter();
      // erzeugtes Objekt am Bildschirm anzeigen
      erstesFenster.setVisible(true);
   }
});
btMitarbeiter.setBounds(40, 120, 200, 23);
contentPane.add(btMitarbeiter);

btAbteilung.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)
   {
      // Objekt der Klasse Fenster_Abteilung erzeugen
      zweitesFenster = new Fenster_Abteilung();
      // erzeugtes Objekt am Bildschirm anzeigen
      zweitesFenster.setVisible(true);
   }
});
btAbteilung.setBounds(300, 120, 200, 23);
contentPane.add(btAbteilung);
```

2.5 Quellcode der GUI-Klasse »Fenster_Mitarbeiter«

Importbereich:

```
import paket_Daten.Testdaten;
import paket_Fachklassen.Mitarbeiter;
```

Deklarationsbereich:

```
//Mitarbeiter und Testdatenobjekt deklarieren
private Mitarbeiter einMitarbeiter;
private Testdaten daten;
```

Konstruktor

```
//Mitarbeiter- und Testdatenobjekt instanziieren
einMitarbeiter = new Mitarbeiter();
daten = new Testdaten();
//Testdaten erzeugen
daten.generiereDaten();
```

Befehlsschaltflächen

```
btZeigen.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)  {
      suchen();
   }
});
btZeigen.setBounds(314, 50, 200, 23);
contentPane.add(btZeigen);

btLeeren.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)  {
      leeren();
   }
});
btLeeren.setBounds(314, 80, 200, 23);
contentPane.add(btLeeren);

btEnde.addMouseListener(new MouseAdapter() {
   public void mouseClicked(MouseEvent e)  {
      zurueck();
   }
});
btEnde.setBounds(314, 110, 200, 23);
contentPane.add(btEnde);
```

Selbst erstellte Methoden:

```
//Mitarbeiterdaten zeigen
private void suchen() {
    String nr = txtMaNr.getText();
    int mAnzahl;
    mAnzahl = daten.getMitarbeiter().size();
    for (int i = 0; i < mAnzahl; i++)   {
        if(daten.getMitarbeiter().get(i).getMitarbeiternummer().equals(nr))  {
            einMitarbeiter = daten.getMitarbeiter().get(i);
            txtName.setText(einMitarbeiter.getName());
            txtVorname.setText(einMitarbeiter.getVorname());
            txtAbteilung.setText(einMitarbeiter.getSeineAbteilung().
            getBezeichnung());
        }
    }
}

//Textfelder leeren
public void leeren() {
    txtMaNr.setText("");
    txtName.setText("");
    txtVorname.setText("");
    txtAbteilung.setText("");
    txtMaNr.requestFocus();
}

//Fenster schließen
private void zurueck() {
    leeren();
    this.setVisible(false);
}
```

2.6 Quellcode der GUI-Klasse »Fenster_Abteilung«

Importbereich:

```
import paket_Daten.Testdaten;
import paket_Fachklassen.Abteilung;
```

Deklarationsbereich:

```
//Mitarbeiter-, Abteilungs- und Testdatenobjekt deklarieren
private Abteilung eineAbteilung;
private Testdaten daten;
```

Konstruktor

```
//Abteilungsobjekt instanziieren
eineAbteilung = new Abteilung();
//Testdatenobjekt instanziieren und Testdaten erzeugen
daten = new Testdaten();
daten.generiereDaten();
```

Befehlsschaltflächen

```
    btEnde.addMouseListener(new MouseAdapter()  {
       public void mouseClicked(MouseEvent e)  {
          zurueck();
       }
    });
    btEnde.setBounds(424, 210, 250, 23);
    contentPane.add(btEnde);//

    btLeeren.addMouseListener(new MouseAdapter()  {
       public void mouseClicked(MouseEvent e)  {
          leeren();
       }
    });
    btLeeren.setBounds(424, 175, 250, 23);
    contentPane.add(btLeeren);

    btAbteilungZeigen.addMouseListener(new MouseAdapter()  {
       public void mouseClicked(MouseEvent e)  {
          abteilung_suchen();
       }
    });
    btAbteilungZeigen.setBounds(424, 46, 250, 23);
    contentPane.add(btAbteilungZeigen);

    btMaZeigen.addMouseListener(new MouseAdapter()  {
       public void mouseClicked(MouseEvent arg0)  {
          ma_zeigen();
       }
    });
    btMaZeigen.setBounds(424, 80, 250, 23);
    contentPane.add(btMaZeigen);
    btMaZeigen.setVisible(false);

    btGehaltssumme.addMouseListener(new MouseAdapter()  {
       public void mouseClicked(MouseEvent e)  {
          gehaltsumme_zeigen();
       }
    });
    btGehaltssumme.setBounds(424, 114, 250, 23);
    contentPane.add(btGehaltssumme);
    btGehaltssumme.setVisible(false);
  }
```

Selbst erstellte Methoden:

```
// Textfelder leeren
public void leeren()  {
    txtAbtNr.setText("");
    txtAbtBez.setText("");
    taListe.setText("");
    lblHead2.setVisible(false);
    lblGehaltssumme.setVisible(false);
    txtGehaltssumme.setVisible(false);
    taListe.setVisible(false);
    txtAbtNr.requestFocus();
    btMaZeigen.setVisible(false);
    btGehaltssumme.setVisible(false);
}

//Abteilungsdaten zeigen
public void abteilung_suchen()  {
    String nr = txtAbtNr.getText();
    int mAnzahl;
    mAnzahl = daten.getAbteilungen().size();
    for (int i = 0; i < mAnzahl; i++)  {
        if(daten.getAbteilungen().get(i).getAbteilungsnummer().equals(nr)) {
            eineAbteilung = daten.getAbteilungen().get(i);
            txtAbtBez.setText(eineAbteilung.getBezeichnung());
            btMaZeigen.setVisible(true);
            btGehaltssumme.setVisible(true);
        }
    }
}

// Mitarbeiterdaten der Abteilung zeigen
public void ma_zeigen()
{
    lblHead2.setVisible(true);
    taListe.setVisible(true);
    int mAnzahl_Ma = eineAbteilung.getMaListe().size();
    for (int i = 0; i < mAnzahl_Ma; i++)  {
        taListe.append(eineAbteilung.getMaListe().get(i).getVorname() + " ");
        taListe.append(eineAbteilung.getMaListe().get(i).getName() + "     ");
        taListe.append(eineAbteilung.getMaListe().get(i).getGehalt() + "\n");
    }
}

// Summe der Gehälter einer Abteilung
public void gehaltsumme_zeigen()  {
    lblGehaltssumme.setVisible(true);
    txtGehaltssumme.setVisible(true);
    double summe = eineAbteilung.gehaltssumme();
    txtGehaltssumme.setText(Double.toString(summe));
}

// aktuelles Fenster schließen
public void zurueck()  {
    this.setVisible(false);
}
```

3 Quellcode zum Projekt "projekt_MitarbeiterGUI_2_erweitert"

Hinweis: **Die in dieser Schriftart dargestellten Texte entsprechen den Lösungen zu den Arbeitsaufträgen**.

3.1 Quellcode der GUI-Klasse »Fenster_Mitarbeiter«

Deklarationsbereich:

```
public final JLabel lblAbteilung = new JLabel("Abteilung");
public final JComboBox cboAbteilung = new JComboBox();
public final JButton btSuchen = new JButton("Mitarbeiterdaten suchen");
public final JButton btNeu = new JButton("Mitarbeiter anlegen");
public final JButton btZuordnen = new JButton("Mitarbeiter zuordnen");

//Mitarbeiter und Testdatenobjekt deklarieren
private Mitarbeiter einMitarbeiter;
private Testdaten daten;
```

Konstruktor

```
//Mitarbeiter- und Testdatenobjekt instanziieren
einMitarbeiter = new Mitarbeiter();
daten = new Testdaten();
//Testdaten erzeugen
daten.generiereDaten();
......
Leeren();
```

Befehlsschaltflächen

```
btNeu.addMouseListener(new MouseAdapter()  {
   public void mouseClicked(MouseEvent e)  {
      elementeZeigen();
   }
});
btNeu.setBounds(314, 170, 200, 23);
contentPane.add(btNeu);

btZuordnen.addMouseListener(new MouseAdapter()  {
   public void mouseClicked(MouseEvent e)  {
      zuordnen();
   }
});
btZuordnen.setBounds(314, 204, 200, 23);
contentPane.add(btZuordnen);
```

Selbst erstellte Methoden:

```
// Textfelder leeren
public void leeren()  {
   txtMaNr.setText("");
   txtName.setText("");
   txtVorname.setText("");
   txtAbteilung.setText("");
   cboAbteilung.setVisible(false);
   lblHead2.setVisible(false);
   btZuordnen.setVisible(false);
   txtMaNr.requestFocus();
}
```

```
//Mitarbeiterdaten suchen
public void suchen()
{
   String nr = txtMaNr.getText();
   int mAnzahl;
   mAnzahl = daten.getMitarbeiter().size();
   for (int i = 0; i < mAnzahl; i++)
   {
      if(daten.getMitarbeiter().get(i).getMitarbeiternummer().equals(nr))
      {
         einMitarbeiter = daten.getMitarbeiter().get(i);
         txtName.setText(einMitarbeiter.getName());
         txtVorname.setText(einMitarbeiter.getVorname());
         txtAbteilung.setText(einMitarbeiter.getSeineAbteilung()
              .getBezeichnung());
      }
   }
}
```

```
//Combobox mit Abteilungsdaten füllen
public void elementeZeigen()
{
   cboAbteilung.removeAllItems();
   for (int i = 0; i < daten.getAbteilungen().size(); i++)
   {
      cboAbteilung
            .addItem(daten.getAbteilungen().get(i).getBezeichnung());
   }
   cboAbteilung.setVisible(true);
   lblHead2.setVisible(true);
   btZuordnen.setVisible(true);
}

public void zuordnen()
{
   String mAbteilung;
   mAbteilung = cboAbteilung.getSelectedItem().toString();
   txtAbteilung.setText(mAbteilung);
   int mAnzahl = daten.getAbteilungen().size();
   for (int i = 0; i < mAnzahl; i++)
   {
      if(daten.getAbteilungen().get(i).getBezeichnung()
            .equals(mAbteilung))
      {
         einMitarbeiter.setSeineAbteilung(daten.getAbteilungen().get(i));
      }
   }
   einMitarbeiter.setMitarbeiternummer(txtMaNr.getText());
   einMitarbeiter.setVorname(txtVorname.getText());
   einMitarbeiter.setName(txtName.getText());
   daten.setMa(einMitarbeiter);
}
```

4 Quellcode zum Projekt "projekt_Geldkarte_Vorlage"

4.1 Quellcode der Fachklasse »Geldkarte«

```
package paket_Fachklassen;
public class Geldkarte
{
   private String kartennr;
   private String pinnummer;
   private double guthaben;
   private Girokonto meinGiro;

   public Geldkarte()
   {
   }

   public Geldkarte(String pKartennr, String pPinnummer, double pGuthaben,
Girokonto pMeinGiro)
   {
      kartennr = pKartennr;
      pinnummer = pPinnummer;
      guthaben = pGuthaben;
      meinGiro = pMeinGiro;
   }

   public void setKartennr(String pKartennr)
   {
     this.kartennr = pKartennr;
   }
   public void setPinnummer(String pPinnummer)
   {
     this.pinnummer = pPinnummer;
   }
   public void setGuthaben(double pGuthaben)
   {
     this.guthaben = pGuthaben;
   }
   public void setMeinGiro(Girokonto pMeinGiro)
   {
      this.meinGiro = pMeinGiro;
   }

   public String getKartennr()
   {
      return kartennr;
   }
   public String getPinnummer()
   {
      return pinnummer;
   }
   public double getGuthaben()
   {
      return guthaben;
   }
   public Girokonto getMeinGiro()
   {
      return meinGiro;
   }
```

4.2 Quellcode der Fachklasse »Girokonto«

```
package paket_Fachklassen;
public class Girokonto
{
   private String kontonr;
   private String inhaber;
   private double kontostand;

   public Girokonto() {
   }
   public Girokonto(String pKontonr, String pInhaber, double pKontostand)
   {
      kontonr = pKontonr;
      inhaber = pInhaber;
      kontostand = pKontostand;
   }

   public String getKontonr()
   {
      return kontonr;
   }
   public String getInhaber()
   {
      return inhaber;
   }
   public double getKontostand()
   {
      return kontostand;
   }
   public void setKontonr(String pKontonr)
   {
      this.kontonr = pKontonr;
   }
   public void setInhaber(String pInhaber)
   {
      this.inhaber = pInhaber;
   }
   public void setKontostand(double pKontostand)
   {
      this.kontostand = pKontostand;
   }
}
```

4.3 Quellcode der Testdatenklasse »Testdaten«

```
package paket_Daten;
import paket_Fachklassen.*;
import java.util.*;

public class Testdaten
{
   private ArrayList<Girokonto> girokontenliste = new ArrayList<Girokonto>();
   private ArrayList<Geldkarte> geldkartenliste = new ArrayList<Geldkarte>();

   public void generiereDaten()
   {
      Girokonto g1,g2,g3,g4,g5,g6,g7,g8,g9,g10;
      g1= new Girokonto("3418555", "Landowski", 100);
      girokontenliste.add(g1);
      g2= new Girokonto("8500412", "Schussenhuber", 10);
      girokontenliste.add(g2);
```

```
    g3= new Girokonto("8500412", "Süssmuth", 2000);
    girokontenliste.add(g3);
    g4= new Girokonto("3445704", "Rudolph", 1000);
    girokontenliste.add(g4);
    g5= new Girokonto("1855014", "Schinderhanns", 50);
    girokontenliste.add(g5);
    g6= new Girokonto("2782540", "Kesselschmied", 1800);
    girokontenliste.add(g6);
    g7= new Girokonto("7255720", "Kettenring", 20000);
    girokontenliste.add(g7);
    g8= new Girokonto("7255720", "Jungblut", 45000);
    girokontenliste.add(g8);
    g9= new Girokonto("7255720", "Pfenninger", 250);
    girokontenliste.add(g9);
    g10= new Girokonto("7255720", "Heißer-Kammauf", 1000);
    girokontenliste.add(g10);

    Geldkarte k1,k2,k3,k4,k5,k6,k7,k8,k9,k10;
    k1=new Geldkarte("0101", "9911", 500, g2);
    geldkartenliste.add(k1);
    k2=new Geldkarte("0102", "5544", 300, g4);
    geldkartenliste.add(k2);
    k3=new Geldkarte("0103", "6655", 250, g7);
    geldkartenliste.add(k3);
    k4=new Geldkarte("0104", "7733", 450, g1);
    geldkartenliste.add(k4);
    k5=new Geldkarte("0105", "1199", 700, g5);
    geldkartenliste.add(k5);
    k6=new Geldkarte("0106", "2288", 150, g6);
    geldkartenliste.add(k6);
    k7=new Geldkarte("0107", "3377", 50, g3);
    geldkartenliste.add(k7);
  }

  public ArrayList<Geldkarte> getGeldkartenliste()
  {
    return geldkartenliste;
  }
  public void setGeldkartenliste(ArrayList<Geldkarte> pGeldkartenliste)
  {
    this.geldkartenliste = pGeldkartenliste;
  }
}
```

5 Quellcodeergänzungen zum Projekt "projekt_Geldkarte"

5.1 Ergänzung der Fachklasse »Geldkarte«

```
  //selbst erstellte Methoden
  public void auszahlen(double pAuszahlung)
  {
    guthaben = guthaben - pAuszahlung;
  }

  public void aufladen(double pBetrag)
  {
    guthaben = guthaben + pBetrag;
    meinGiro.setKontostand(meinGiro.getKontostand() - pBetrag);
    String meldung="Ihre Geldkarte wurde aufgeladen";
  }
```

```
   public boolean pruefen(String pNummer, String pPin)
   { boolean mOK;
     if((pPin.equals(pinnummer)) && pNummer.equals(kartennr))
     { mOK = true;}
     else
     { mOK = false;}
     return mOK;
   }

   public int pruefeBetrag(double pBetrag)
   { int mFehler=0;
     if (pBetrag<0)
     {
       mFehler=1;
     }
     else
     {
       if(pBetrag > guthaben)
       {
         mFehler=2;
       }
     }
     return mFehler;
   }
}
```

5.2 Ergänzung der GUI-Klasse »Hauptfenster«

Importbereich:

```
import javax.swing.JOptionPane;
import javax.swing.JRadioButton;import javax.swing.ImageIcon;
import paket_Daten.*;
import paket_Fachklassen.*;
```

Deklarationsbereich:

```
  //JRadioButtons werden erstellt
  private JRadioButton rbAufladen;
  private JRadioButton rbAuszahlung;
  // JRadioButtons werden zur ButtonGroup hinzugefügt
  private ButtonGroup gruppe;
  //Testdaten- und Geldkartenobjekt
  private Geldkarte karte;
  private Testdaten testdaten;
  //Zählervariable deklarieren
  int mZaehler = 0;
```

Konstruktor:

```
  //Testdatenobjekt erzeugen und Testdaten generieren
  testdaten = new Testdaten();
  testdaten.generiereDaten();
  // Geldkartenobjekt erzeugen
  karte = new Geldkarte();
........
  //Einbinden einer Grafik in ein Label-Objekt
  lblBild = new JLabel("");
  lblBild.setBounds(300, 266, 240, 93);
  lblBild.setIcon(new ImageIcon(getClass().getResource("/paket_GUI/
                                                    hinweis.jpg")));
  contentPane.add(lblBild);
........
```

```
//RadioButtons einfügen
rbAufladen = new JRadioButton("aufladen");
rbAufladen.setBounds(178, 204, 111, 23);
contentPane.add(rbAufladen);
rbAuszahlung = new JRadioButton("auszahlen");
rbAuszahlung.setBounds(178, 230, 111, 23);
contentPane.add(rbAuszahlung);
gruppe = new ButtonGroup();
gruppe.add(rbAufladen);
gruppe.add(rbAuszahlung);

//Mouse-Events der Schaltflächen
btnOK = new JButton("Eingabe bestaetigen");
btnOK.addMouseListener(new MouseAdapter()  {
  public void mouseClicked(MouseEvent arg0)
  {
    bestaetigen();
  }
});
btnOK.setBounds(30, 108, 234, 25);
contentPane.add(btnOK);

btnBestaetigen = new JButton("Betrag bestätigen");
btnBestaetigen.addMouseListener(new MouseAdapter()  {
  public void mouseClicked(MouseEvent e)
  {
    buchen();
  }
});
btnBestaetigen.setBounds(30, 303, 234, 25);
contentPane.add(btnBestaetigen);

btnEnde = new JButton("Anwendung beenden");
btnEnde.addMouseListener(new MouseAdapter()  {
  public void mouseClicked(MouseEvent e)
  {
    System.exit(0);
  }
});
btnEnde.setBounds(340, 365, 174, 25);
contentPane.add(btnEnde);

btNeueingabe = new JButton("Neue Karte wählen");
btNeueingabe.addMouseListener(new MouseAdapter()  {
  public void mouseClicked(MouseEvent e)
  {
    neueKarte();
  }
});
btNeueingabe.setBounds(29, 365, 235, 25);
contentPane.add(btNeueingabe);

rbAufladen.addMouseListener(new MouseAdapter()  {
  public void mouseClicked(MouseEvent arg0)
  {
    girokontostandAnzeigen();
  }
});

//Ausblenden der nicht gewünschten GUI-Elemente
verstecken();

//Ende Konstruktor
```

```
//selbst erstellte Methoden
//GUI-Elemente ausblenden
public void verstecken()
{
   tfBetrag.setVisible(false);
   lblBetrag.setVisible(false);
   btnBestaetigen.setVisible(false);
   btNeueingabe.setVisible(false);
   lblKontostand.setVisible(false);
   lblGiro.setVisible(false);
   lblBild.setVisible(false);
   tfGiroKunde.setVisible(false);
   tfGiroBetrag.setVisible(false);
   tfRest.setVisible(false);
   lblWunsch.setVisible(false);
   rbAuszahlung.setVisible(false);
   rbAufladen.setVisible(false);
   rbAuszahlung.setSelected(true);
}


//GUI-Elemente einblenden
public void zeigen()
{
   tfBetrag.setVisible(true);
   lblBetrag.setVisible(true);
   btnBestaetigen.setVisible(true);
   btNeueingabe.setVisible(true);
   lblKontostand.setVisible(true);
   tfRest.setVisible(true);
   lblWunsch.setVisible(true);
   rbAuszahlung.setVisible(true);
   rbAufladen.setVisible(true);
   tfBetrag.requestFocus();
   pfPinnummer.setEditable(false);
   //Guthaben der Karte anzeigen
   tfRest.setText(Double.toString(karte.getGuthaben()));
}


//eigenständige Methode zur Ausgabe der Fehlermeldungen
public void fehlermeldung(String pText)
{
   JOptionPane.showMessageDialog(null, pText);
}


// Kartenidentifikation
public void bestaetigen()
{
   String mKartennr;
   int mAnzahl;
   char[] mZeichen;
   boolean mPruefeOK;
   mKartennr = tfKartennr.getText();
   mZeichen = pfPinnummer.getPassword();
   String mPin = new String(mZeichen);
   mAnzahl = testdaten.getGeldkartenliste().size();
```

```
    if(mZaehler < 3)
    {
      System.out.println(mZaehler);
      mZaehler = mZaehler + 1;
      for (int i = 0; i < mAnzahl; i++)
      {
        if(testdaten.getGeldkartenliste().get(i).getKartennr()
            .equals(mKartennr))
        {
          karte = testdaten.getGeldkartenliste().get(i);
        }
      }

      if(karte.getKartennr() == null)
      {
        meldung("Karte gibt es nicht!");
        tfKartennr.setText("");
        pfPinnummer.setText("");
        tfKartennr.requestFocus();
      }
      else
      {
        tfKartennr.setEditable(false);
        mPruefeOK = karte.pruefen(mKartennr, mPin);
        if(mPruefeOK == true)
        {
          zeigen();
        }
        else
        {
          meldung("Pin falsch");
          pfPinnummer.setText("");
          pfPinnummer.requestFocus();
        }
      }
    }
    else
    {
      meldung("Tschüss - Sie hatten 3 Versuche!!!");
      System.exit(0);
    }
  }


  public void girokontostandAnzeigen()
  {
    lblGiro.setVisible(true);
    tfGiroKunde.setVisible(true);
    tfGiroBetrag.setVisible(true);
    String name = karte.getMeinGiro().getInhaber();
    double stand = karte.getMeinGiro().getKontostand();
    tfGiroKunde.setText(name);
    tfGiroBetrag.setText(Double.toString(stand));
  }


  public void buchen()
  {
    double mWert, mKontostand;
    int mPruefZahl;
    mWert = Double.parseDouble(tfBetrag.getText());
    mPruefZahl = karte.pruefeBetrag(mWert);
```

```
    if(mPruefZahl == 1)
    {
      meldung("Ihr Wunschbetrag ist negativ!!!");
    }
    else
    {
      if(rbAuszahlung.isSelected())
      {
        if(mPruefZahl == 2)
        {
          meldung("Ihr Guthaben ist zu gering");
        }
        else
        {
          karte.auszahlen(mWert);
          meldung("Auszahlung gebucht!!");
        }
      }
      else
      {
        karte.aufladen(mWert);
        mKontostand = karte.getMeinGiro().getKontostand();
        tfGiroBetrag.setText(Double.toString(mKontostand));
        if(mKontostand < 0)
        {
          lblBild.setVisible(true);
        }
        meldung("Ihre Geldkarte wurde aufgeladen");
      }
    }
    tfRest.setText(Double.toString(karte.getGuthaben()));
    tfBetrag.setText("");
    tfBetrag.requestFocus();
  }


  public void meldung(String pText)
  {
    JOptionPane.showMessageDialog(this, pText);
  }


  public void neueKarte()
  {
    verstecken();
    pfPinnummer.setEditable(true);
    tfKartennr.setEditable(true);
    tfRest.setText("");
    pfPinnummer.setText("");
    tfKartennr.setText("");
    pfPinnummer.requestFocus();
  }
```