

## 2 Entwurf eines einfachen Telefonbuchs

Zuerst solltest du dir darüber Gedanken machen, welche Funktionen von unserem Telefonbuch erwarten.

Man kann

- eine neue Person einfügen,
- den ersten bzw. letzten Eintrag des Telefonbuchs aufrufen,
- nach bestimmten Personen suchen,
- durch das Telefonbuch rückwärts und vorwärts blättern

Diese Aufgaben sollten für den ersten Entwurf ausreichend sein.

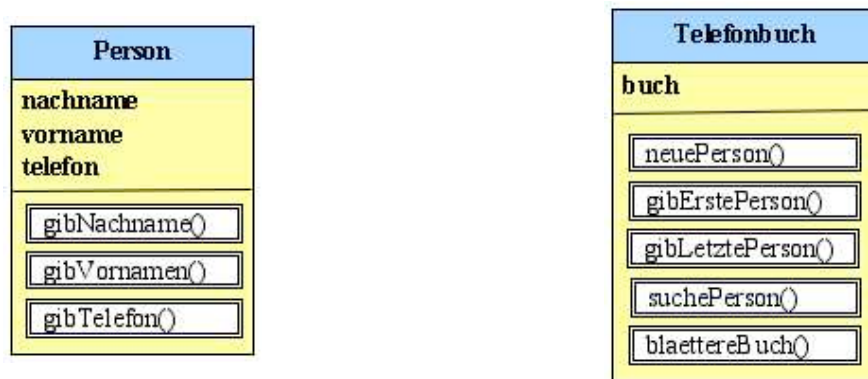


Abbildung 2.1: Modelle der Klassen *Person* und *Telefonbuch*

### 2.1 Die Klassen *Person* und *Telefonbuch*

Die Klasse *Person* muss nicht verändert werden und kann deshalb komplett aus dem Projekt *Telefon1b* des Kapitels 1 übernommen werden.

#### 2.1.1 Einfügen einer neuen Person

Das Einfügen einer neuen Person hast du bereits im ersten Kapitel gelernt.

#### Übung 2.1:

Speichere das Projekt *Telefon1b* unter dem Namen *Telefon2a*. Erzeuge eine neue Person und füge diese dem Telefonbuch hinzu. Kontrolliere mit Hilfe der Methode `zeigeAllePersonen()`, ob diese neue Person im Telefonbuch korrekt verwaltet wird.

### 2.1.2 Aufrufen des ersten bzw. letzten Eintrags

Die Java-Dokumentation zeigt, dass die Klasse *TreeSet* die beiden Methoden *first()* bzw. *last()* zur Verfügung stellt, die das erste Objekt bzw. das letzte Objekt in der sortierten *TreeSet* liefern.

```
public Person gibErstePerson()
{
    Person erstePerson = buch.first();
    return erstePerson;
}
```

**Abbildung 2.2:** Die Methode *gibErstePerson()* in der Klasse *Telefonbuch*

#### Übung 2.2:

Mit [Help > Java Class Libraries](#) kannst du in einem Browser die Java API-Dokumentation öffnen. Informiere dich hierin über die Klasse *TreeSet* und ihre Methoden.

Implementiere im Projekt *Telefon2a* die Methode *gibErstePerson()* aus [Abbildung 2.2](#). Implementiere auch eine entsprechende Methode *gibLetztePerson()*.

### 2.1.3 Suche nach bestimmten Personen

Um nach einer bestimmten Person zu suchen, wird deren Nachname als Parameter übergeben. Der Rückgabewert dieser Methode ist ein String, der die Daten dieser Person enthält. Dieser String wird in einer grafischen Benutzeroberfläche in ein Textfeld eingetragen.

```
public String suchePerson(String name)
{
    Person gefundenePerson = null;
    //somit existiert immer ein Objekt gefundenePerson

    for (Person person : buch) {
        String nachname = person.gibNachname();
        if (nachname.equals(name)) {
            gefundenePerson = person;
        }
    }
    String nachname = gefundenePerson.gibNachname();
    String vorname = gefundenePerson.gibVorname();
    String telefon = gefundenePerson.gibTelefon();
    String daten = nachname + " " + vorname + " " + telefon;
```

```
    return daten;  
}
```

**Abbildung 2.3:** Die Methode *suchePerson()* in der Klasse *Telefonbuch*

### Übung 2.3:

Formuliere schriftlich,

- welche Aufgaben die Methode *suchePerson()* in der *for-each*-Schleife erledigt,
- auf welche Weise die Methode *suchePerson()* die Daten der Person zurückgibt.

### Übung 2.4:

Implementiere die in Abbildung 2.3 dargestellte Methode *suchePerson()*. Suche nun nach den Personen „Heider“, „Schurian“ und „Tobias“.

- Warum wird (Schurian Isabella 555) erkannt, jedoch nicht (Schurian Alexander 222)?
- Warum erfolgt bei der Suche nach „Tobias“ eine Fehlermeldung?

Falls im Telefonbuch mehrere Personen mit gleichem Nachnamen existieren, müssen all diese Personen in einer Menge gesammelt werden. Um später auf eine bestimmte Person dieser Menge zugreifen zu können, verwende ich als Sammlungstyp nun eine *ArrayList*.

```
public String suchePerson(String name)  
{  
    ArrayList<Person> gefundenePersonen = new ArrayList<Person>();  
    StringBuffer alleGefundenenPersonen = new StringBuffer();  
  
    for (Person person : buch) {  
        String nachname = person.gibNachname();  
        if (nachname.equals(name)) {  
            gefundenePersonen.add(person);  
        }  
    }  
    for (Person person : gefundenePersonen) {  
        String nachname = person.gibNachname();  
        String vorname = person.gibVorname();  
        String telefon = person.gibTelefon();  
        alleGefundenenPersonen.append(nachname + " " + vorname + " "  
                                     + telefon + "\n");  
    }  
    return alleGefundenenPersonen.toString();  
}
```

```
}
```

**Abbildung 2.4:** Die erweiterte Methode *suchePerson()* in der Klasse *Telefonbuch*

In der ersten *for-each*-Schleife werden also alle gefundenen Personen in der *ArrayList* *gefundenePersonen* gesammelt. In der zweiten *for-each*-Schleife werden die Daten von allen gefundenen Personen zu einem String verarbeitet und von der Methode *suchePerson()* zurückgeliefert.

### Übung 2.5:

Verbessere die Methode *suchePerson()* wie in *Abbildung 2.4* dargestellt. Suche anschließend nach den Geschwistern „Schurian“.

### Übung 2.6:

Erweitere die Methode *suchePerson()*, so dass als Parameter auch ein Vorname eingegeben werden kann. Suche anschließend nach „Tobias“.

## 2.1.4 Das Telefonbuch durchblättern

Um durch das Telefonbuch zu blättern, muss der Nachfolger bzw. Vorgänger einer bestimmten Person gefunden werden. Deswegen wird das Telefonbuch als Sammlung vom Typ *ArrayList* gespeichert. Nun kann man nach dem Index der eingegebenen Person suchen. Wird dieser Index um 1 vergrößert bzw. verkleinert, erhält man aus dieser Liste den Nachfolger bzw. den Vorgänger.

```
public Person blaettereBuch(String nachnameH, String vornameH)
{
    int index = 0;

    ArrayList<Person> buchList = new ArrayList<Person>(buch);
    for (Person person : buchList) {
        String nachname = person.gibNachname();
        String vorname = person.gibVorname();
        if (nachname.equals(nachnameH) && vorname.equals(vornameH)) {
            index = buchList.indexOf(person);
        }
    }
    index++;
    Person person = buchList.get(index);
    return person;
}
```

**Abbildung 2.5:** Die Methode *blaettereBuch()* in der Klasse *Telefonbuch*

### Übung 2.7:

Implementiere die in Abbildung 2.5 dargestellte Methode *blaettereBuch()*.  
Verhält sich die Methode so, wie du es erwartest?

### Übung 2.8:

Übergebe als Parameter „Straka“, „Tobias“.

- Warum erscheint eine Fehlermeldung?
- Welche Person wäre ein sinnvoller Nachfolger?
- Welchen Wert sollte also der Index annehmen?
- Was bedeutet die Anweisung `index = index % buchList.size();` ?
- Implementiere diese Anweisung in der Methode *blaettereBuch()* an der richtigen Stelle. Kontrolliere nun deine Überlegungen!

Nun soll das Telefonbuch auch rückwärts durchblättert werden können. Das bedeutet, dass sich der Index um 1 erniedrigen muss. Wird der Methode *blaettereBuch()* ein weiterer Parameter *boolean vor* übergeben, so kann je nach Wert von *vor* der Index vergrößert bzw. verkleinert werden.

### Bemerkung:

Im Quelltext von Abbildung 2.5 bedeutet

*vor*        vorwärts blättern

*!vor*      rückwärts blättern

```
public Person blaettereBuch(String nachnameH,  
                           String vornameH, boolean vor)  
{  
    .....  
    .....  
    if (vor) {  
        index++;  
        index = index % buchList.size();  
    }  
    if (!vor) {  
        .....  
        .....  
    }  
    .....  
    .....  
}
```

**Abbildung 2.6:** Teil der erweiterten Methode *blaettereBuch()* in der Klasse *Telefonbuch*

## Übung 2.9:

Implementiere die in Abbildung 2.6 dargestellte Methode *blaettereBuch()* und ergänze die fehlenden Anweisungen in der zweiten *if*-Bedingung.

Kontrolliere nun deine Überlegungen!

### 2.1.5 Alle Person im Telefonbuch zeigen

In der Klasse *Telefonbuch* befindet sich die Methode *zeigeAllePersonen()*, die als einzige noch das Terminal-Fenster aufruft. Diese wird nun so gestaltet, dass sie später zum Speichern des Telefonbuchs verwendet werden kann. Im Prinzip werden die Ideen der Methode *suchePerson()* noch einmal verwendet.

```
/** Listet alle Personen im Telefonbuch auf. */
public String zeigeAllePersonen()
{
    StringBuffer allePersonen = new StringBuffer();

    for (Person person : buch) {
        String nachname = person.gibNachname();
        String vorname = person.gibVorname();
        String telefon = person.gibTelefon();
        allePersonen.append(nachname + " " + vorname + " "
                           + telefon + "\n");
    }
    return allePersonen.toString();
}
```

**Abbildung 2.7:** Die erweiterte Methode *zeigeAllePersonen()* in der Klasse *Telefonbuch*

In einer *for*-each-Schleife wird das *TreeSet buch* durchlaufen und die Daten jeder einzelnen Person in einer Zeile im *StringBuffer allePersonen* gesammelt. Die Methode *zeigeAllePersonen()* erzeugt daraus einen String und liefert diesen zurück.

## Übung 2.10:

Implementiere die in Abbildung 2.7 dargestellte Methode *zeigeAllePersonen()*. Verhält sich die Methode so, wie du es erwartest?

## 2.2 Die grafische Oberfläche

Nachdem jetzt das Telefonbuch funktioniert, kannst du dich nun der Benutzeroberfläche widmen. Diese werde ich in Java-Swing gestalten. Swing

bietet mehr Gestaltungsmöglichkeiten als das AWT (Advanced Windowing Toolkit).

Die Gestaltungsmöglichkeiten mit Swing werden im *Handbuch der Java-Programmierung* von Guido Krüger, im *The JFC Swing Tutorial, Second Edition* von Kathy Walrath u. a. oder in ähnlichen Büchern beschrieben.

In diesem Kapitel werde ich die in Abbildung 2.8 dargestellte Benutzeroberfläche beschreiben. Um die Oberfläche übersichtlich zu gestalten, habe ich die einzelnen Bereiche mit verschiedenen Farben dargestellt.



**Abbildung 2.8:** Benutzeroberfläche von *TelefonGUI* mit den beiden Registerkarten *suchen* und *neu*

In Swing werden die Fensterkomponenten in eine Art Container, so genannte Panels, gepackt. In der Abbildung 2.7 bezeichne ich den gelben Teil des gesamten Fensters, in dem gezeichnet werden kann (also ohne der Menüleiste) als Hauptpanel *mainP*. Dieses Hauptpanel ist wiederum unterteilt in das

- hellblaue Navigationspanel *navigationP*,
- dunkelblaue Personpanel *personP*,
- hellrote Suchenpanel *suchenP* in der Registerkarte *suchen*
- hellrote Neupanel *neuP* in der Registerkarte *neu*

Jeder Button erhält einen *ActionListener*, der das Anklicken überwacht und dann die entsprechenden Aktionen auslöst. Diese Aktionen werden später in den Methoden *ersteBActionPerformed()*, *letzteBActionPerformed()*, etc. implementiert, so dass die erste oder auch letzte Person im Telefonbuch aufgerufen werden kann

### Übung 2.11:

Hole vom Schulserver die Datei *TelefonGUI* und speichere diese als Projekt *Telefon2b*. Studiere den Quelltext der Klasse *Ansicht*. Suche die Komponenten, die im obigen Text beschrieben worden sind.

In vielen grafischen Oberflächen wird die Anordnung der Komponenten durch Angaben absoluter Koordinaten vorgenommen. Da jedoch Java-Programme auf vielen unterschiedlichen Plattformen laufen sollen, ist eine solche Vorgehensweise nicht sinnvoll. Zur Anordnung der Komponenten verwendet man in Java verschiedene so genannte *Layoutmanager*. Diese sorgen dafür, dass die Benutzeroberfläche auf allen Plattformen das gleiche Aussehen hat (weitestgehend).

Ich verwende in diesem Beispiel den *GridLayout-Manager* und den *BoxLayout-Manager*. Im *GridLayout* werden die Komponenten innerhalb eines rechteckigen Gitters angeordnet. Die Parameter beim Aufruf des Konstruktors bestimmen die vertikale und die horizontale Anzahl der Elemente. Im *BoxLayout* werden die Komponenten standardmäßig untereinander in der benötigten Größe angeordnet. Weitere Einzelheiten werden in den oben genannten Büchern beschrieben.

Die folgenden Übungen geben dir schrittweise eine kurze Einführung in die Programmierung von grafischen Oberflächen in Java. Zum Schluss wirst du eine Oberfläche programmiert haben wie in Abbildung 2.9 dargestellt. Verwende so oft wie möglich das Verfahren „Copy and Paste“!

### Übung 2.12:

- a) Ändere in der Methode *setTitle()* das Argument „Telefonbuch“. Ändere in der Methode *setBounds()* die Werte (50, 50, 400, 250) in (300, 200, 400, 250) bzw. in (50, 50, 600, 400). Formuliere schriftlich, welche Veränderungen dadurch stattfinden.  
Mache deine Veränderungen wieder rückgängig.
- b) Füge in das Navigationspanel *navigationP* zwei weitere Button *zurueck* und *vor* ein. Implementiere auch die zunächst noch leeren Methoden für die zugehörigen *ActionListener*.
- c) Füge in das Personpanel *personP* ein Label *Telefon:* und das zugehörige Textfeld ein.
- d) Füge in der Registerkarte *neu* die Label und Textfelder für Vorname und Telefon ein.

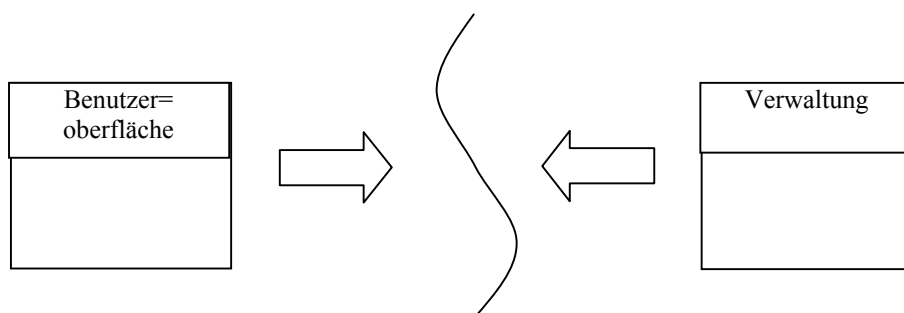
Nach Abschluss dieser Übungen sollte das Fenster des Projekts *Telefon2b* der Abbildung 2.9 gleichen:



**Abbildung 2.9:** Benutzeroberfläche von *Telefon2e* mit den beiden Registerkarten *suchen* und *neu*

### 2.3 Verbindung zwischen Telefonbuch und Ansicht

Unser Projekt, das ein einfaches Telefonbuch darstellt, ist also in zwei größere Teile zerlegt worden: Der erste Teil (die in Kapitel 2.2 erstellte Benutzeroberfläche) ermöglicht dem Benutzer, durch Eingabe in die Textfelder bzw. Drücken von Buttons nach Personen zu suchen bzw. weitere Personen einzugeben. Der zweite Teil (das in Kapitel 2.1 erstellte Verwaltungsverfahren) implementiert die Verwaltung des Telefonbuchs.



**Abbildung 2.10:** Festlegung der Schnittstelle

Die Abbildung 2.10 zeigt, dass für beide Teile klar definiert sein muss, wie sie den jeweils anderen Teil benutzen können – die Schnittstelle zwischen ihnen muss definiert werden. In größeren Projekten muss es also klare Richtlinien geben, welches Entwicklungsteam für welche Aufgaben zuständig ist und wie die verschiedenen Teile schließlich in der Gesamtanwendung zusammenspielen sollen. Es wird also notwendig sein, die Schnittstelle festzulegen, bevor an der Implementierung der Teile gearbeitet werden kann.

In dieser Einführung des Projekts habe ich aus didaktischen Gründen den umgekehrten Weg beschrieben

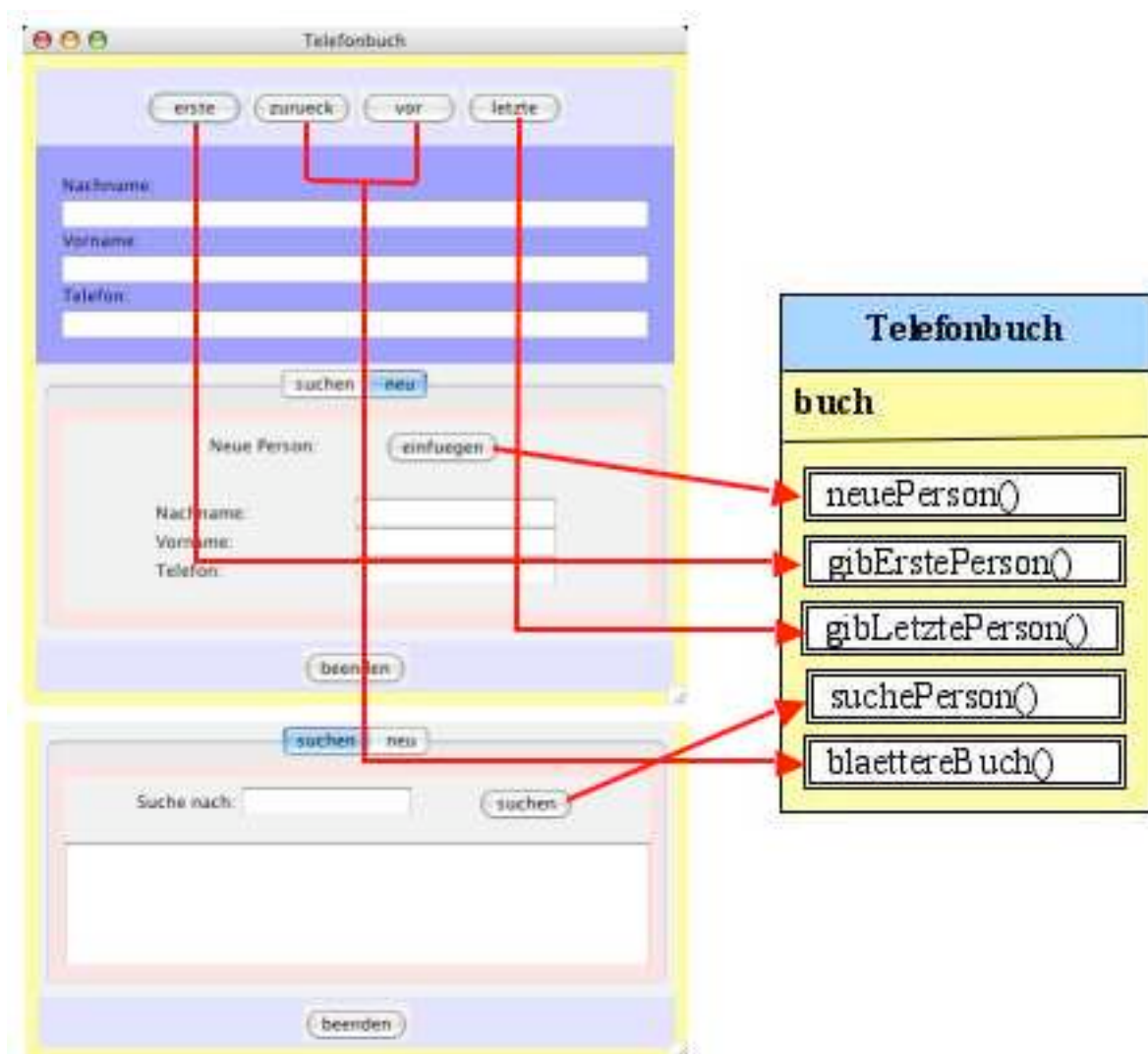


Abbildung 2.11: Schnittstelle zwischen GUI und Telefonbuch

Du wirst nun die Benutzeroberfläche mit der Verwaltung des Telefonbuchs verbinden. Abbildung 2.11 zeigt, wie die Schnittstelle zwischen diesen beiden Klassen definiert wird.

```
private Telefonbuch telefonbuch;

public Ansicht()
{
    erzeugeFenster();
    telefonbuch = new Telefonbuch();
}
```

**Abbildung 2.12:** Konstruktor der Klasse *Ansicht* erzeugt das Objekt *Telefonbuch*

Dazu muss natürlich zuerst ein Objekt der Klasse *Telefonbuch* im Konstruktor erzeugt werden.

### Übung 2.13:

Hole mit dem Menüpunkt „Edit > Add Class from File...“ die beiden Klassen *Telefonbuch* und *Person* aus *Telefon2a* in dein Projekt *Telefon2b*.

Ergänze den Konstruktor in der Klasse *Ansicht*.

### 2.3.1 Aufrufen des ersten bzw. letzten Eintrags

Zum Aufruf des ersten Eintrags stellt die Klasse *Telefonbuch* bereits die Methode *gibErstePerson()* zur Verfügung.

```
private void ersteBActionPerformed(ActionEvent e)
{
    Person person = telefonbuch.gibErstePerson();
    eintrageTextfelderPersonP(person);
}
```

**Abbildung 2.13:** Die Methode *ersteBActionPerformed()* in der Klasse *Ansicht*

Diese Methode liefert die erste Person, deren Daten in die entsprechenden Textfelder des Personpanels *personP* noch eingetragen werden müssen. Dazu wird in der Klasse *Ansicht* die Methode *eintrageTextfelderPersonP()* implementiert.

```
private void eintrageTextfelderPersonP(Person person)
{
    nachnameTF.setText(person.gibNachname());
}
```

```
    vornameTF.setText(person.gibVorname());
    telefonTF.setText(person.gibTelefon());
}
```

**Abbildung 2.14:** Die Methode *eintrageTextfelderPersonP()* füllt die Textfelder in *personP*

### Übung 2.14:

Implementiere die beiden Methoden *ersteBActionPerformed()* und *eintrageTextfelderPersonP()*. Vereinbare die letztere Methode als *private*. Kontrolliere die Funktion des Buttons *erste*.

Erstelle nun eine Methode, die den letzten Eintrag im Telefonbuch in die Textfelder von *personP* zeigt. Kontrolliere die Funktion des Buttons *letzte*.

## 2.3.2 Das Telefonbuch durchblättern

Zum Durchblättern stellt die Klasse *Telefonbuch* bereits die Methode *blaetterebuch()* zur Verfügung:

```
private void vorBActionPerformed(ActionEvent e)
{
    boolean vorwaerts = true; //also Telefonbuch vorwaerts durchlaufen
    String nachname = nachnameTF.getText();
    String vorname = vornameTF.getText();
    Person person = telefonbuch.blaettereBuch(nachname,
                                                vorname, vorwaerts);
    eintrageTextfelderPersonP(person);
}
```

**Abbildung 2.15:** Die Methode *vorBActionPerformed()* in der Klasse *Ansicht*

Die Methode *blaettereBuch()* erwartet die drei Parameter *nachname*, *vorname* und *vorwaerts*. Der Nachname und der Vorname werden aus den Textfeldern von *personP* gelesen. Um vorwärts zu blättern, wird *vorwaerts* auf den Wert *true* gesetzt. So kann nun von der Klasse *Telefonbuch* die nachfolgende Person geliefert werden, deren Daten in die Textfelder von *personP* eingetragen werden.

### Übung 2.15:

Implementiere die Methode *vorBActionPerformed()*. Kontrolliere die Funktion des Buttons *vor*.

Erstelle nun eine Methode, mit der der Anwender im Telefonbuch zurück blättern kann.

### 2.3.3 Einfügen einer neuen Person

Zum Einfügen einer neuen Person stellt die Klasse *Telefonbuch* bereits die Methode *neuePerson()* zur Verfügung:

```
private void einfuegenBActionPerformed(ActionEvent e)
{
    //holt sich die Eintraege aus den Textfeldern
    String nachname = neuNachnameTF.getText().trim();
    String vorname = neuVornameTF.getText().trim();
    String telefon = neuTelefonTF.getText().trim();
    //erzeugt eine neue Person mit diesen Daten
    Person person = new Person(nachname, vorname, telefon);
    //fuegt dies Person in das Telefonbuch
    telefonbuch.neuePerson(person);
    //fuellt die Textfelder in personP
    eintrageTextfelderPersonP(person);
    //leert die Textfelder in neuP fuer einen weiteren Eintrag
    leereTextfelderNeuP();
}
```

**Abbildung 2.16:** Die Methode *einfuegenBActionPerformed()* in der Klasse *Ansicht*

Zuerst werden die Daten der Person aus den Textfeldern von *neuP* herausgelesen. Die Methode *trim()* entfernt eventuelle Leerzeichen, die der Benutzer aus Versehen getippt hat und nicht in das Telefonbuch übernommen werden sollen. Nun wird eine neue Person erzeugt, in das Telefonbuch eingefügt und in die Daten in die entsprechenden Textfelder von *personP* eingelesen. Zum Schluss leert die Methode *leereTextfelderNeuP()* die Textfelder in *neuP*, so dass der Anwender nun wieder eine neue Personen eintragen kann.

#### Übung 2.16:

Implementiere die Methode *einfuegenBActionPerformed()*. Programmiere auch eine private Methode *leereTextfelderneuP()*, die alle Textfelder im Panel *neuP* leert.

Kontrolliere die Funktion des Buttons *einfuegen*.

### 2.3.4 Suche nach bestimmten Personen

Für die Suche nach bestimmten Personen stellt die Klasse *Telefonbuch* bereits die Methode *suchePerson()* zur Verfügung:

```
private void suchenBActionPerformed(ActionEvent e)
{
    //entfernt Eintraege in suchAusgabeTA
    leereTextareaSuchenP();
    //holt sich den Eintrag aus demTextfeld
    String eingabe = suchEingabeTF.getText().trim();
    //erhaelt einen String mit Daten aller Personen
    String eintrag = telefonbuch.suchePerson(eingabe);
    suchAusgabeTA.append(eintrag);
}
```

**Abbildung 2.17:** Die Methode *suchenBActionPerformed()* in der Klasse *Ansicht*

Zuerst werden eventuelle Einträge im Textarea des Panels *suchenP* entfernt. Anschließend wird der gesuchte Name aus dem Textfeld gelesen und im Telefonbuch nach allen Personen mit diesem Namen gesucht. Die Daten dieser gefundenen Personen werden zum Schluss in das Textarea eingetragen.

#### Übung 2.17:

Implementiere die Methode *suchenBActionPerformed()*. Programmiere auch eine private Methode *leereTextareaSuchenP()*, die das Textarea im Panel *suchenP* leert.

Kontrolliere die Funktion des Buttons *suchen*.

## 2.4 Unzulänglichkeiten dieses Modell

Dieser Entwurf des Telefonbuchs weist noch viele benutzerunfreundliche Eigenschaften auf. Einige werde ich in den folgenden Übungen aufzeigen

#### Übung 2.18:

- Suche nach einem Namen, der im Telefonbuch nicht existiert. Hier sollte dem Benutzer eine entsprechende Meldung gegeben werden.
- Ändere die Telefonnummer einer Person. Dem Benutzer sollte man eine Möglichkeit geben, falsche Eingaben zu ändern.
- Lasse in der Registerkarte *suchen* die Textfelder leer und drücke auf den Button *einfüegen*. Blättere anschließend durch das Telefonbuch.

- d) Lösche eine Person aus dem Telefonbuch.
- e) Der Benutzer sollte die Möglichkeit haben, auch durch Betätigen der Return-Taste Aktionen auszulösen.
- f) .....