

3 Speichern, Öffnen, Drucken

Ein Projekt wie das beschriebene Telefonbuch ist sinnlos, wenn die Personendaten nicht in eine Datei z. B. auf die Festplatte gespeichert werden können. Ein Anwender will diese Datei nach dem Neustart deines Programms wieder von der Festplatte öffnen und eventuell die Personendaten auch ausdrucken können.

In diesem Kapitel werde ich deshalb kurz auf diese Möglichkeiten eingehen. Ich möchte jedoch nicht jede Einzelheit erklären, sondern nur plausibel machen, wie die entsprechenden Methoden zum Speichern, Öffnen und Ausdrucken funktionieren. Weitere Einzelheiten dazu solltest du deshalb in den vorgeschlagenen Büchern nachlesen.

Zunächst wird die GUI erweitert durch die Buttons *speichernB*, *oeffnenB* und *druckenB*.

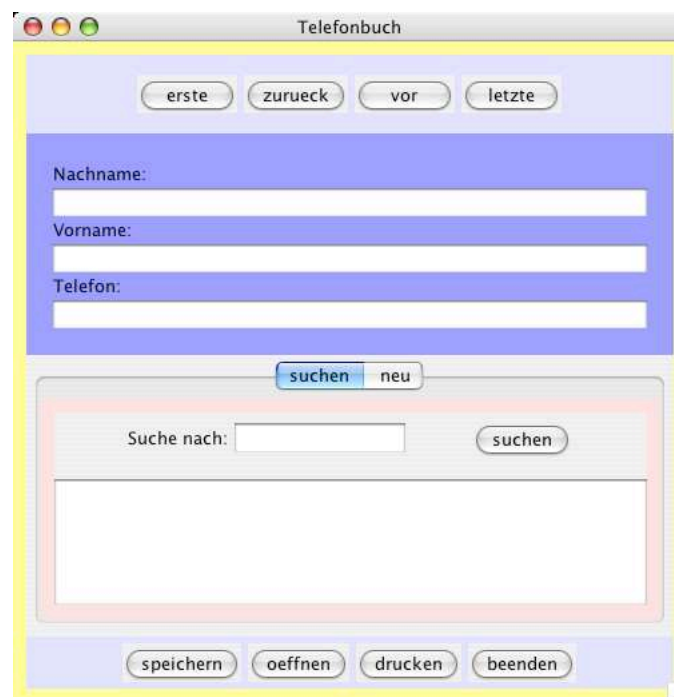


Abbildung 3.1: Erweiterung der grafischen Benutzeroberfläche

Übung 3.1:

Speichere das Projekt *Telefon2bunter* dem Namen *Telefon3a*. Erweitere nun die Oberfläche wie in Abbildung 3.1 dargestellt. Implementiere auch die zugehörigen noch leeren Aktionsmethoden.

3.1 *Writer und Reader*

Die einzelnen Personendaten *nachname*, *vorname* und *telefon* werden in Textdateien gespeichert. In Java werden Klassen, die mit solchen Textdateien umgehen können *Writer* und *Reader* genannt und sind im Paket *java.io* enthalten.

3.1.1 Ausgabe von Text mit *FileWriter*

Um nun die Personendaten in eine Datei zu speichern, muss du diese Datei öffnen, in diese die Daten schreiben und zuletzt die Datei wieder schließen.

Abbildung 3.2 zeigt das Grundmuster dieser drei Schritte:

```
try {
    FileWriter ziel = new FileWriter(file);
    .....
    ziel.write(beschreibung);
    .....
    ziel.close();
}
catch(IOException ioe) {
    System.out.println("Daten nicht gespeichert!");
}
```

Abbildung 3.2: Grundmuster zum Speichern in eine Textdatei

Zuerst wird ein Objekt der Klasse *FileWriter* erzeugt, das als Parameter den Namen der Zielfile erhält. Gleichzeitig wird diese Zielfile geöffnet und auf den Empfang von Ausgabedaten vorbereitet. Nun kann die Methode *write()* die Daten in Form von Strings in die Zielfile schreiben. Zuletzt muss die Zielfile wieder explizit geschlossen werden. Damit ist der Speichervorgang beendet. Falls einer dieser drei Schritte aus irgendwelchen Gründen fehlschlägt, wird eine Exception ausgelöst, die entsprechend behandelt wird.

3.1.2 Einlesen von Text mit *FileReader*

Das Einlesen von Daten aus einer Textdatei erfolgt mit einem *FileReader*. Auch hier muss zuerst die Datei geöffnet, die Daten eingelesen und die Datei wieder geschlossen werden. In der Regel liegen die gespeicherten Daten in der Textdatei in mehreren Zeilen vor. Die Klasse *FileReader* bietet allerdings keine Methode zum Lesen einer Zeile. Deswegen werden die Daten in einem so genannten *BufferedReader* zwischengespeichert, der eine Methode *readLine()* anbietet.

Abbildung 3.3 zeigt das Grundmuster.

```
try {
    BufferedReader quelle = new BufferedReader(
        new FileReader(file));
    String zeile = quelle.readLine();
    while(zeile != null) {
        ..... tue was mit dieser Zeile .....
        zeile = quelle.readLine()
    }
    quelle.close();
}
catch(IOException ioe) {
    System.out.println("Daten nicht geoeffnet!");
}
```

Abbildung 3.3: Grundmuster zum Öffnen einer Textdatei

3.2 Speichern in eine Datei

Die Klasse *FileWriter* ermöglicht mit seinen Methoden das Speichern von Dateien. Im *Handbuch der Java-Programmierung* stellt Guido Krüger ein kleines Beispielprogramm „*Listing1801.java*“ auf S. 408f im Kapitel „18.2 Ausgabe-Streams“ vor.

Übung 3.2:

Im Computerraum wirst du das o. g. Buch finden. Ansonsten frage deinen Informatik-Lehrer. Suche das entsprechende Beispiel, implementiere es in **BlueJ** und teste es. Die gespeicherte Textdatei findest du nun im entsprechenden Projektordner. Öffne diese mit einem beliebigen Texteditor.

Versuche das Listing soweit wie möglich zu verstehen.

Ein Nachteil dieses Programms ist wohl, dass der Benutzer nicht selbstständig den Ort wählen kann, wo er die Datei ablegt. Hier bietet Java-Swing die Klasse *JFileChooser*, die den Standard-Sichern-Dialog öffnet. Der Benutzer kann mit dessen Hilfe den Ort wählen und den Namen der Textdatei eingeben.

Bemerkung:

Im Betriebssystem *Windows XP* heißt dieses Dialogfenster *Speichern*.

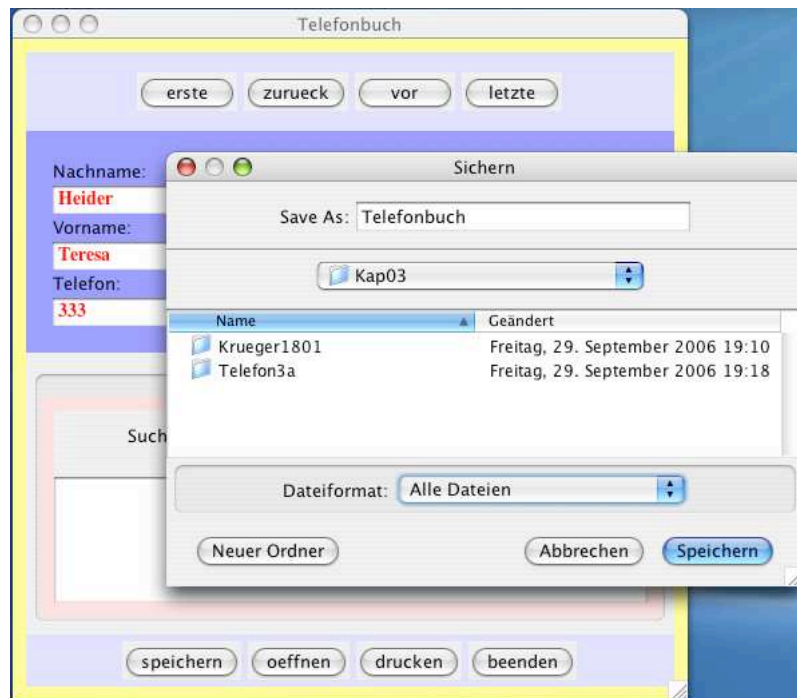


Abbildung 3.4: Aufruf des Standard-Sichern-Dialogs

Abbildung 3.4 zeigt den Standard-Sichern-Dialog. In diesem Dialogfenster wählt der Benutzer den Namen *Telefonbuch* für die Textdatei mit den Personendaten und legt diese in den Ordner *Kap03* ab.

Weitere Einzelheiten kannst du in *The JFC Swing Tutorial, Second Edition* von Kathy Walrath, u. a. nachlesen. Im Kapitel 7 *How To Use FileChooser* auf S. 206ff wird beschrieben, wie dieser verwendet wird.

Zuerst einmal musst du die Klasse *Ansicht* für die Dateneingabe, Datenausgabe und für den Sichern-Dialog vorbereiten. Abbildung 3.3 zeigt, welche Pakete zusätzlich noch importiert werden müssen. Weiterhin musst du die beiden Objekte *fc* und *file* deklarieren und im Konstruktor das Objekt *fc* erzeugen.

```
import java.io.*;
import javax.swing.filechooser.*;

//zum Speichern und Oeffnen
private JFileChooser fc;
private File file;

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

```
    fc = new JFileChooser();  
}
```

Abbildung 3.5: Erweiterungen der Klasse *Ansicht* bei den *import*-Anweisungen, bei den Datenfeldern und im Konstruktor

Nun hast du alles für das Speichern der Personendaten vorbereitet. Klickst du auf den Button *speichern*, so sollte sich der Standard-Sichern-Dialog öffnen. Den zugehörigen Quelltext in der Methode *speichernBActionPerformed()* zeigt die Abbildung 3.4. Sie ruft zuerst den Standard-Sichern-Dialog auf. In der *if*-Anweisung wird nachgefragt, ob der Anwender in diesem Dialog einen Namen derjenigen Textdatei eingetragen hat, in die er die Daten speichern will. Diese Datei wird nun innerhalb der *try*-Anweisung erzeugt oder geöffnet. Die Klasse *Telefonbuch* stellt nun mit der Methode *telefonbuch.zeigeAllePersonen()* einen String mit allen Personendaten zur Verfügung, der als *beschreibung* in die Textdatei eingeschrieben wird. Zum Schluss wird die Textdatei wieder geschlossen.

```
/** Reagiert auf Druecken von speichernB. */  
private void speichernBActionPerformed(ActionEvent e)  
{  
    //verwaltet den Standard-Sichern-Dialog  
    int returnVal = fc.showSaveDialog(Ansicht.this);  
    if (returnVal == JFileChooser.APPROVE_OPTION) {  
        file = fc.getSelectedFile();  
    }  
    //versucht die Daten in die Textdatei zu schreiben  
    try {  
        FileWriter ziel = new FileWriter(file);  
        String beschreibung = telefonbuch.zeigeAllePersonen();  
        ziel.write(beschreibung);  
        ziel.close();  
    }  
    catch(IOException ioe) {  
        System.out.println("Daten nicht gespeichert!");  
    }  
}
```

Abbildung 3.6: Die Methode *speichernBActionPerformed()* in der Klasse *Ansicht*

Übung 3.3:

Implementiere in deinem Projekt *Telefon3a* den in Abbildung 3.5 und 3.6 dargestellten Quelltext an den entsprechenden Stellen der Klasse *Ansicht*.

a) Überprüfe, ob der Standard-Sichern-Dialog aus Abbildung 3.2 gezeigt

wird.

- b) Speichere die Datei unter dem Namen *Telefonbuch*.
- c) Suche die Datei *Telefonbuch* in dem entsprechenden Ordner und öffne diese mit einem beliebigen Texteditor. Vergleiche dazu Abbildung 3.5.



Abbildung 3.7: Die Textdatei *Telefonbuch*

3.3 Zeichenketten zerlegen

Du kannst nun alle Personendaten, die im *TreeSet buch* verwaltet werden, in eine Textdatei *Telefonbuch* speichern. Nach einem erneuten Einschalten deines Computers sollten die gesamten Personendaten aus dieser Textdatei wieder in den *TreeSet buch* eingelesen werden. Somit ist es anschließend möglich, alle Personen in den entsprechenden Textfeldern der GUI wieder darzustellen.

Das Einlesen der Daten aus der Textdatei *Telefonbuch* in die entsprechenden Textfelder unserer GUI gestaltet sich jedoch etwas kompliziert. Wie Abbildung 3.7 zeigt, wird pro Person eine Zeile geschrieben und diese Zeile besteht jeweils aus Nachname, Vorname und Telefon.

Du lernst nun ein Verfahren kennen, das eine solche Zeichenkette zuerst in einzelne Zeilen zerlegt und anschließend jede solche Zeile wiederum in ihre einzelnen Worte zerlegt.

Die Klasse *String* bietet die Methode *split()*. Sie kann einen String in getrennte Substrings zerlegen und diese in einem *String*-Array zurückliefern. Der Parameter, der der Methode *split()* übergeben wird, definiert, an welcher Art von Zeichen der Originalstring aufgeteilt werden soll.

```
import java.util.*;

public class Zerlegen
{
    String zeile = "Heider Teresa 333";
```

```
public void drucken()
{
    String[] daten = zeile.split(" ");
    //verdeutlicht nur den Stringarray daten
    String nachname = daten[0];
    String vorname = daten[1];
    String telefon = daten[2];
    //zeigt die Personendaten im Terminalfenster.
    System.out.println(zeile + "\n");
    System.out.println(nachname);
    System.out.println(vorname);
    System.out.println(telefon);
}
}
```

Abbildung 3.8: Quelltext des Projekts *Zerlegung*

Die Methode *drucken()* teilt den String *Heider Teresa 333* in seine einzelnen Worte. Hierzu übergibt man der Methode *split()* als Parameter das Leerzeichen. Diese erzeugt automatisch einen *String*-Array der korrekten Größe, aus dem anschließend die einzelnen Worte ermittelt werden können. Nun können diese im Terminalfenster angezeigt werden.



Abbildung 3.9: Das Terminalfenster zeigt die einzelnen Personendaten

Übung 3.4:

Hole vom Schulserver das Projekt *Zerlegung1*. Studiere den Quelltext der Methode *drucken()*. Suche die Komponenten, die im obigen Text beschrieben worden sind.

Füge zu den bestehenden Personendaten auch Straße, PLZ und Wohnort hinzu.

Abbildung 3.7 zeigt allerdings, dass die gesamten Personendaten in mehreren Zeilen vorliegen. Du müsstest also den gesamten String zuerst in seine fünf Zeilen zerlegen und dann anschließend jede Zeile in seine drei Worte. Auf

welche Weise dies programmiert werden kann, lernst du in den folgenden Kapiteln.

3.4 Öffnen einer Datei

Die Klasse *FileReader* ermöglicht mit seinen Methoden das Lesen von Dateien. Im *Handbuch der Java-Programmierung* stellt Guido Krüger ein kleines Beispielprogramm „*Listing1808.java*“ auf S. 423f im Kapitel „18.3 Eingabe-Streams“ vor.

Übung 3.5:

Im Computerraum wirst du das o. g. Buch finden. Ansonsten frage deinen Informatik-Lehrer. Suche das entsprechende Beispiel, implementiere es in **BlueJ** und teste es.

- a) Erstelle eine beliebige Textdatei mit einem Texteditor. Merke dir den Namen und den Pfad. Öffne diese Datei mit Hilfe des Programms.
- b) Versuche die Datei aus Übung 3.3 oder 3.5 zu finden und zu öffnen

Ein Nachteil dieses Programms ist wohl, dass der Benutzer nicht selbstständig den Ort wählen kann, wo die gewünschte Datei liegt. Hier bietet Java-Swing die Klasse *JFileChooser*, die den Standard-Öffnen-Dialog öffnet. Der Benutzer kann mit dessen Hilfe den Ort der gewünschten Datei wählen.

Nun ist alles für das Lesen der Personendaten vorbereitet. Abbildung 3.11 zeigt den Quelltext der Methode *oeffnenBActionPerformed()*. Sie ruft zuerst den Standard-Öffnen-Dialog auf. In der *if*-Anweisung wird nachgefragt, ob der Anwender in diesem Dialog einen Namen derjenigen Textdatei eingetragen hat, die er öffnen will. Diese Datei wird nun in der *try*-Anweisung geöffnet. Anschließend wird jeweils eine komplette Textzeile eingelesen und mit Hilfe der Methode *split()* in seine drei Bestandteile zerlegt. Nun kann damit eine neue Person erzeugt werden und diese in das Telefonbuch eingefügt werden. Existiert in der Textdatei keine weitere Textzeile, wird die Textdatei wieder geschlossen. Zum Schluss soll noch die erste Person in den Textfeldern gezeigt werden.

```
import java.util.*;

/** Reagiert auf Druecken von oeffnenB. */
private void oeffnenBActionPerformed(ActionEvent e)
{
    //verwaltet den Standard-Sichern-Dialog
    int returnVal = fc.showOpenDialog(Ansicht.this);
```

```
if (returnVal == JFileChooser.APPROVE_OPTION) {
    file = fc.getSelectedFile();
}
//versucht die Daten aus der Textdatei zu lesen
try {
    BufferedReader quelle = new BufferedReader(
        new FileReader(file));

    String zeile = quelle.readLine();
    //Liest zeilenweise aus der gespeicherten Textdatei.
    while (zeile != null) {
        //zerlegt die Zeile in einzelne Woerter
        String[] daten = zeile.split(" ");
        //verdeutlicht nur den Stringarray daten
        String nachname = daten[0];
        String vorname = daten[1];
        String telefon = daten[2];
        Person person = new Person(nachname, vorname, telefon);
        telefonbuch.neuePerson(person);
        //holt naechste zeile
        zeile = quelle.readLine();
    }
    quelle.close();
}
catch(IOException ioe) {
    System.out.println("Daten nicht geoeffnet!");
}
//zeigt ersten Eintrag aus dem Telefonbuch
Person person = telefonbuch.gibErstePerson();
eintrageTextfelderPersonP(person);
}
```

Abbildung 3.10: Die Methode *öffnenBActionPerformed()* in der Klasse *Ansicht*

Übung 3.6:

Speichere das Projekt *Telefon3a* unter dem Namen *Telefon3b*.

- Da du nun die Daten aller Personen in der Datei *Telefonbuch* bereits gespeichert hast, darf im Konstruktor der Klasse *Telefonbuch* nur noch die Anweisung `buch = new TreeSet<Person>();` stehen. Entferne alle weiteren Anweisungen, die die Testdaten betreffen!!
- Implementiere die Methode *oeffnenBActionPerformed()* in der Klasse *Ansicht*.
- Überprüfe, ob der Standard-Öffnen-Dialog aus Abbildung 3.11 gezeigt wird.

- d) Öffne die Datei *Telefonbuch*. Anschließend sollten die Personendaten nun in den entsprechenden Textfeldern der GUI stehen.

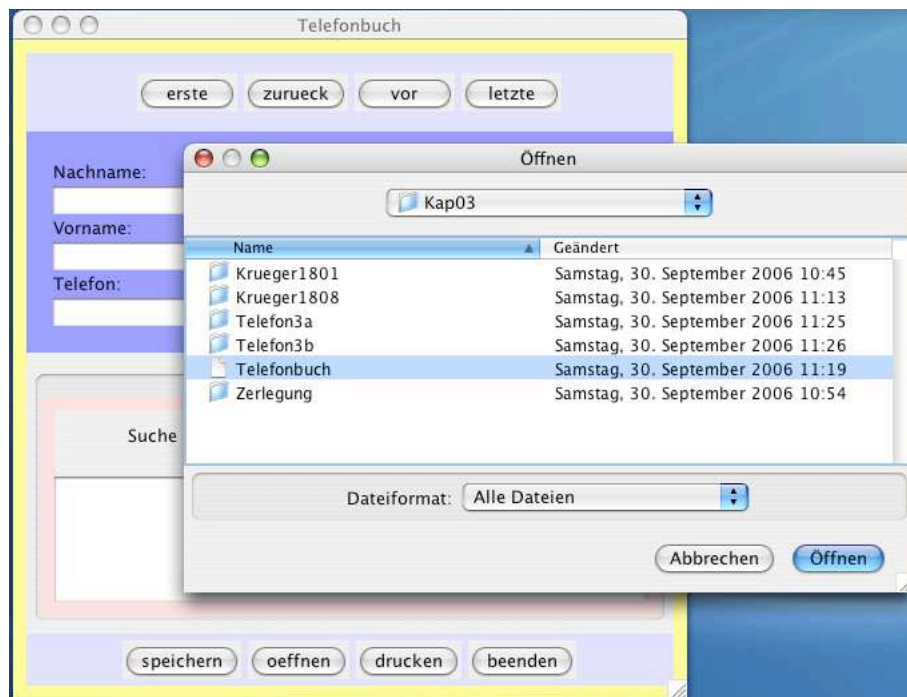


Abbildung 3.11: Aufruf des Standard-Öffnen-Dialogs

Abbildung 3.11 zeigt den Standard-Öffnen-Dialog. Der Benutzer wählt hier gerade die Datei *Telefonbuch* aus dem Ordner *Kap03*.

Übung 3.7:

Füge nun weitere Personen in dein Telefonbuch ein. Speichere dein Telefonbuch. Du kannst nun die Anwendung beenden oder das Projekt schließen und wieder neu starten. Deine Personendaten hast Du auf der Festplatte gespeichert.

3.5 Drucken

Alle Klassen, die die Druckausgabe betreffen, befinden sich im Paket *java.awt.print.**, das in die Klasse *Ansicht* importiert werden muss.

Das Drucken von Daten wird vom System gesteuert ähnlich wie die Ausgabe auf dem Bildschirm. Der Programmierer muss nur dafür sorgen, dass die zu druckenden Daten an das System angemeldet werden. Im Prinzip muss Folgendes der Reihe nach erledigt werden:

1. Zunächst muss ein *PrinterJob* geholt werden mit Hilfe der Methode *getPrinterJob()*. Dieser *PrinterJob* speichert die Informationen, die für das Drucken erforderlich sind. Darüber hinaus sorgt er für die entsprechenden Dialoge „Seite einrichten ...“ und „Drucken ...“ und dem eigentlichen Starten des Drucks.
2. Dem *PrinterJob*-Objekt muss nun mitgeteilt werden, wo sich die auszudruckenden Daten befinden. Hierzu werden die Methoden *setPrintable()* (bzw. *setPageable()*) verwendet.
3. Falls erforderlich werden nun die Dialoge zur Einrichtung der Druckseite und zur Druckerauswahl angezeigt. Hierzu werden die Methoden *printDialog()* (bzw. *pageDialog()*) verwendet.
4. Jetzt ist die Anwendung soweit, dass der Druckjob angestoßen werden kann. Hierzu wird die Methode *print()* verwendet.

Diese vier Schritte werden in der Methode *druckenBActionPerformed()* erledigt.

```
public void druckenBActionPerformed(ActionEvent e)
{
    PrinterJob pj = PrinterJob.getPrinterJob();
    pj.setPrintable(this);
    if (pj.printDialog()) {
        try {
            pj.print();
        }
        catch (Exception pe) {
            pe.printStackTrace();
        }
    }
}
```

Abbildung 3.12: Die Methode *druckenBActionPerformed()*

Zuerst wird also das *PrinterJob*-Objekt *pj* mit der Methode *getPrinterJob()* geholt. Durch die Methode *setPrintable()* in Verbindung mit *this* wird dem Druckjob gesagt, was ausgedruckt wird bzw. wo sich die Druckdaten befinden.

In der *if*-Anweisung wird nachgefragt, ob der OK-Button im folgenden plattformsspezifischen Druckdialog gedrückt wurde. Die Methode *printDialog()* liefert einen booleschen Rückgabewert, der anzeigt, ob der Dialog mit „OK“ beendet oder abgebrochen wurde.

Leider kann man nicht immer davon ausgehen, dass der Ausdruck sofort beginnt. Es könnte Papier fehlen, die Patrone leer sein, usw. Aus diesem Grunde

wird innerhalb des *try-catch*-Blocks zunächst einmal versucht, den Druck zu starten. Ist alles in Ordnung, wird die Methode *print()* aufgerufen, die für die Druckausgabe sorgt ähnlich wie die Methode *paint()* für die Bildschirmausgabe. Andernfalls wird eine Exception ausgelöst, die im nachfolgenden Exceptionhandler ausgewertet werden kann.

```
public int print(Graphics g, PageFormat pf, int pi) throws PrinterException
{
    String ausgabe1 = "Hallo, ich erkläre dir die Grundlagen des
                      Ausdrucks.";
    String ausgabe2 = "Dein Informatik-Lehrer";

    if (pi >= 1) {
        return Printable.NO_SUCH_PAGE;
    }
    g.setFont(new Font("Serif", Font.BOLD, 15));
    g.drawString(ausgabe1, 70, 150);
    g.drawString(ausgabe2, 190, 190);
    return Printable.PAGE_EXISTS;
}
```

Abbildung 3.13: Die Methode *print()* in der Klasse *Ansicht*

Analog wie bei der Methode *paint()* wird auch bei der Methode *print()* das übergebene Graphics-Objekt verwendet. Weiterhin wird ein konfigurierbares *PageFormat*-Objekt *pf* übergeben, das dazu benutzt wird, die Größe des Papiers und dessen bedruckbaren Bereich zu ermitteln (wird in diesem Beispiel allerdings nicht verwendet). Der Rückgabewert von *print()*, nämlich der Parameter *int pi* (*pageIndex*), gibt an, ob die Seitenzahl gültig war und die Druckausgabe erzeugt werden konnte, oder ob eine ungültige Seitenzahl übergeben wurde. In unserem Beispiel handelt es sich um einen *printable job*, also um einen Druckjob, der nur eine Seite enthält. Dieser Seitenindex beginnt mit 0 und darf deshalb bei einem einseitigen Dokument nicht größer oder gleich 1 sein. In diesem Fall würde die Methode *print()* die vordefinierte Konstante *NO_SUCH_PAGE* zurückgeben. Hat der Seitenindex den Wert 0, so kann die Methode *g.drawString()* aufgerufen und ausgeführt werden. Daher kann anschließend der Wert *PAGE_EXISTS* zurückgeliefert werden.

Übung 3.8:

Hole vom Schulserver das Projekt *Drucken* und speichere es unter dem Namen *Drucken1*.

- a) Studiere den Quelltext. Versuche die oben beschriebenen Anweisungen zu finden. Rufe die Methode *druckenBActionPerformed()* auf. Da du in dieser

Übung keine Benutzeroberfläche verwendest, wird der Parameter *ActionEvent e* nicht benötigt.

- b) Lagere die beiden Anweisungen *g.drawString(.....)* aus in eine separate Methode *druckeText()*.
- c) Versuche den Ausgabertext 10-mal untereinander zu schreiben. Verwende dazu eine Schleife.

Im Projekt *Drucken1* liegen die Daten, die ausgedruckt werden sollen, als String *ausgabe1* bzw. *ausgabe2* vor. Im deinem Projekt *Telefonbuch* werden die Personendaten auf zwei verschiedene Arten gespeichert:

1. als Textdatei *telefonbuch* irgendwo auf der Festplatte,
2. im TreeSet *buch* als Objekte der Klasse *Person*.

Übung 3.9:

Überlege dir Vor- und Nachteile, beim Ausdruck die Personendaten aus der Textdatei *telefonbuch* oder aus den TreeSet *buch* zu verwenden.

Ich habe mich entschieden, im folgenden Projekt *Zerlegung2* die Personendaten aus dem TreeSet *buch* zu verwenden. Hier liegt immer die aktuelle Version vor, somit muss der Benutzer vor dem Ausdrucken nicht diese neueste Version zuerst speichern.

In der folgenden Übung lernst du, wie alle Personendaten, die als String *beschreibung* vorliegen, zur Druckausgabe vorbereitet werden.

Heider	Teresa	333
Riedel	Franziska	444
Schurian	Alexander	222
Schurian	Isabella	555
Straka	Tobias	111

Abbildung 3.14: Der String *beschreibung*, der alle Personendaten enthält

Abbildung 3.14 zeigt, dass der String *beschreibung* für jede Person eine neue Zeile anlegt. Jede solche Zeile enthält als Worte die entsprechenden Attributwerte dieser Person. Mit Hilfe der Methode *split()* kannst du nun diesen String zuerst in seine Zeilen, jede Zeile anschließend in seine Worte zerlegen. Hierzu verwendest du dieselben Techniken wie im Projekt *Zerlegung1*.

```
/** Schreibt die gespeicherten Daten in Tabellenform. */  
private void druckeTelefonbuch(Graphics g)
```

```
{
    int neueZeile = 0; //naechste Zeile beim Ausdrucken

    g.setColor(Color.blue);
    g.setFont(new Font("Serif", Font.PLAIN, 14));

    String beschreibung = zeigeAllePersonen();
    String[] zeile = beschreibung.split("\n");//zerlegt in Zeilen
    for (int i = 0; i < zeile.length; i++) {
        int neuesWort = 0; //naechstes Wort beim Ausdruck
        String[] daten = zeile[i].split(" ");//zerlegt in Woerter
        for (int n = 0; n < daten.length; n++) {
            g.drawString(daten[n], 100 + neuesWort*100,
                150 + neueZeile*20);

            neuesWort++;
        }
        neueZeile++;
    }
}
```

Abbildung 3.15: Quelltext der Methode *druckeTelefonbuch()*

Der Quelltext der Methode *druckeTelefonbuch()* in Abbildung 3.15 zeigt, dass der String *beschreibung* mit Hilfe des Trennzeichens "\n" in seine Zeilen zerlegt wird und diese Zeilen in *zeile[i]* verwaltet werden. Anschließend wird jede solche Zeile mit Hilfe des Trennzeichens " " in ihre Wörter zerlegt und diese Wörter werden in *daten[i]* verwaltet. Nun kann die Methode *drawString()* jedes dieser Wörter an der entsprechenden Position schreiben bzw. hier ausdrucken.

Übung 3.9:

Hole vom Schulserver das Projekt *Zerlegung2*. Studiere den Quelltext der Methode *druckenBActionPerformed()*. Suche die Komponenten, die im obigen Text beschrieben worden sind.

Füge zu den bestehenden Personendaten nun einen Wohnort hinzu und lasse alle Daten ausdrucken. (vgl. Projekt *Zerlegung3*)

Heider	Teresa	333	EEE
Fiedel	Franziska	444	BBB
Schurian	Alexander	222	CCC
Schurian	Isabella	555	DDD
Straka	Tobias	111	AAA

Abbildung 3.16: Ausdruck, den die Methode *druckenBActionPerformed()* liefert

Im Prinzip hast du nun den Druckvorgang verstanden. Du kannst nun wieder zum Projekt *Telefon* übergehen.

Übung 3.10:

Speichere das Projekt *Telefon3b* unter dem Namen *Telefon3c*.

Implementiere die im Projekt *Zerlegung2* beschriebenen Methoden *druckenBActionPerformed()*, *print()* und *druckeTelefonbuch()* in die Klasse *Ansicht*.

Vergiss nicht die beiden Änderungen:

```
import java.awt.print.*;  
public class Ansicht extends JFrame implements Printable
```

Übung 3.11:

Verändere die Methode *druckeTelefonbuch()*, so dass sie einen Ausdruck liefert wie in Abbildung 3.17 dargestellt.

Nachname	Vorname	Telefon
Heider	Teresa	333
Riedel	Franziska	444
Schurian	Alexander	222
Schurian	Isabella	555
Straka	Tobias	111

Abbildung 3.17: Ausdruck der gespeicherten Datei in einer Tabelle

Nun kannst du in Telefonbuch Personen eingeben, sie werden alphabetisch sortiert, du kannst den gesamten Datenbestand speichern, ihn jederzeit ergänzen und ausdrucken lassen. Ich habe nicht jede Einzelheit erklärt, sondern nur versucht, plausibel machen, wie die entsprechenden Methoden zum Speichern, Öffnen und Ausdrucken funktionieren. Weitere wertvolle Hilfen findest du im „*Handbuch der Java-Programmierung*“ von Guido Krüger und im „*The JFC Swing Tutorial*“ von Katy Walrath u. a.

Natürlich weist dieser Entwurf noch viele Unzulänglichkeiten auf. Die Mängelliste aus Kapitel 2 lässt sich noch beliebig ergänzen auf.

Übung 3.12:

- a) Auf dem Ausdruck haben Daten von etwa 30 Personen Platz. Was passiert mit den weiteren Persondaten.
- b) Beim Schließen der Anwendung sollte automatisch der aktuelle Datenbestand noch einmal gespeichert werden.
- c) Der Anwender sollte ein neues bzw. weiteres Telefonbuch erstellen können.
- d)