Programmdokumentation

Der 3. Abgabe am 9.12.1998
über das Programm

Rechnungsbearbeitung

INHALTSVERZEICHNIS:

Bearbeiter:	Sascha Nemecek
Matrikelnummer:	9825815
Übung:	Einführung in das Programmieren - Syntaxpraktikum
Tutor:	Fischmeister
Abgabedatum:	09.12.1998

1 Rechnungserstellung

1.1 Modularisierung und Datenstruktur

1.1.1 Grundidee

Das Prinzip ist bei diesem Programm das gleiche wie beim Ersten, es wurden nur die Ausgaben vom Systemoutput in eine Liste umgeleitet und dort gespeichert. Wenn alle Daten eingelesen, und die Rechnungen berechnet wurden, wird der Frame geöffnet. Im Frame ist ein Scrollpane enthalten, der wiederum einen Canvas enthält. Die Größe des Canvas muss erst mit Hilfe von FontMetrics berechnet werden.
Im Gegensatz zum ersten Programm müssen auch Eingabeparameter berücksichtigt werden, die die Schriftgröße des auszugebenden Textes festlegen.

1.1.2 Wahl der Klassen

1.1.2.1 Klasse Abgabe3

Meine Hauptklasse. In der Mainmethode werden zuerst alle Daten mit Hilfe der anderen Klassen eingelesen. Dann wird die graphische Ausgabe gestartet. Die Graphikklasse wird als Observer der Rechnungsdaten bestimmt.

1.1.2.2 Klasse Artikel

In dieser Klasse werden die Artikeldaten geladen, gesichert und verwaltet. Dafür enthält sie eine *Unterklasse Daten*. Die *Daten* werden in einer **Hashtable** abgelegt, wo bei die Artikelnummer als Key verwendet wird. Die Artikelanzahl ist daher nur mehr durch den Hauptspeicher beschränkt.

1.1.2.3 Klasse AusgabeDaten

Ermöglicht das die verschiedenen Diagramme mit einem Datentyp arbeiten. Die *AusgabeDaten* werden vorher berechnet, und dann an das jeweilige Zeichenmethode übergeben.

1.1.2.4 Klasse DialogBearbeiten

Er möglicht das Bearbeiten der Rechnungsdaten in einem Dialogfenster. Folgende Veränderungen einer Rechnung können vorgenommen werden:

- Artikeleintrag in Rechnung löschen
  (falls sich keine Artikel mehr in der Rechnung befinden, wird sie gelöscht)
- Artikeleintrag einer Rechnung bearbeiten
- Neuen Artikeleintrag erstellen (noch nicht programmiert)

1.1.2.5 Klasse DialogEditElement

Ermöglicht das Bearbeiten eines Artikeleintrages einer Rechnung in einem Dialogfenster. Dabei kann aber nur die Menge des gewählten Artikels verändert werden.

1.1.2.6 DialogLöschen

Zeigt alle Rechnungen in einer Liste an. Der Benutzer kann nun einen Eintrag auswählen, und durch das betätigen des OK-Buttons die Rechnung löschen.

1.1.2.7 DialogNeu

Erzeugt eine neue Rechnung, wobei darauf geachtet wird, dass keine alte Rechnung überschrieben wird. Dieser Dialog ist noch nicht vollständig implementiert.

1.1.2.8 ElementeDaten

In dieser Classe werden die Artikeleinträge einer Rechnung abgelegt.

1.1.2.9 Menue

Ermöglicht das benutzen der DialogFenster.

1.1.2.10 MyColors

Da bei der Diagrammausgabe verschieden Farben verwendet werden sollen, kann mit dieser Klasse eine gewisse Farbpalette immer wieder durchlaufen werden.

1.1.2.11 MyException

Um meine auftretenden Fehler weiterzuleiten

1.1.2.12 MyGUI

Die graphische Ausgabe und der Aufbau des Frames wird in dieser Klasse geregelt.

1.1.2.13 Klasse Rechnung

In dieser Klasse werden die Rechnungsdaten geladen, gesichert und verwaltet. Dafür enthält sie eine *Unterklasse RechnungsDaten*. Die Rechnungen werden in einem **Vector** abgelegt. Die maximale Anzahl von Rechnungen ist daher auch nicht mehr beschränkt.

1.1.3 Wahl der Methoden

1.2 Pseudocode

Variablen definieren und initialisieren
File mit Artikeln öffnen
Solange Zeile vorhanden:
    Zeile einlesen
    Zeile in Substrings aufteilen (Tokenize) und Anzahl der Substrings überprüfen (muss 3 sein, sonst Fehlermeldung)
    Substrings in jeweiliges Format umwandeln (Artikelnummer: **Int**, Beschreibung: **String**, Preis: **double**)
    Wenn keine Fehlermeldung Artikeldaten im Datenarray speichern

File mit Rechnungen öffnen:
Solange Zeile vorhanden:
    Zeile einlesen
    Zeile in Substrings aufteilen (Tokenize) und Anzahl der Substrings überprüfen (muss 2 sein, sonst Fehlermeldung)
    Substrings in jeweiliges Format umwandeln (Artikelnummer: **Int**, Menge: **Int**)
    Wenn keine Fehlermeldung Rechnungsdaten im Rechnungsarray speichern

Rechnungsdaten durchgehen
    Artikeldaten zur Rechnung suchen
    Rechnungsergebnis berechnen und und Ausgabepuffer sicher

Frame initialisieren und die max. Canvasgrößeberechenen
Ausgabepuffer durchgehen und jeden Datensatz am Graphikoutput ausgeben

1.3 Source Code

1.3.1 Abgabe3.java

```
import java.awt.*;
import java.awt.event.*;

/**
 * Hauptklasse.
 * Hier werden die Daten initialisiert und die graphicsche Ausgabe gestartet.
 */
class Rechnungsbearbeitung {
    private Artikel artikelDaten;
    private Rechnung rechnungsDaten;

    final static String ARTIKELPFAD = "f:\\daten\\uni\\1.semester\\eprog\\abgabe3\\Files\\";
    final static String ARTIKELFILE = "artikel.txt";
    final static String RECHNUNGSPFAD = "f:\\daten\\uni\\1.semester\\eprog\\abgabe3\\Files\\";

    /**
     * Konstruktor.
     * Versucht die Daten einzulesen. Falls dies nicht mögliche ist, wird das Programm mit
     * einer Fehlermeldung abgebrochen.
     */
    Rechnungsbearbeitung () {
          try {
              artikelDaten = new Artikel (ARTIKELPFAD, ARTIKELFILE);
              rechnungsDaten = new Rechnung (RECHNUNGSPFAD);
              }
          catch (MyException e) {
              System.out.println (e.getMessage());
              System.exit (0);
              }

          // Startet graphische Ausgabe
          MyGUI m = new MyGUI (artikelDaten, rechnungsDaten);
          rechnungsDaten.addObserver (m);
          }

    /**
     * Hauptmethode.
     * Initialisiert Rechnungsdaten, Artikeldaten und Frame.
     */
    public static void main (String argv[]) {
          Rechnungsbearbeitung r = new Rechnungsbearbeitung ();
          }
    }
```

1.3.2 Artikel.java

```
import java.util.*;
import java.io.*;

/**
 * Klasse Artikel: Verwaltet die Artikeldaten.
 */
public class Artikel {

    /**
     * private Hashtable daten.
     * Hier werden alle Artikeldaten abgelegt.
     */
    private Hashtable daten = new Hashtable();


    /**
     * In diesem Format werden die Artikeldaten abgelegt.
     */
    private class Daten
        {
        String name = null;
        double preis = 0;

        /**
           * Konstruktor für ein neues Element.
           */
        Daten(String na, double p)
              {
              name = na;
              preis = p;
              }
        }
```

```
/**
 *    Konstuktor: ließt das angegebene File ein.
 *  Ließt das durch pfad und filename angegeben File ein und Wertet den Inhalt als Artikeldaten aus.
 *  Falls der Inhalt gültig ist, wird er in einer HashTable abgelegt.
 *  @param String pfad: Gibt das Verzeichnis des einzulesenden Files an.
 *  @param String filename: Gibt den Filenamen des einzulesenden Files an.
 *  @see StringtoArtikel.
 *  @exception MyException.
 *    @see MyException.
 */
Artikel (String pfad, String filename) throws MyException {
      String s = null;
      ReadFile file;

      // Versucht File zu öffnen. Wenn nicht möglich, Abbruch
      file = new ReadFile (pfad, filename);

      // Ließt solange Artikel ein, bis keine mehr vorhanden sind
      while ((s = file.readLine()) != null)
          {
          try
                {
                // Wandelt einen String ins Artikelformat um
                StringtoArtikel (s);
                }
          catch (MyException e)
                {
                // Fehlermeldung (ErrorDialog)
                System.out.println ("Fehler: " + e.getMessage());
                }
          }
      file.close ();
      if (daten.isEmpty())
          throw new MyException ("Es wurden keine Daten eingelesen - Programmabbruch");
      }


/**
 * Methode: String to Artikel.
 * Versucht den übergebenen String in der Hashtable "Artikel" abzulegen.
 */
private void StringtoArtikel (String s) throws MyException
      {
      int nummer = 0;
      String name = null;
      double preis = 0;

      // Initialisiert Stringteiler und definiert die Trennzeichen
      StringTokenizer T = new StringTokenizer (s, ",");

      try {
          // Überprüft, ob die Parameteranzahl korrekt ist
          if ((T.countTokens () != 3) || (s.charAt (s.length () - 1) == ','))
                throw new MyException ("Falsche Syntax: " + s + " - Datensatz ignoriert.");

          // Ließt die Artikelnummer, wenn nicht dreistellig Abbruch
          name = T.nextToken ();
          if (name.length() !=3)
                throw new MyException ("Ungültige Artikelnummer im angegebenen File.");
          nummer = Integer.parseInt (name);

          if ((nummer < 100) || (nummer > 399))
                throw new MyException ("Ungültige Artikelnummer im angegebenen File.");

          // Ließt Artikelnamen, Überprüft, ob der Artikelname zulässig ist, und bricht bei Fehler ab
          name = T.nextToken ();
          if (name.length() > 20)
                throw new MyException ("Artikelbezeichnung zu lang (max. 20 Zeichen)");

          // Ließt den Preis
          preis = Double.valueOf(T.nextToken()).doubleValue();
          }
      catch (NumberFormatException e)
          {
          throw new MyException (e.getMessage());
          }

      // Wenn kein Fehler bei der Umwandlung aufgetreten ist, Datensatz speichern
      daten.put (Integer.toString(nummer), new Daten (name,preis));
      }


/**
 * Methode find (int art).
 * Sucht in Artikeldaten nach übergebener Artikelnummer und gibt den Key bei Erfolg zurück.
 * @return Daten wenn gefunden - null, wenn nicht vorhanden.
 */
Daten find (String art)
      {
      return (Daten) daten.get (art);
      }


/**
 * Wandelt einen String in einen Integer um.
 * Dieser Methode darf nur für Strings verwendet werden, bei denen sichergestellt ist, dass sich sich umwandeln lassen.
 * @return int key.
 */
int key (String key)
      {
      return Integer.parseInt (key);
      }
```

```
/**
 * Gibt die Artikelnummer des übergebenen String zurück.
 * @return int Artikelnummer.
 */
int getArtikelNr (String s) {
      if (s == null)
          return 0;
      for (Enumeration e = daten.keys() ; e.hasMoreElements() ;) {
          String r = (String) e.nextElement();
          if (name (r).equals (s.substring (0, s.indexOf (" "))))
                return key (r);
          }
      return -1;
      }


/**
 * Gibt den Artikelnamen des gewünschten Artikel zurück.
 * @return den Artikelnamen.
 */
String name (String j)
      {
      return find(j).name;
      }


/**
 * Methode: preis.
 * Gibt den Preis des gewünschten Artikels zurück.
 * @return preis - der Preis des Artikels.
 */
double preis (String j)
      {return find(j).preis;}


/**
 * Methode int lenght ().
 * Liefert die Anzahl der gespeicherten Artikel.
 * @return Anzahl der Artikel.
 */
int length ()
      {return daten.size();}


/**
 * Methode: berechneMenge.
 * Berechnet die Anzahl der konsumierten Artikel.
 * @param r - Die gekauften Artikel.
 * @return Vector - mit Mengen und Beschriftung.
 */
public Vector berechneMenge (Rechnung r) {
      Vector v = new Vector ();
      for (Enumeration e = daten.keys() ; e.hasMoreElements() ;)
          {
          String s = (String) e.nextElement();
          int menge = r.getArtikelMenge (key (s));

          if (menge > 0)
                v.addElement (new AusgabeDaten (menge, name (s)));
          }
      return v;
      }


/**
 * Methode: berechneGrpMenge.
 * Berechnet die Anzahl der konsumierten Artikelgruppen.
 * @param r - Die gekauften Artikel.
 * @return Vector - mit Mengen und Beschriftung.
 */
public Vector berechneGrpMenge (Rechnung r) {
      String grps[] = {"Gruppe 1", "Gruppe 2", "Gruppe 3"};
      String grp = null;
      int index = 0;

      Vector v = new Vector ();
      for (Enumeration e = daten.keys() ; e.hasMoreElements() ;)
          {
          String s = (String) e.nextElement();
          grp = grps [Integer.parseInt (s.substring (0,1)) - 1];
          int menge = r.getArtikelMenge (key (s));

          if (menge > 0)
          if ((index = ArtikelGrpIndex (grp, v)) >= 0)
                {
                v.addElement (new AusgabeDaten (menge + ((AusgabeDaten)v.elementAt (index)).menge, grp));
                v.removeElementAt (index);
                }
          else
                v.addElement (new AusgabeDaten (menge, grp));
          }
      return v;
      }
```

```
   /**
     * Durchsucht den übergebenen Vektor nach einer Artikelgruppe und liefert deren Index zurück.
     * @param art - die gesuchte Artikelgruppe.
     * @param d - die zu untersuchender Vektor.
     * @return int - der Index der gesuchten Artikelgruppe.
     */
    private int ArtikelGrpIndex (String art, Vector d) {
          for (int i = 0; i < d.size (); i++)
              if ((((AusgabeDaten) d.elementAt (i)).bez).equals (art))
                    return i;
          return -1;
          }
    }
```

1.3.3 AusgabeDaten.java

```
/**
 * In diesem Format werden die Daten für die Ausgabe gespeichert.
 */
class AusgabeDaten {
    int menge;
    String bez;

    AusgabeDaten (int m, String s) {
          menge = m;
          bez = s;
          }
    }
```

1.3.4 DialogBearbeiten.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * Dialog Bearbeiten.
 * Ermöglicht das Bearbeiten einer oder mehrerer Rechnungen.
 * Zur Zeit ist nur das Löschen von Artikeln in Rechnungen und das bearbeiten von Artikeln möglich.
 */
class DialogBearbeiten extends Dialog implements ActionListener, ItemListener, Observer {
    Choice choiceRechnungen = new Choice();
    Panel pEingabe = new Panel ();
    List lElements = new List (5, false);
    Button bNeu = new Button ("Neu");
    Button bEd = new Button ("Bearbeiten");
    Button bDel = new Button ("Löschen");
    Button bOk = new Button ("OK");
    TextField tfArt = new TextField ();
    Rechnung rechnungsDaten;
    Artikel artikelDaten;
    Frame parent;


    /**
     * Methode: Update.
     * Führt bei einer Veränderung des zu überwachenden Objektes die fillListmethode des Dialoges aus.
     * @param o - das Überwachte Objekt.
     * @param arg - die Veränderung.
     */
    public void update(Observable o, Object arg) {
          fillList ();
          }


    /**
     * Konstrukor.
     * Hier wird der Dialog initialisiert.
     */
    DialogBearbeiten (Frame p, String titel, boolean modal, Rechnung v, Artikel a) {
          super(p,titel);
          parent = p;
          rechnungsDaten = v;
          artikelDaten = a;
          setBackground(Color.lightGray);
          setResizable(false);
          setLayout(new BorderLayout());

          // Choice füllen
        for (int i = 0; i < rechnungsDaten.RechnungsSize (); i++)
              choiceRechnungen.add(rechnungsDaten.getName(i));

          // Füllt die Liste
          fillList ();

          pEingabe.setLayout(new GridLayout(3,1));
          pEingabe.add(bNeu);
          pEingabe.add(bEd);
          pEingabe.add(bDel);

          choiceRechnungen.addItemListener(this);
          bOk.addActionListener(this);
          bNeu.addActionListener(this);
          bDel.addActionListener(this);
          bEd.addActionListener(this);

          lElements.addActionListener (this);
          lElements.addItemListener (this);
```

```
          add ("North", choiceRechnungen);
          add ("Center", lElements);
          add ("East", pEingabe);
          add ("South", bOk);

          // WindowListener zum Schließen des Dialoges
          addWindowListener(new Dialogschliesser ());
          pack();
          }


    /**


     * Schließt das Dialogfenster.
     */
    class Dialogschliesser extends WindowAdapter
          {
          public void windowClosing (WindowEvent e)
              {cancel ();}
          }


    /**
     * methode fillList.
     * Füllt die Liste mit dem Inhalt einer Rechnung.
     */
    private void fillList () {
          String sRechnung;

          // selektierte Rechnungsnummer holen
          if ((sRechnung = choiceRechnungen.getSelectedItem()) == null)
              cancel ();

          lElements.removeAll ();
          rechnungsDaten.resetRechnung ();
          rechnungsDaten.findRechnung (sRechnung);

          for (int i = 0; i < rechnungsDaten.elementsSize (); i++) {
              lElements.add (artikelDaten.name ("" + rechnungsDaten.elementArtikel ()) + "     " + rechnungsDaten.elementMenge
());
              rechnungsDaten.nextElement ();
              }
          }


    /**
     * Beendet den Dialog und kehrt zurück zum Hauptframe.
     */
    void cancel() {
          setVisible(false);
          dispose();
          ((Window)getParent()).toFront();
          getParent().requestFocus();
          }


    /**
     * Wertet die ActionEvent aus und reagiert je nach Event.
     */
    public void actionPerformed (ActionEvent e) {
          // Beendet Bearbeitung
          if (e.getActionCommand().equals("OK")) cancel();

          // noch nicht ausprogrammiert
          else if (e.getActionCommand().equals("Neu")) cancel();

          // Startet einen neuen Dialog zum verändern des ausgewählten Artikels
          else if (e.getActionCommand().equals("Bearbeiten")) {
              String s = lElements.getSelectedItem ();
              DialogEditElement dEd;

              if (s != null) {
                    dEd = new DialogEditElement (parent, s, artikelDaten, rechnungsDaten);
                    dEd.setVisible (true);
                    }
              }

          // Löscht den ausgewählten Artikeleintrag einer Rechnung
          else if (e.getActionCommand().equals("Löschen")) löschen();
          }


    /**
     * methode itemStateChanged.
     * Verarbeitet die Item Events des Choice- und Listobjekts..
     */
    public void itemStateChanged (ItemEvent e) {
          // Wenn eine andere Rechnung ausgewählt wurde - aktualisieren
          if (e.getSource () != lElements)
              fillList ();
          }
```

```
    /**
     * Löscht das selektierte Listenelement.
     */
    public void löschen() {
          String sElement;

          // selektierte Rechnungsnummer holen
          if ((sElement = lElements.getSelectedItem()) != null)
              {
              // Rechnungsnummer aus Choice entfernen
              lElements.remove (sElement);

              // Rechnungsnummer aus Daten entfernen
              rechnungsDaten.removeElement (sElement, artikelDaten);
              }
          }
    }
```

1.3.5 DialogEditElement.java

```
import java.awt.*;
import java.awt.event.*;

/**
 * Editiert ein Artikelelements.
 */
class DialogEditElement extends Dialog implements ActionListener, ItemListener {
    Button bOk = new Button ("OK");
    Button bCancel = new Button ("Cancel");
    Panel p = new Panel ();
    TextField tf = new TextField ("0", 1);
    Label lb = new Label ();
    Artikel artikelDaten;
    Rechnung rechnungsDaten;
    String art;


    DialogEditElement (Frame parent, String s, Artikel a, Rechnung r) {
          super(parent, "Edit");
          artikelDaten = a;
          rechnungsDaten = r;
          art = s;

          // Setzt Parameter für Dialog
        setBackground(Color.lightGray);
        setResizable(false);
        setLayout(new BorderLayout());

          // Initialisiert die ActionListener
          bOk.addActionListener (this);
          bCancel.addActionListener (this);
          lb.setText ("Stück: " + s.substring (0, s.indexOf (" ")));
          tf.setText (s.substring (s.indexOf (" ") + 4, s.length ()));

          // Fügt buttons in Panel ein
          p.setLayout (new GridLayout(1, 2));
            p.add (bOk);
            p.add (bCancel);

          // Positioniert die Objekte im Dialog
          add ("Center", tf);
          add ("North", lb);
          add ("South", p);

        // WindowListener zum Schließen des Dialoges
        addWindowListener(new Dialogschliesser ());

          // Richtet Objekte ein
        pack();
        }

    class Dialogschliesser extends WindowAdapter
          {
          public void windowClosing (WindowEvent e)
              {cancel ();}
          }

    /**
     * Beendet den Dialog und kehrt zurück zum Hauptframe.
     */
    void cancel() {
          setVisible(false);
          dispose();
          ((Window)getParent()).toFront();
          getParent().requestFocus();
          }
```

```
    /**
     * Verarbeitet ActionEvents.
     */
    public void actionPerformed (ActionEvent e) {
          // Beendet Bearbeitung, wenn Eingabe korrekt
          if (e.getActionCommand().equals ("OK"))
              {
              boolean err = false;
              int menge = 0;
              try {menge = Integer.parseInt (tf.getText ());}
              catch (NumberFormatException ex) {err = true;}

              // Wenn kein Fehler aufgetreten ist - sichern
              if ((menge > 0) && !err) {
                    rechnungsDaten.setMenge (menge, artikelDaten.getArtikelNr (art));
                    cancel ();
                    }
              }

          // Beendet Bearbeitung sofort - ohne Änderung
          if (e.getActionCommand().equals ("Cancel"))
              cancel ();
          }

    /**
     * methode itemStateChanged.
     * Verarbeitet die Item Events des Choice- und Listobjekts..
     */
    public void itemStateChanged (ItemEvent e) {
          }
    }
```

1.3.6 DialogLöschen.java

```
import java.awt.*;
import java.awt.event.*;

/**
 * Dialog Löschen.
 * Löschen einer oder mehrerer Rechnungen.
 */
class DialogLoeschen extends Dialog implements ActionListener, ItemListener {
    List choiceRechnungen = new List (5, false);
    Panel pEingabe = new Panel();
    Button bOk = new Button("OK");
    Button bCancel = new Button ("Cancel");
    Rechnung rechnungsDaten;


    /**
     * Konstrukor.
     * Hier wird der Dialog initialisiert.
     */
    DialogLoeschen(Frame parent, String titel, boolean modal, Rechnung v) {
          super(parent,titel);
          rechnungsDaten = v;
          setBackground(Color.lightGray);
          setResizable(false);

          // Liste füllen
        for (int i = 0; i < rechnungsDaten.RechnungsSize (); i++)
              choiceRechnungen.add(rechnungsDaten.getName(i));

          pEingabe.setLayout(new GridLayout(2,1));
            pEingabe.add(bOk);
            pEingabe.add(bCancel);
            bOk.addActionListener(this);
            bCancel.addActionListener(this);

          choiceRechnungen.addItemListener(this);
          add("Center", choiceRechnungen);
          add("South", pEingabe);

          // WindowListener zum Schließen des Dialoges
          addWindowListener(new Dialogschliesser ());
          pack();
          }


    /**


     * Schließt das Dialogfenster.
     */
    class Dialogschliesser extends WindowAdapter
          {
          public void windowClosing (WindowEvent e)
              {cancel ();}
          }


    /**
     * Beendet den Dialog und kehrt zurück zum Hauptframe.
     */
    void cancel() {
          setVisible(false);
          dispose();
          ((Window)getParent()).toFront();
          getParent().requestFocus();
          }
```

```
    /**
     * Wertet die ActionEvent aus und reagiert je nach Event.
     */
    public void actionPerformed (ActionEvent e) {
          // löschen aufrufen
          if (e.getActionCommand().equals("OK")) löschen();

          // Zurück zum Hauptframe
          if (e.getActionCommand().equals("Cancel")) cancel();
          }


    /**
     * methode itemStateChanged.
     * Verarbeitet die Item Events der Listbox.
     */
    public void itemStateChanged (ItemEvent e) {}


    /**
     * Methode löschen.
     * Löscht die ausgewählte Rechnung.
     */
    void löschen() {
          // selektierte Rechnungsnummer holen
          String sRechnung = choiceRechnungen.getSelectedItem();

          if (sRechnung != null)
              {
              // Rechnungsnummer aus Choice entfernen
              choiceRechnungen.remove (sRechnung);

              // Rechnungsnummer aus Daten entfernen
              rechnungsDaten.removeRechnung (sRechnung);

              if (rechnungsDaten.RechnungsSize () == 0)
                    cancel ();
              }
          }
    }
```

1.3.7 DialogNeu.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * Dialog Neu.
 * Fügt eine Rechnung hinzu.
 * Noch nicht funktionstüchtig, da noch die Ausprogrammierung der Artikelauswahl fehlt.
 */
class DialogNeu extends Dialog implements ActionListener, ItemListener {
    Panel pEingabe = new Panel ();
    Label errMsg = new Label ("", Label.CENTER);
    TextField tfNr = new TextField("0", 1);
    TextField tfKassa = new TextField("0", 1);
    Button bCancel = new Button ("Cancel");
    Button bOk = new Button ("OK");
    Rechnung rechnungsDaten;
    Artikel artikelDaten;


    /**
     * Konstrukor.
     * Hier wird der Dialog initialisiert.
     */
    DialogNeu (Frame parent, String titel, boolean modal, Rechnung v, Artikel a) {
          super(parent,titel);
          rechnungsDaten = v;
          artikelDaten = a;
          setBackground(Color.lightGray);
          setResizable(false);
          setLayout(new BorderLayout());

          pEingabe.setLayout (new GridLayout(3,2));
          pEingabe.add (new Label ("Rechnungnummer:"));
          pEingabe.add (new Label ("Kassa:"));
          pEingabe.add (tfNr);
          pEingabe.add (tfKassa);
          pEingabe.add (bOk);
          pEingabe.add (bCancel);

          bOk.addActionListener (this);
          bCancel.addActionListener (this);

          add ("Center", pEingabe);
          add ("South", errMsg);

          // WindowListener zum Schließen des Dialoges
          addWindowListener(new Dialogschliesser ());
          pack();
          }

    /**
     * Schließt das Dialogfenster.
     */
    class Dialogschliesser extends WindowAdapter
          {
          public void windowClosing (WindowEvent e)
              {cancel ();}
          }
```

```
    /**
     * Beendet den Dialog und kehrt zurück zum Hauptframe.
     */
    void cancel() {
          setVisible(false);
          dispose();
          ((Window)getParent()).toFront();
          getParent().requestFocus();
          }


    /**
     * Wenn Button gedrückt wurde
     */
    public void actionPerformed (ActionEvent e) {
          // Beendet Bearbeitung
          if (e.getActionCommand().equals("Cancel")) cancel();

          // Fügt neue Rechnung hinzu
          else if (e.getActionCommand().equals("OK")) add();
          }


    /**
     * methode itemStateChanged.
     * Verarbeitet die Item Events des Choice- und Listobjekts..
     */
    public void itemStateChanged (ItemEvent e) {
          }


    /**
     * Fügt die neue Rechnung zu den Daten hinzu.
     */
    public void add() {
          String sElement;
          int nr = 0, kassa = 0;
          boolean err = false;

          errMsg.setText ("");

          try {nr = Integer.parseInt (tfNr.getText ());}
          catch (
              NumberFormatException e) {errMsg.setText ("Ungültige Rechnungsnummer");
              err = true;
              }

          try {kassa = Integer.parseInt (tfKassa.getText ());}
          catch (
              NumberFormatException e) {errMsg.setText ("Ungültige Kassennummer");
              err = true;
              }

          // Wenn die Rechnungsnummer ungültig ist
          if ((nr <= 0) && !err) {
              err = true;
              errMsg.setText ("Rechnungsnummer zu klein");
              }

          // Wenn die Kassennummer ungültig ist
          else if ((kassa <= 0) && !err) {
              err = true;
              errMsg.setText ("Kassennummer zu klein");
              }

          // Wenn die Rechnung schon existiert
          else if (rechnungsDaten.RechnungExists (nr)) {
              err = true;
              errMsg.setText ("Rechnung existiert bereits");
              }

          // Wenn kein Fehler aufgetreten ist, wird die neue Rechnung angelegt
          if (!err) {
              Vector v = new Vector ();

              if (rechnungsDaten.addRechnung (nr, kassa, v))
                    errMsg.setText ("Daten angelegt");
              else
                    errMsg.setText ("Rechnung ist leer");
              }
          }
    }
```

1.3.8 ElementeDaten.java

```
/**
 * Klasse Daten.
 * In diesem Format werden die einzelnen Elemente einer Rechnung gespeichert.
 */
public class ElementeDaten {
    int artikel, menge;

    ElementeDaten (int a, int m)
        {
        artikel = a;
        menge = m;
        }
    }
```

1.3.9 Menue.java

```
import java.awt.*;
import java.awt.event.*;


/**
 * Erzeugt des Menü.
 * Das Menü enthält folgende Punkte:
 *  -) eine neue Rechnung eingeben.
 *  -) eine Rechnung verändern.
 *  -) eine Rechnung löschen.
 *
 *  -) das Programm beenden.
 */
class Menue extends MenuBar {
    private Menu menuRechnung;

    /**
     * Konstruktor.
     * Hier wird das Menü initialisiert.
     */
    public Menue(ActionListener listener){
          menuRechnung = new Menu ("Bearbeiten");
          addMenuItem(menuRechnung,"Rechnung Eingeben",listener);
          addMenuItem(menuRechnung,"Rechnung Ändern",listener);
          addMenuItem(menuRechnung,"Rechnung Löschen",listener);
          menuRechnung.addSeparator();
          addMenuItem(menuRechnung,"Beenden",listener);
          add(menuRechnung);
          }


    /**
     * Methode addMenuItem.
     * Reagiert auf Menüauswahlen.
     */
    void addMenuItem(Menu m, String name, ActionListener l) {
          MenuItem mi;
          mi = new MenuItem(name);
          mi.setActionCommand(name);
          mi.addActionListener(l);
          m.add(mi);
          }
   }
```

1.3.10 MyColors.java

```
import java.awt.*;

/**
 * Klassa MyColors.
 * Verwaltet meine Farbpalette, die für die Balken-/Kreisdiagramme notwendig ist.
 * Wenn man einmal eine Instanz erzeugt hat, erhält man mit der Methode getNext() die nächste Farbe.
 * Wenn alle Farbe verwendet wurden, wird von Neuem begonnen.
 */
class MyColors {
    private final static Color C[] = {Color.blue, Color.cyan, Color.darkGray, Color.green, Color.magenta, Color.lightGray,
                                      Color.orange, Color.pink, Color.gray, Color.red, Color.yellow};
    private int aktualColor = 0;

    /**
     * Konstruktor.
     * ohne Anweisung.
     */
    MyColors () {}


    /**
     * Methode: reset ().
     * Setzt die aktuelle Farbe auf die Erste zurück.
     */
    public void reset () {
          aktualColor = 0;
          }


    /**
     * Methode: getNext ().
     * Liefert die aktuelle Farbe und geht zur nächsten.
     * @return die aktuelle Farbe.
     */
    public Color getNext () {
          if (aktualColor >= C.length)
              reset ();
          return C [aktualColor++];
          }
    }
```

1.3.11 MyException.java

```
/**
 * Klasse: MyExceptions.
 * Leitet meine Exceptions weiter.
 */
class MyException extends Exception {

    /**
     * Konstruktor: MyExceptions.
     * @param String s: zu werfende Fehlermeldung.
     */
    MyException (String s)
          {
          super (s);
          }
     }
```

1.3.12 MyGUI.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * class MyGUI.
 * Hier wird die graphische Ausgabe erzeugt.
 * Mit Hilfe des Observer Interface werden die Rechnungsfiles überwacht.
 */
public class MyGUI extends Frame implements ItemListener, Observer, ActionListener {
    private Artikel artikelDaten;
    private Rechnung rechnungsDaten;
    private CheckboxGroup cbg1, cbg2;
    private Checkbox artikel, kassa, balken, kreis;
    private ScrollPane scrollpane;
    private OutputCanvas output;
    private Label header;

    final static String LBL_1 = "Verkaufte Artikel als Balkendiagramm";
    final static String LBL_2 = "Verkaufte Artikelgruppen als Kreisdiagramm";
    final static String LBL_3 = "Verkaufte Artikel pro Kassa als Balkendiagramm";
    final static String LBL_4 = "Verkaufte Artikel pro Kassa als Kreisdiagramm";
    final static int BL_VGAP = 10;
    final static int BL_HGAP = 10;


    /**
     * Konstruktor.
     * Öffnet Frame und initialisiert die Anzeige.
     * @param artikelDaten - die Artikeldaten.
     * @see Artikel.
     * @param rechnungsDaten - die Rechnungsdaten.
     * @see Rechnung.
     */
    MyGUI (Artikel artikelDaten, Rechnung rechnungsDaten) {
          // Initialisiert Frame
          super ("Rechnungsbearbeitung");
          setSize (600, 400);
          addWindowListener(new Fensterschliesser ());
          Menue mb = new Menue (this);
          setMenuBar (mb);

          // kopiert die Referenzen
          this.artikelDaten = artikelDaten;
          this.rechnungsDaten = rechnungsDaten;

          // Setzt Leyout auf BorderLayout
          setLayout (new BorderLayout(BL_HGAP, BL_VGAP));

            // Erzeugt Label:
            header = new Label (LBL_1, Label.CENTER);
            add ("North", header);

            // Erzeugt den Canvas + Scrollpane
            scrollpane = new ScrollPane ();
            output = new OutputCanvas ();
            scrollpane.add (output);
            add ("Center", scrollpane);
            scrollpane.doLayout ();

            // Setzt Scrollbareinheiten
            scrollpane.getHAdjustable().setUnitIncrement (40);
            scrollpane.getVAdjustable().setUnitIncrement (30);

            // Erzeugt die Radiobuttons und deren Listener
            Panel p = new Panel ();
              p.setLayout (new GridLayout(3, 2));
              cbg1 = new CheckboxGroup();
              cbg2 = new CheckboxGroup();
              p.add (new Label ("Was anzeigen?", Label.LEFT));
              p.add (new Label ("Wie anzeigen?", Label.LEFT));
              artikel = new Checkbox("Artikel", cbg1, true);
              artikel.addItemListener (this);
              p.add (artikel);
              balken = new Checkbox("Balkendiagramm", cbg2, true);
              balken.addItemListener (this);
              p.add (balken);
```

```
            kassa = new Checkbox("Kassen", cbg1, false);
            kassa.addItemListener (this);
            p.add (kassa);
            kreis = new Checkbox("Kreisdiagramm", cbg2, false);
            kreis.addItemListener (this);
            p.add (kreis);
          add ("South", p);

         // Zeigt fertigen Frame an
         setVisible (true);
         }


    /**
     * Methode paint.
     * führt keine Anweisung aus.
     */
    public void paint (Graphics g)
          {}


    /**
     * Methode: itemStateChanged.
     * Wertet Veränderungen in der CheckBoxGroup aus & veranlasst ein Neuzeichnen des Canvas.
     */
    public void itemStateChanged (ItemEvent e)
          {
        if ((e.getSource() == kassa) && output.artikel)
                    output.artikel = false;
          else if ((e.getSource () == artikel) && !output.artikel)
                    output.artikel = true;
        else if ((e.getSource () == kreis) && output.balken)
                    output.balken = false;
          else if ((e.getSource () == balken) && !output.balken)
                    output.balken = true;

          output.repaint ();
          }


    /**
     * Wertet die ActionEvents des Menüs aus.
     */
    public void actionPerformed (ActionEvent e) {
          String s = e.getActionCommand();

          // Dialog zum Rechnung eingeben aufrufen
          if (s.equals("Rechnung Eingeben")) {
              DialogNeu dNeu = new DialogNeu (this, "Rechnung Bearbeiten", false, rechnungsDaten, artikelDaten);
              dNeu.setVisible(true);
              }

          // Dialog zum Rechnung ändern aufrufen
          else if (s.equals("Rechnung Ändern")) {
                DialogBearbeiten dEdit = new DialogBearbeiten (this, "Rechnung Bearbeiten", false, rechnungsDaten,
artikelDaten);
              rechnungsDaten.addObserver (dEdit);
              dEdit.setVisible(true);
              }

          // Dialog zum Rechnung löschen aufrufen
          else if (s.equals("Rechnung Löschen")) {
              DialogLoeschen dLöschen = new DialogLoeschen (this, "Rechnung Löschen", false, rechnungsDaten);
              dLöschen.setVisible(true);
              }

          // Programm beenden
          else if (s.equals("Beenden")) System.exit(0);
          }


    /**
     * Klasse: OutputCanvas.
     * Die Zeichenfläche. Die Abmessungen des Canvas werden durch die auszugebenden Strings definiert.
     * Der Canvas ist aber mindestens so groß, wie der geöffnete Frame.
     */
    private class OutputCanvas extends Canvas
          {
          public boolean balken = true, artikel = true;
          public int anz = 0;

          /**
           * Konstruktor.
           * Hier wird die Größe des Canvas berechnet & gesetzt
           */
          OutputCanvas ()
              {
              Dimension groesse;
              groesse = scrollpane.getViewportSize ();
              setSize (groesse.width, groesse.height);
              }
```

```
        /**
         * Methode: paint.
         * Hier wird das gewünschte Diagramm ausgegeben.
         **/
        public void paint (Graphics g)
            {
            Vector h = new Vector ();

            // Auswertung, welches Diagramm gezeichnet werden soll
            if (artikel && balken) {
                  header.setText (LBL_1);
                  drawBalkenDiagramm (g, artikelDaten.berechneMenge (rechnungsDaten));
                  }
            else if (artikel && !balken) {
                  header.setText (LBL_2);
                  drawKreisDiagramm (g, artikelDaten.berechneGrpMenge (rechnungsDaten));
                  }
            else if (!artikel && balken) {
                  header.setText (LBL_3);
                  drawBalkenDiagramm (g, rechnungsDaten.berechneMenge ());
                  }
            else if (!artikel && !balken) {
                  header.setText (LBL_4);
                  drawKreisDiagramm (g, rechnungsDaten.berechneMenge ());
                  }
            }


        /**
         * Zeichnet Balkendiagramm.
         * @param g - die Graphikinstanz.
         * @param h - die Auszugebenden Daten in einem Vektor.
         */
        private void drawBalkenDiagramm (Graphics g, Vector h) {
            int hspace = 0, xbez = 20, ybez, maxX, maxY, breite = 20;
            double fact = 0.0;
            Dimension groesse;
            MyColors m = new MyColors ();
            FontMetrics fm;

            groesse = scrollpane.getViewportSize ();
            maxX = groesse.width * 7 / 10;
            maxY = groesse.height * 7 / 10;
            ybez = groesse.height / 10;
            g.setFont (fitFont (ybez));
            fm = g.getFontMetrics ();

            // Berechnet die max. Breite einer Bezeichnung + höchsten Balken
            for (int i = 0; i < h.size (); i++)
                  {
                  if (fm.stringWidth(((AusgabeDaten)h.elementAt (i)).bez) + 10 > hspace)
                      hspace = fm.stringWidth(((AusgabeDaten)h.elementAt (i)).bez) + 10;
                  if (fm.stringWidth("" + ((AusgabeDaten)h.elementAt (i)).menge) + 10 > hspace)
                      hspace = fm.stringWidth("" + ((AusgabeDaten)h.elementAt (i)).menge) + 10;
                  if (fact < (1.0*((AusgabeDaten)h.elementAt (i)).menge/(maxY)))
                      fact = 1.0*((AusgabeDaten)h.elementAt (i)).menge/(maxY);
                  }

            if (fact == 0)
                  fact = 1;
            if (hspace <= breite)
                  hspace = breite * 2;
            if ((h.size() * hspace + 2 * xbez) < groesse.width)
                  output.setSize (groesse.width, groesse.height);
            else
                  // Passt den Canvas an das auszugebende Diagramm an
                  output.setSize (h.size() * hspace + 2 * xbez, groesse.height);

            scrollpane.doLayout ();

            // Gibt das Diagramm aus
            for (int i = 0; i < h.size (); i++) {
                  g.setColor (Color.black);
                  drawStringZentriert (g, ((AusgabeDaten)h.elementAt (i)).bez, xbez + hspace*i, maxY + ybez*5/2, hspace);
                  drawStringZentriert (g, "" + ((AusgabeDaten)h.elementAt (i)).menge, xbez + hspace*i, maxY + ybez -
(int)Math.round(((AusgabeDaten)h.elementAt (i)).menge / fact), hspace);
                  g.setColor (m.getNext ());
                  g.fillRect(xbez + hspace*i + hspace/2 - breite/2, maxY + ybez*3/2 -
(int)Math.round(((AusgabeDaten)h.elementAt (i)).menge / fact), breite, (int)Math.round(((AusgabeDaten)h.elementAt (i)).menge /
fact));
                  }
            }
```

```
         /**
          * Zeichnet Kreisdiagramm.
          * @param g - die Graphikinstanz.
          * @param h - die Auszugebenden Daten in einem Vektor.
          */
         private void drawKreisDiagramm (Graphics g, Vector h) {
             int hspace = 0, xbez = 20, ybez, maxX, maxY, breite = 20;
             int ges = 0, start = 0, radius;
             Dimension groesse;
             MyColors m = new MyColors ();
             FontMetrics fm;

             groesse = scrollpane.getViewportSize ();
             maxX = groesse.width * 8 / 10;
             maxY = groesse.height * 8 / 10;
             radius = Math.min (maxX, maxY)/2;
             ybez = maxY;
             g.setFont (fitFont (maxY/8));
             fm = g.getFontMetrics ();

             // Berechnet die Gesamtanzahl der Artikel
             for (int i = 0; i < h.size (); i++)
                   ges += ((AusgabeDaten)h.elementAt (i)).menge;

             if (hspace <= breite)
                   hspace = breite * 2;

             // Passt den Canvas an das auszugebende Diagramm an
             output.setSize (groesse.width, groesse.height);
             scrollpane.doLayout ();

             // Gibt das Diagramm aus
             for (int i = 0; i < h.size (); i++) {
                   int bogen = (int) Math.round (360.0 * ((AusgabeDaten)h.elementAt (i)).menge / ges);

//                 int texty = (int) Math.round (radius * Math.asin (start + bogen/2));
//                 int textx = (int) Math.round (texty * Math.atan (start + bogen/2));
//                 System.out.println (textx + " " + texty + " " + Math.asin (0.1));
/*                 g.setColor (m.getNext ());
                   g.fillArc (maxX/9, maxY/9, 200, 200, 0, 180);*/
//                 g.setColor (Color.black);
//                 drawStringZentriert (g, ((AusgabeDaten)h.elementAt (i)).bez, groesse.width/2 + textx, groesse.height/2 +
texty, hspace);
                   g.setColor (m.getNext ());
                   g.fillArc (groesse.width/2 - radius, groesse.height/2 - radius, radius*2, radius*2, start, bogen);
                   start += bogen;
                   }
             g.setColor (Color.black);
             g.drawOval (groesse.width/2 - radius, groesse.height/2 - radius, radius*2, radius*2);
             }
         }


    /**
     * Berechnet die optimale Schriftgrösse.
     */
    private Font fitFont (int max) {
          FontMetrics fm;
          Font font = null;

          int groesse = 20;
          while (groesse > 6) {
              font = new Font ("Serif", Font.PLAIN, groesse);
              fm = getFontMetrics (font);
              if (fm.getHeight () + 5 < max) break;
              groesse--;
              }
          return font;
          }


    /**
     * Gibt einen String zentriert aus.
     */
    private void drawStringZentriert (Graphics g, String s, int x, int y, int m) {
          FontMetrics fm = g.getFontMetrics ();
          int w = fm.stringWidth (s);

          g.drawString (s, x + (int)m/2 - (int)w/2, y);
          }


    /**
     * Methode: Update.
     * Führt bei einer Veränderung des zu überwachenden Objektes die Repaintmethode des Canvas aus.
     * @param o - das Überwachte Objekt.
     * @param arg - die Veränderung.
     */
    public void update(Observable o, Object arg) {
          output.repaint ();
          }

    /**
     * Klasse: Fensterschliesser.
     * Implementiert das windowClosing Event zum Schließen des Frames.
     */
    class Fensterschliesser extends WindowAdapter
          {
          public void windowClosing (WindowEvent e)
              {System.exit (0);}
          }
    }
```

1.3.13 Rechnung.java

```
import java.util.*;
import java.io.*;

/**
 * Klasse Rechnung.
 * Verwaltet und beinhaltet die Rechnungsdaten.
 */
class Rechnung extends Observable {
    private int aktuelleRechnung = 0;
    private int aktuellesElement = 0;

    /**
     * Vektor daten.
     * Hier werden die Rechnungselemente gespeichert.
     */
    private Vector daten = new Vector ();


    /**
     * Klasse RechnungsDaten.
     * In diesem Format wird eine gesamte Rechnung abgelegt.
     */
    public class RechnungsDaten
          {
          int rechnungsNr, kassa;
          Vector elements;
          RechnungsDaten (int r, int k, Vector e)
              {
              rechnungsNr = r;
              kassa = k;
              elements = e;
              }
          }


    /**
     * Konstruktor.
     * Ließt alle Rechnungsfiles aus dem angegebenen Verzeichnis ein.
     * @param String pfad: Verzeichnis aus dem die Rechnungsfiles gelesen werden sollen.
     * @exception MyException.
     * @see MyException.
     */
    Rechnung (String pfad) throws MyException {
          // Öffnet Directory
          File file = new File (pfad);

          // Ließt alles Filenamen im angegebenen Directory ein
          String f[] = file.list ();

          for (int i = 0; i < f.length; i++)
                 if  ((f[i].length()  >  14)  &&  (f[i].substring  (0,  8).equalsIgnoreCase  ("rechnung"))  &&  (f[i].substring
(f[i].length() - 4, f[i].length()).equalsIgnoreCase (".txt")))
                    {
                    StringTokenizer st = new StringTokenizer (f[i], ".");
                    if ((st.countTokens() == 3) && (isValidNumber (st.nextToken())) && (isValidKassa (st.nextToken())))
                        ReadRechnung (pfad, f[i]);
                    }

          Observable o = new Observable ();
          }


    /**
     * Methode: load.
     * Ließt die Rechnungen aus angegebenem File ein.
     * @param pfad - gibt den Pfad an, in dem sich das Rechnungsfile befindet.
     * @param filename - gibt den Filenamen an.
     * @exception MyException.
     * @see MyException.
     */
    void ReadRechnung (String pfad, String filename) throws MyException
          {
          String s = null;
          int artind = 0;
          ReadFile file;
          boolean err;
          Vector v = new Vector ();

          // Öffnet File
          file = new ReadFile (pfad, filename);

          // Ließt bis zum Fileende ein, oder bricht bei mehr als 10 Rechnungssätzen ab
          while ((s = file.readLine()) != null)
              {
              err = false;
              ElementeDaten h = null;
              // Wandelt Sting in Rechnungsdatensatz um
              try {h = StringtoDaten (s);}
              catch (MyException e) {
                    System.out.println (e.getMessage ());
                    err = true;}
```

```
            // Überprüft, ob dieser Artikel in der Rechnung schon vorkommt
            //  ja: ergänzt den alten Rechnungssatz
            //  nein: legt neuen Rechnungssatz an
            if (!err)
                  if ((artind = artikelIndex (h.artikel, v)) >= 0)
                      {
                      h.menge += ((ElementeDaten)v.elementAt (artind)).menge;
                      v.addElement (h);
                      v.removeElementAt (artind);
                      }
                  else
                      v.addElement (h);
            }
         if (!v.isEmpty())
                daten.addElement  (new  RechnungsDaten  (Integer.parseInt(filename.substring  (8,  filename.indexOf  (".".))),
Integer.parseInt (filename.substring (filename.indexOf (".")+1, filename.length () - 4)), v));
         }


    /**
     * Methode StringtoRechnung.
     * Versucht den übergebenen String in das Datenformat umzuwandeln.
     * @param s - der umzuwandelnde String.
     * @return Daten - Rechnungsdatensatz.
     */
    private ElementeDaten StringtoDaten (String s) throws MyException
         {
         int artikel = 0, stück = 0, artnr = 0;
         String help = null;

         // Initialisiert Stringteiler und definiert die Trennzeichen
         StringTokenizer st = new StringTokenizer (s, ",");

         try {
             if ((st.countTokens () != 2) || (s.charAt (s.length () - 1) == ','))
                   throw new MyException ("Falsche Syntax: " + s + " - Datensatz ignoriert.");
             artikel = Integer.parseInt (st.nextToken());
             stück = Integer.parseInt (st.nextToken());
             }
         catch (NumberFormatException e)
             {
             throw new MyException ("Fehler im Datensatz: " + s + " - Datensatz ignoriert.");
             }
         return new ElementeDaten (artikel, stück);
         }


    /**
     * Methode: isValidNumber (String s).
     * Überprüft, ob der Filename eine gültige Rechnungsnummer enthält.
     * @param s - Erster Teil des Filenamens.
     * @return true wenn gültig.
     */
    private boolean isValidNumber (String s) {
        try {if (Integer.parseInt (s.substring (8, s.length())) <= 0)
                   return false;}
          catch (NumberFormatException e) {return false;}
          return true;
          }


    /**
     * Methode: isValidKassa (String s).
     * Überprüft, ob der Filename eine gültige Kassennummer enthält.
     * @param s - Zweiter Teil des Filenamens.
     * @return true wenn gültig.
     */
    private boolean isValidKassa (String s) {
        try {if (Integer.parseInt (s) <= 0)
                   return false;}
          catch (NumberFormatException e) {return false;}
          return true;
          }


    /**
     * Methode: berechneMenge.
     * Berechnet die Anzahl der konsumierten Artikel pro Kassa.
     * @param r - Die gekauften Artikel.
     * @return Vector - mit Mengen und Beschriftung.
     */
    Vector berechneMenge () {
          Vector v = new Vector ();
          int ind = 0;

          for (int i = 0; i < daten.size (); i++) {
              ind = kassaIndex (((RechnungsDaten)daten.elementAt (i)).kassa, v);
              if (ind >= 0)
                        {
                              v.addElement  (new  AusgabeDaten  (VerkaufteArtikelproKassa  (((RechnungsDaten)daten.elementAt
(i)).elements) + ((AusgabeDaten)v.elementAt (ind)).menge,
                        "" + ((RechnungsDaten)daten.elementAt (i)).kassa));
                        v.removeElementAt (ind);
                        }
              else
                   v.addElement (new AusgabeDaten (VerkaufteArtikelproKassa (((RechnungsDaten)daten.elementAt (i)).elements),
"" + ((RechnungsDaten)daten.elementAt (i)).kassa));
              }
          return v;
          }
```

```
/**
 * Berechnet die Gesamtmenge der Artikel in einer Rechnung.
 * @param d - die Rechnungen.
 * @return Verkauften Artikel in einer Rechnung.
 */
private int VerkaufteArtikelproKassa (Vector d) {
      int anz = 0;
      for (int i = 0; i < d.size (); i++)
          anz += ((ElementeDaten) d.elementAt (i)).menge;
      return anz;
      }


/**
 * Durchsucht den übergebenen Vektor nach einer bestimmten Kassa und liefert deren Index zurück.
 * @param art - der gesuchte Artikel.
 * @param d - die zu untersuchender Vektor.
 * @return int - der Index der gesuchten Kassa.
 */
private int kassaIndex (int kassa, Vector d) {
      for (int i = 0; i < d.size (); i++)
          if ((((AusgabeDaten) d.elementAt (i)).bez).equals ("" + kassa))
                return i;
      return -1;
      }


/**
 * Durchsucht eine Rechnung nach einem bestimmten Artikel und liefert dessen Index zurück.
 * @param art - der gesuchte Artikel.
 * @param d - die zu untersuchende Rechnung.
 * @return int - der Index des gesuchten Artikels.
 */
private int artikelIndex (int art, Vector d) {
      for (int i = 0; i < d.size (); i++)
          if (((ElementeDaten) d.elementAt (i)).artikel == art)
                return i;
      return -1;
      }


/**
 * Berechnet die Gesamtmenge eines Artikels in einer Rechnung.
 * @param art - der gesuchte Artikel.
 * @param d - die zu untersuchende Rechnung.
 * @return int - die Menge des gesuchten Artikels.
 */
private int CountArtOccurence (int art, Vector d) {
      int i;
      if ((i = artikelIndex (art, d)) >= 0)
          return ((ElementeDaten) d.elementAt (i)).menge;
      return 0;
      }


/**
 * Berechnet die Gesamtmenge eines Artikels in allen Rechnungen.
 * @param art - der gesuchte Artikel.
 * @return occ - die Menge des gesuchten Artikels.
 */
public int getArtikelMenge (int art) {
      int occ = 0;
      for (int i = 0; i < daten.size (); i++)
          occ += CountArtOccurence (art, ((RechnungsDaten)daten.elementAt (i)).elements);
      return occ;
      }


/**
 * Liefert eine Referenz auf den Vektor mit den gespeicherten Daten.
 */
public Vector getElements () {
      return daten;
      }


/**
 * Liefert eine Referenz auf den Vektor mit den gespeicherten Daten.
 * @param index - der Index der gesuchte Rechnungsdaten.
 * @return Vector - mit den Rechnungsdaten.
 */
public Vector getElements (int index) {
      return ((RechnungsDaten) daten.elementAt (index)).elements;
      }


/**
 * Liefert eine Referenz auf das Element mit den gespeicherten Daten.
 * @param nr - die Artikelnummer des gesuchten Elements.
 */
public ElementeDaten getElement (int nr) {
      Vector e = getElements (aktuelleRechnung);
      for (int i = 0; i < e.size (); i++)
          {
          if (nr == ((ElementeDaten) e.elementAt (i)).artikel)
                return ((ElementeDaten) e.elementAt (i));
          }
      return null;
      }
```

```
    /**
     * Erhöht aktuellesElement, wenn möglich.
     */
    public void nextElement () {
          if (elementsSize () > aktuellesElement)
              aktuellesElement++;
          }


    /**
     * Liefert die Menge des aktuellen Elements zurück.
     * @returns int - Menge.
     */
    public int elementMenge () {
          Vector e = getElements (aktuelleRechnung);
          return ((ElementeDaten)e.elementAt (aktuellesElement)).menge;
          }


    /**
     * Liefert die Artikelnummer des aktuellen Elements zurück.
     * @return int - Artikelnummer.
     */
    public int elementArtikel () {
          Vector e = getElements (aktuelleRechnung);
          return ((ElementeDaten)e.elementAt (aktuellesElement)).artikel;
          }


    /**
     * Setzt die aktuelle Rechnung und das aktuelle Element auf 0 zurück.
     */
    public void resetRechnung () {
          aktuelleRechnung = 0;
          aktuellesElement = 0;
          }


    /**
     * Sucht nach einer Rechnung und setzt, wenn gefunden aktuelleRechnung auf die gesuchte.
     * @param s - die gesuchte Rechung.
     */
    public void findRechnung (String s) {
          for (int i = 0; i < daten.size (); i++)
              if (s.substring (8,s.indexOf (".")).equals ("" + ((RechnungsDaten)daten.elementAt (i)).rechnungsNr))
                    aktuelleRechnung = i;
          }


    /**
     * Liefert die Anzahl der Element der aktuellen Rechnung zurück.
     * @returns int - Menge der Elemente.
     */
    public int elementsSize () {
          return ((RechnungsDaten)daten.elementAt (aktuelleRechnung)).elements.size ();
          }


    /**
     * Liefert die Anzahl der Rechnungen zurück.
     * @returns int - Anzahl der Rechnungen.
     */
    public int RechnungsSize () {
          return daten.size ();
          }


    /**
     * Liefert den Filename der gesuchten Rechnung.
     * @param i - die gesucht Rechnung.
     * @return Filenamen der Rechnung.
     */
    public String getName (int i) {
          return  "Rechnung"  +  ((RechnungsDaten)daten.elementAt  (i)).rechnungsNr  +  "."  +  ((RechnungsDaten)daten.elementAt
(i)).kassa + ".txt";
          }


    /**
     * Löscht eine Rechnung.
     * @param s - die zu löschende Rechnung.
     */
    public void removeRechnung (String s) {
          for (int i = 0; i < daten.size (); i++)
              if (s.substring (8,s.indexOf (".")).equals ("" + ((RechnungsDaten)daten.elementAt (i)).rechnungsNr))
                    daten.removeElementAt (i);

          // Informiert Observer
          setChanged();
          notifyObservers();
          }
```

```
/**
 * Löscht ein Artikelelement aus der aktuellen Rechnung.
 * @param s - der zu löschende Artikel.
 * @param a - Instanz auf die artikelDaten
 */
public void removeElement (String s, Artikel a) {
      Vector e = getElements (aktuelleRechnung);
      for (int i = 0; i < e.size (); i++)
          if (s.substring (0,s.indexOf (" ")).equals (a.name ("" + ((ElementeDaten)e.elementAt (i)).artikel)))
                e.removeElementAt (i);

      // Informiert Observer
      setChanged();
      notifyObservers();
      }


/**
 * Überprüft, ob eine Rechnung schon Existiert.
 * @param nr - die zu vergleichende Nummer.
 * @returns true, wenn die Rechnung bereits vorhanden ist.
 */
public boolean RechnungExists (int nr) {
      for (int i = 0; i < daten.size (); i++)
          {
          System.out.println (nr + " " + ((RechnungsDaten)daten.elementAt (i)).rechnungsNr);
          if (nr == ((RechnungsDaten)daten.elementAt (i)).rechnungsNr)
                return true;
          }
      return false;
      }


/**
 * Fügt eine neue Rechnung hinzu.
 * @param nr - die anzulegende Rechnung.
 * @param kassa - die Kasse.
 */
public boolean addRechnung (int nr, int kassa, Vector v) {
      if (v.size () > 0) {
          daten.addElement (new RechnungsDaten (nr, kassa, v));

          // Informiert Observer
          setChanged();
          notifyObservers();
          return true;
          }
      return false;
      }


/**
 * Setzt die Artikelmenge des angegebenen Artikels, der aktuellen Rechnung.
 * @param menge - die neue Artikelmenge
 * @param artnr - die Artikelnummer des zu verändernden Artikels
 */
public void setMenge (int menge, int artnr) {
      ElementeDaten el = getElement (artnr);

      el.menge = menge;

      // Informiert Observer
      setChanged();
      notifyObservers();
      }
}
```