Programmdokumentation

Der 2. Abgabe am 19.11.1998
über die Programme

Rechnungserstellung
&
Banner

INHALTSVERZEICHNIS:

Bearbeiter:	Sascha Nemecek
Matrikelnummer:	9825815
Übung:	Einführung in das Programmieren - Syntaxpraktikum
Tutor:	Fischmeister
Abgabedatum:	19.11.1998

1 Erstes Programm: Rechnungserstellung

1.1 Modularisierung und Datenstruktur

1.1.1 Grundidee: Struktogramm

Variable & Konstanten initialisieren	
File mit Artikeln öffnen & solange Zeilen vorhanden:	
	Zeile einlesen
	Zeile auswerten und in Array von Artikel schreiben
	Bei Fehler diesen Artikel ignorieren
File mit Rechnungen öffnen & solange Zeilen vorhanden:	
	Zeile einlesen
	Zeile auswerten und in Array von Rechnungen schreiben
	Bei Fehler diese Rechnung ignorieren
Berechnen und am GUI ausgeben	

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 Beispiel1

Sie erweitert die Klasse Frame und ist somit für meine Ausgabe zuständig. In der Mainmethode werden zuerst alle Daten mit Hilfe der anderen Klassen eingelesen und die Übergabeparameter ausgewertet, dann der Frame geöffnet, und die auszugebenden Daten im Canvas dargestellt.

1.1.2.2 Klasse Artikeln

In dieser Klasse werden die Artikeldaten geladen, gesichert und verwaltet. Dafür enthält sie eine *Unterklasse Daten*, die zu einem Array of Daten zusammengefasst wird. Die maximale Artikelanzahl ist durch dieses Array auf 30 beschränkt.

1.1.2.3 Klasse Rechnung

In dieser Klasse werden die Rechnungsdaten geladen, gesichert und verwaltet. Dafür enthält sie eine *Unterklasse Rechnungen*, die zu einem Array of Rechnungen zusammengefasst wird. Die maximale Anzahl von Artikeln in einer Rechnung ist dadurch auf 10 beschränkt.

1.1.2.4 Klasse Berechnung

Die Daten der Klasse Artikel und Rechnung werden hier verknüpft (die Berechnung wird durchgeführt) und in den Ausgabepuffer gelegt.

1.1.2.5 Klasse MagList

Verwaltet eine doppelt verkettete List, die als Ausgabepuffer dient. Es wird in einem Datensatz jeweils der auszugebende String, und das Format (Text/Überschrift) gespeichert.

1.1.2.6 Klasse MyException

Leitet auftretende Exceptions weiter.

1.1.2.7 Klasse ReadFile

Modifizierte ReadFile-Klasse. Es werden keine Ausgaben mehr an den Standardoutput geschickt, sondern die Ausgaben über Exceptions an die aufrufende Methode zurückgeliefert.

1.1.3 Wahl der Methoden

1.1.3.1 Klasse Beispiel1

- **Beispiel1()**
  Der Konstruktor. Hier wird der Frame initialisiert, und die Canvasgröße berechnet.
- **public static void main(String[])**
  Hier werden die Daten initialisiert, eingelesen und der Frame geöffnet.
- **private static void getParameters(String[])**
  Hier werden die Übergabeparameter ausgewertet.
- **private static String[] getRechnungsFilenames(String)**
  Sucht im durch den String angegebenen Pfad nach „rechnung*.txt“-Files und liefert ein Array of Strings zurück.

1.1.3.2 Klasse Artikeln

- **Artikeln()**
  Der Konstruktor. Hier wird die Klassenvariable *anz* auf 0 gesetzt.
- **void DatenLesen(String, String, MagList)**
  Öffnet das durch die Strings angegebene File und ließt die darin enthaltenen Zeilen in ein String ein. Dieser String wird dann an StringtoArtikel weitergeleitet. Dieser Vorgang wird solange wiederholt, bis die maximale Anzahl von Artikeln (=30) erreicht ist, oder das Fileende erreicht wurde.
- **private StringtoArtikel(String)**
  Wandelt den übergebenen String in das Artikelformat um. Tritt dabei ein Fehler auf, wir eine Exception geworfen und der Datensatz ignoriert. War die Umwandlung erfolgreich, wird der Datensatz gesichert.
- **int find(int)**
  Sucht in den vorhanden Datensätzen nach einer Artikelnummer, und gibt dessen Position an die Aufrufende Methode zurück.
- **int length()**
  Gibt die Anzahl der Artikel an die aufrufenden Methode zurück.
- **String name(int)**
  Gibt den Artikelnamen des gewünschten Artikels (int) zurück.
- **int nummer(int)**
  Gibt die Artikelnummer des gewünschten Artikels (int) zurück.
- **double preis(int)**
  Gibt den Preis des gewünschten Artikels (int) zurück.
- **void printall(MagList)**
  Gibt alle Artikel in den Ausgabepuffer aus.

1.1.3.3 Klasse Rechnung

- **Rechnung()**
  Der Konstruktor. Hier wird die Klassenvariable *anz* auf 0 gesetzt.
- **void load(String path, String filename, MagList l) throws MyException**
  Läd das angegebene Rechnungsfile und wertet die Zeilen aus. Auftretende Fehler werden in den Ausgabepuffer geschrieben.
- **private void StringtoRechnung(String s) throws MyException**
  Wandelt einen eingelesenen String in das Rechnungsformat um.
- **int anz()**
  Gibt die Anzahl der vorhandenen Rechnungen zurück.
- **void destroy()**
  Löscht die letzte Rechnung.
- **int find(int art)**
  Sucht im Array of Rechnungen nach einem Datensatz mit der Artikelnummer **art**, und liefert dersen Index.
- **int nummer(int j)**
  Gibt die Artikelnummer der gewünschten Rechnung[**j**] zurück.
- **int menge(int j)**
  Gibt die Artikelmenge der gewünschten Rechnung[**j**] zurück.

1.1.3.4 Klasse Berechnung

- **Berechnung(Artikeln a, Rechnung e, MagList l)**
  Führt alle Berechnungen aus, und speichert die Ausgaben im Ausgabepuffer.

1.1.3.5 Klasse MagList

- **public void add(String x, boolean s)**
  Erzeugt einen neuen Datensatz mit dem übergebenen Inhalt an der aktuellen Position.
- **public boolean eol()**
  Liefert true, wenn sich der Datenzeiger am Ende der Liste befindet.
- **public void gotoNext()**
  Setzt den Datenzeiger auf das nächste Element.
- **public void gotoLast()**
  Setzt den Datenzeiger auf das vorherige Element.
- **public void reset()**
  Setzt den Datenzeiger auf das erste Element.
- **public String getData()**
  Liefert den auszugebenden String des aktuellen Elements.
- **public boolean getStyle()**
  Liefert das Format des aktuellen Elements.

1.1.3.6 Klasse MyException

Enthält nur den Konstruktor.

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 Beispiel1.java

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


class Beispiel1 extends Frame {
    final static String PFAD = "f:\\daten\\uni\\1.semester\\eprog\\abgabe1\\beispiel1";
    final static String FILENAME = "ARTIKEL.TXT";
    final static int STARTWIDTH = 300, STARTHEIGHT = 400;
    private static Font textfont, headerfont;
    private int textheight, headerheight, textwidht, headerwidht, enderg, maxwidth;
    private int rechmax [] = new int [6];
    private int artmax [] = new int [3];
    private static MagList ausg;

    /**
     * Konstruktor.
     * Hier wird der Frame initialisiert. Es wird ein Scrollpane und ein Canvas erzeugt.
     * Der Canvas wird dann in den Scrollpane gelegt. Bevor der Canvas aber erzeugt wird, werden
     * seine Abmessungen mit Hilfe von FontMetrics berechnet.
     */
    public Beispiel1 ()
          {
          // Auslesen der FontMetrics-Informationen
          headerheight = getFontMetrics (headerfont).getHeight();
          textheight = getFontMetrics (textfont).getHeight();
          textwidht = getFontMetrics (textfont).getMaxAdvance ();
          String s = null;

          // Berechnung der Canvasgrösse + Infosuche für Formatierung
          maxwidth = 0;
          int n = 0, maxheight = 0;

          for (ausg.reset (); !ausg.eol (); ausg.gotoNext ())
              // Wenn Überschrift: Breitenüberprüfung, und Längenberechnung
              if (ausg.getStyle ())
                    {
                    maxheight += headerheight + 3 * textheight;
                    if ((n = getFontMetrics (headerfont).stringWidth (ausg.getData ())) > maxwidth - 100)
                        maxwidth = n + 100;
                    }
              // Wenn Text: Breitenüberprüfung, und Längenberechnung
              else
                    {
                    maxheight += 1.3 * textheight;
                    if ((n = getFontMetrics (textfont).stringWidth (ausg.getData ())) > maxwidth - 100)
                        maxwidth = n + 100;
                    s = ausg.getData ();
                    StringTokenizer T = new StringTokenizer (s,"\t");
                    switch (T.countTokens ())
                        {
                        case 6: for (int i = 0; i < 6; i++)
                                      if ((n = getFontMetrics (textfont).stringWidth (T.nextToken())) > (rechmax[i] - 20))
                                          rechmax[i] = n + 20;
                                  break;
                        default: break;
                        }
                    }
          enderg = rechmax [0] + rechmax [1] + rechmax [2] + rechmax [3] + rechmax [4];

          // Layout setzten, Canvas und ScrollPane initalisieren
          setLayout (new BorderLayout());
            OutputCanvas output = new OutputCanvas (maxwidth + 40, maxheight + textheight);
            ScrollPane scrollpane = new ScrollPane ();
            scrollpane.add (output);
            add (scrollpane);
            scrollpane.doLayout ();

            // Setzt Scrollbareinheiten
            scrollpane.getHAdjustable().setUnitIncrement (textwidht);
            scrollpane.getVAdjustable().setUnitIncrement (textheight);
          }

    /**
     * Methode Fensterschliesser.
     * Implementier windowClosing Event in Frame.
     */
    private class Fensterschliesser extends WindowAdapter
          {
          public void windowClosing (WindowEvent e)
              {System.exit (0);}
          }
```

```
/**
 * Klasse OutputCanvas.
 * Zeichenebene zur Ausgabe der Daten.
 */
private class OutputCanvas extends Canvas
      {
      /**
       * Konstruktor.
       * Die Abmessungen werden initialisiert.
       */
      OutputCanvas (int w, int h)
          {
          if (w < STARTWIDTH)
                w = STARTWIDTH;
          if (h < STARTHEIGHT)
                h = STARTHEIGHT;
          setSize (w, h);
          }

      /**
       * Methode paint
       */
      public void paint (Graphics g)
          {
          g.setFont (textfont);
          int i= 0, y;
          String s = null;

          for (ausg.reset (); !ausg.eol (); ausg.gotoNext ())
                {
                if (ausg.getStyle())
                    {
                    g.setFont (headerfont);
                    i += 3 * textheight;
                    g.drawString (ausg.getData(), 30, i);
                    g.setFont (textfont);
                    i += headerheight;
                    }
                else
                    {
                    y = 30;
                    i += 1.3 * textheight;
                    s = ausg.getData ();
                    StringTokenizer T = new StringTokenizer (s, "\t");
                    switch (T.countTokens ())
                          {
                          case 6: for (int j = 0; j < 6; j++)
                                        {
                                        g.drawString (T.nextToken(), y, i);
                                        y += rechmax [j];
                                        }
                                  break;
                          case 5: g.drawString (T.nextToken (), y, i);
                                    y += enderg; T.nextToken(); T.nextToken(); T.nextToken();
                                    g.drawString (T.nextToken (), y, i);
                                    break;
                          case 3: for (int j = 0; j < 3; j++)
                                        {
                                        g.drawString (T.nextToken(), y, i);
                                        y += rechmax [j];
                                        }
                                  break;
                          default:
                              if (ausg.getData () != "=") g.drawString (ausg.getData (), 30, i);
                              else g.drawLine (y, i, y + maxwidth, i);
                          break;
                          }
                    }
                }
          }
      }
```

```
    // Hauptprogramm
    public static void main(String argv[])
          {
          ausg = new MagList ();

          getParameters (argv);

          // Initialisierung der Varible a, dabei wird die Liste der Artikel eingelesen
         Artikeln a = new Artikeln ();

          // Einlesen der Artikel
          try {a.DatenLesen (PFAD, FILENAME, ausg);}
          catch (MyException e) {ausg.add (e.getMessage(), false);}

          if (a.length () > 0)
              {
              // Ausgabe der vorhanden Artikel in Zwischenbuffer
              a.printall (ausg);

              // Initialisiert die Rechnungsklasse
              Rechnung r = new Rechnung ();

              // Sucht alle Rechnungsfiles im angegebenen Pfad
              String f[] = getRechnungsFilenames (PFAD);

              // Falls keine Rechnungsfiles gefunden wurden
              if (f.length == 0)
                    ausg.add ("Im Angegebenen Verzeichnis wurden keine Rechnungsfiles gefunden.", false);

              // Einlesen der Rechnungen aus dem File & deren Ausgabe
              for (int i = 0; i < f.length; i++)
                    {
                    ausg.add (f[i], true);

                    try {
                        r.load (PFAD, f[i], ausg);
                        }
                    catch (MyException e)
                        {
                        ausg.add (e.getMessage(), false);
                        }
                    Berechnung b = new Berechnung (a, r, ausg);
                    r.destroy ();
                    }
              }

          // Startet Ausgabe in Frame
          Frame frame = new Beispiel1 ();
          frame.setSize (STARTWIDTH, STARTHEIGHT);
          frame.setTitle ("Beispiel 1: Rechnungserstellung");
          frame.addWindowListener (new Fensterschliesser());
          frame.setVisible (true);
          }

    // Ließt Directory in ein Array of Strings ein & Löscht alle Files aus der Liste,
    // die kein Rechnungsfile sind. Das Array liefert dann alle "rechnung*.txt"-Files
    // an aufrufende Methode
    private static String[] getRechnungsFilenames (String pfad)
          {
          // Öffnet Directory
          File file = new File (pfad);

          // Ließt alles Filenamen im angegebenen Directory ein
          String f[] = file.list ();
          int anz = 0;

          for (int i = 0; i < f.length; i++)
              if ((f[i].substring (0, 8).equalsIgnoreCase ("rechnung")) && (f[i].substring (f[i].length() - 4,
f[i].length()).equalsIgnoreCase (".txt")))
                    f [anz++] = f [i];

          // Erstellt Array mit den gesuchten Dateinamen
          String h[] = new String[anz];
          for (int i = 0; i < anz; i++)
              h[i] = f[i];

          // Retourniert Referenz auf Liste mit Rechnungsfiles
          return h;
          }
```

```
    // Wertet die Übergabeparameter aus
    private static void getParameters (String argv[])
          {
          int headersize = 16, textsize = 14;

          // Auswertung der Übergabeparameter
          switch (argv.length)
              {
              // ohne Parameter, Defaultwerte verwenden
              case 0: break;

              // Bei 1 Parameter, wird versucht die Schriftgrösse zu setzen
              case 1:
                    try {
                        headersize = textsize = Integer.parseInt (argv [0]);
                        }
                    catch (NumberFormatException e)
                        {
                        // Bei Fehler: Fehlermeldung und Standardwerte setzen
                        ausg.add ("Falsches Argument: " + e.getMessage (), false);
                        ausg.add ("Es werden die Defaultwerte verwendet", false);
                        headersize = 16;
                        textsize = 14;
                        }
                    break;

              // Bei 2 Parametern, Überschriftsgrösse und Textgrösse setzen
              case 2:
                    // setzen der Überschriftsgrösse
                    try {
                        headersize = Integer.parseInt (argv [0]);
                        }
                    catch (NumberFormatException e)
                        // Bei Fehler: Fehlermeldung und Standardwerte setzen
                        {
                        ausg.add ("Falsches Argument: " + e.getMessage (), false);
                        ausg.add ("Es werden die Defaultwerte verwendet", false);
                        headersize = 16;
                        }

                    // setzen der Textschriftgrösse
                    try {
                        textsize = Integer.parseInt (argv [1]);
                        }
                    catch (NumberFormatException e)
                        // Bei Fehler: Fehlermeldung und Standardwerte setzen
                        {
                        ausg.add ("Falsches Argument: " + e.getMessage (), false);
                        ausg.add ("Es werden die Defaultwerte verwendet", false);
                        textsize = 14;
                        }
                    break;
              default:
                    ausg.add ("Falsche Argumentenanzahl. Es werden die Defaultwerte verwendet", false);
                    break;
              }

          // definieren der Schriftarten mit der ausgewerteten Schriftgrösse
          textfont = new Font ("Monospaced", Font.PLAIN, textsize);
          headerfont = new Font ("Monospaced", Font.BOLD, headersize);
          }
    }
```

1.3.2 Artikeln.java

```
import java.util.*;

// Verwaltet die Artikel
class Artikeln
    {
    private Daten[] a = new Daten[30];
    private int anz;

    // Konstruktor
    Artikeln ()
          {
          anz = 0;
          }


    // Hier werden die Artikeldaten abgelegt
    private class Daten
        {
        int nummer = 0;
        String name = null;
        double preis = 0;


        // Konstruktor für neues Element
        Daten(int nr, String na, double p)
              {
                    nummer= nr;
              name = na;
              preis = p;
              }
        }
```

```
// Konstruktor: Ließt die Artikel ein
void DatenLesen (String path, String filename, MagList l) throws MyException
      {
      String s = null;
      ReadFile file;

      // Versucht File zu öffnen. Wenn nicht möglich, Abbruch
      try {
          file = new ReadFile (path, filename);
          }
      catch (MyException e)
          {
          throw new MyException (e.getMessage ());
          }

      while ((s = file.readLine()) != null)
          {
          // Wenn zu viele Artikel: Abbruch
          if (anz >= 30)
                throw new MyException ("Zu viel Artikel im File - Es wurden nur die ersten 30 berücksichtigt.");

          try
                {
                // Wandelt einen String ins Artikelformat um
                StringtoArtikel (s);
                }
          catch (MyException e)
                {
                l.add ("Fehler: " + e.getMessage(), false);
                }
          }
      file.close ();
      if (anz == 0)
          throw new MyException ("Es wurden keine Daten eingelesen - Programmabbruch");
      }


// Versucht den übergebenen String im Array "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);

          // 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
      a[anz++] = new Daten (nummer,name,preis);
      }


// Gibt die Liste aller vorhanden Artikel aus
void printall (MagList l)
      {
      l.add ("Liste der vorhandenen Artikel:", true);
      for (int i = 0; i < anz; i++)
          l.add (a[i].nummer + ", " + a[i].name + ", " + a[i].preis, false);
      l.add ("", false);
      }

// Sucht in Artikeldaten nach übergebener Artikelnummer und gibt sie bei Erfolg zurück
int find (int art)
      {
      for (int i = 0; i < anz; i++)
          if (art == a[i].nummer)
                return (i);
      return -1;
      }

// Gibt die Artikelnummer des gewünschten Artikel zurück
int nummer (int j)
      {
      return a[j].nummer;
      }
```

```
    // Gibt den Artikelnamen des gewünschten Artikel zurück
    String name (int j)
          {
          return a[j].name;
          }


    // Gibt die Artikelnummer des gewünschten Artikel zurück
    double preis (int j)
          {
          return a[j].preis;
          }

    // liefert die Anzahl der Artikel
    int length ()
          {return anz;}
    }
```

1.3.3 Rechnung.java

```
import java.util.*;

// Verwaltet die Rechensätze
public class Rechnung
    {
    private Rechnungen [] r = new Rechnungen [10];
    private int anz;

    // Konstruktor
    Rechnung ()
          {
          anz = 0;
          }

    // Hier werden die Rechensätze abgelegt
    private class Rechnungen
          {
          int artikel;
          int menge;
          Rechnungen (int a, int m)
              {
              artikel = a;
              menge = m;
              }
          }

    // ließt die Rechnungen aus angegebenem File ein
    void load (String path, String filename, MagList l) throws MyException
          {
          String s = null;
          int stück = 0, trennzeichen = 0, artikel = 0, artnr = 0;
          ReadFile file;

          try {
              // Öffnet File
              file = new ReadFile (path, filename);
              }
          catch (MyException e)
              {
              throw new MyException (e.getMessage ());
              }

          // Ließt bis zum Fileende ein, oder bricht bei mehr als 10 Rechnungssätzen ab
          while ((s = file.readLine()) != null)
              {
              // Abbruch, wenn mehr als 10 Rechensätzen
            if (anz > 10)
                  throw new MyException ("Zu viele Artikel in dieser Rechnung.\nEs wurden nur die ersten 10 berücksichtigt.");
              try
                    {
                    // Wandelt Sting in Rechnungsdatensatz um
                    StringtoRechnung (s);
                    }
              catch (MyException e)
                    {
                    l.add (e.getMessage (), false);
                    }
              }
          }


    // Versucht den übergebenen String im Array "Rechnungen" abzulegen
    private void StringtoRechnung (String s) throws MyException
          {
          int artikel = 0, stück = 0, artnr = 0;
          String help = null;

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

          try {
              if ((T.countTokens () != 2) || (s.charAt (s.length () - 1) == ','))
                    throw new MyException ("Falsche Syntax: " + s + " - Datensatz ignoriert.");
              artikel = Integer.parseInt (T.nextToken());
              stück = Integer.parseInt (T.nextToken());
              }
          catch (NumberFormatException e)
              {
              throw new MyException ("Fehler im Datensatz: " + s + " - Datensatz ignoriert.");
              }
```

```
          // Überprüft, ob dieser Artikel in der Rechnung schon vorkommt
          //  ja: ergänzt den alten Rechnungssatz
          //  nein: legt neuen Rechnungssatz an
          if ((artnr = find (artikel)) >= 0)
              r [artnr].menge += stück;
          else
              r [anz++] = new Rechnungen (artikel, stück);
          }


    // Sucht in Rechnungssätzen nach angegebener Artikelnummer
    int find (int art)
          {
          for (int i = 0; i < anz; i++)
              if (art == r[i].artikel)
                    return (i);
          return -1;
          }

    // Liefert die Anzahl der geladenen Rechnungen zurück
    int anz ()
          {
          return anz;
          }

    // Setzt die Einträge auf 0 zurück
    void destroy ()
          {
          anz = 0;
          }


    // Liefert die gesuchte Artikelnummer
    int nummer (int j)
          {
          return r[j].artikel;
          }


    // Liefert die gesuchte Artikelmenge
    int menge (int j)
          {
          return r[j].menge;
          }
    }
```

1.3.4 Berechnung

```
import java.text.*;

// Führt die Berechnungen durch
public class Berechnung
    {
    Berechnung (Artikeln a, Rechnung e, MagList l)
          {
          int j=0;
          double gesamtpreis = 0;
          String trennung = "=";
          NumberFormat N = NumberFormat.getInstance ();
          N.setMaximumFractionDigits (2);
          N.setMinimumFractionDigits (2);
          N.setGroupingUsed (false);
          l.add ("lfd.Nr.\tArt.Nr.\tBezeichnung\tPreis pro Stk.\tMenge\tGesamtpreis\t", false);
          for(int i = 0; i < e.anz(); i++)
              if(0 <= (j = a.find (e.nummer(i))))
                    {
                    l.add (i + 1 + "\t" + a.nummer(j) + "\t" + a.name(j) + "\t" + a.preis(j) + "\t" + e.menge(i) + "\t" +
N.format(a.preis(j) * e.menge(i)), false);
                    gesamtpreis += a.preis(j) * e.menge(i);
                    }
              else
                    l.add (i + 1 + "\t" + e.nummer (i) + "\tArtikel nicht gefunden", false);
          l.add (trennung, false);
          l.add ("Gesamtpreis:\t \t \t \t" + N.format(gesamtpreis) + "", false);
          }
    }
```

1.3.5 MyException.java

```
/**
 * Klasse: MyExceptions
 * Leitet meine Exceptions weiter
 */
class MyException extends Exception {
    MyException (String s)
          {
          super (s);
          }
     }
```

1.3.6 MagList.java

```
public class MagList
    {
    private Node start, now, end;

    public MagList ()
          {
          now = null;
          start = null;
          end = null;
          }

    public void add (String x, boolean s)
          {
          if (start == null)
              {
              start = new Node (x, s, null, null);
              end = now = start;
              }
          else
              {
              Node T = new Node (x, s, now.next, now);
              now.next = T;
              now = T;
              if (now.next == null)
                    end = T;
              }
          }

    public boolean eol ()
          {
          return now == null;
          }

    public void gotoNext ()
          {
          now = now.next;
          }

    public void gotoLast ()
          {
          now = now.last;
          }

    public void reset ()
          {
          now = start;
          }

    public String getData ()
          {
          return now.data;
          }

    public boolean getStyle ()
          {
          return now.style;
          }

    private class Node
          {
          String data;
          boolean style;
          Node next;
          Node last;

          Node (String d, boolean s, Node n, Node l)
              {
              data = d;
              style = s;
              next = n;
              last = l;
              }
          }
    }
```

1.3.7 ReadFile.java

```
import java.io.*;

public class ReadFile {
          private DataInputStream dis;
          private boolean fehler;

          public ReadFile(String path, String name) throws MyException {
                    try {
                              dis = new DataInputStream(new FileInputStream(new File(path, name)));
                              fehler = false;
                              }
                    catch (Exception e) {
                              fehler = true;
                              throw new MyException ("Fehler beim Öffnen: " + e.getMessage());
                              }
                    }

          public ReadFile(String pathname) throws MyException {
                    try {
                              dis = new DataInputStream(new FileInputStream(new File(pathname)));
                              fehler = false;
                              }
```

```
                catch (Exception e) {
                        fehler = true;
                        throw new MyException ("Fehler beim Öffnen: " + e.getMessage());
                        }
                }

        public String readLine() throws MyException {
                String s = null;
                if (fehler)
                        return null;
                try {
                        s = dis.readLine();
                        }
                catch (IOException e) {
                        throw new MyException ("Fehler beim Lesen: " + e.getMessage());
                        }
                return s;
                }

        public void close() throws MyException {
                try {
                        fehler = true;
                        dis.close();
                        }
                catch (IOException e) {
                        throw new MyException ("Fehler beim Schließen: " + e.getMessage());
                        }
                }
        }
```

2 Zweites Programm: Datumsumwandlung

2.1 Modularisierung und Datenstruktur

2.1.1 Wahl der Klassen

2.1.1.1 Banner

Die Hauptklasse! Sie dient zur Initialisierung der Daten und der Erzeugung der graphischen Oberfläche.

2.1.1.2 Banners

Es soll aus einem File die auszugebenden Zeilen eingelesen werden. Diese Zeilen enthalten den **Vergrößerungsfaktor** und nach einem Leerzeichen den auszugebenden String. Da nicht bekannt ist, wieviele Zeilen dieses File enthält, werden sie in einer verketteten Liste gespeichert. Für die Verwaltung dieser Liste und das Einlesen der Daten wurde die Klasse **Banners** erzeugt.

2.1.1.3 MyChars

Aus einem gegebenen Verzeichnis sollen alle „*.def"-Files gelesen werden. Diese Files enthalten jeweils die Informationen, wie ein Zeichen gezeichnet werden soll (A.def steht für das Zeichen a, usw.). Bei diesen Informationen ist zwischen zwei Arten zu unterscheiden:

- Kurve <x> <y> <Außenradius> <Anfangswinkel> <Bogenwinkel> und
- Linie <x1> <y1> <x2> <y2>.

Da auch hier die Anzahl der „*.def"-File variieren kann, werden die Daten wieder in einer Liste gespeichert, wobei ein Datensatz, die Bezeichnung des Zeichens (z.B.: A) und eine weiter Liste, mit den Zeichenanweisungen enthält. Für die Verwaltung dieser Liste und das Einlesen der Daten wurde die Klasse **Banners** erzeugt.
Die graphische Ausgabe der Zeichen wurde auch in dieser Klasse verwirklicht.

2.1.1.4 MagList

Da die Klassen *Banners* und *Chars* nach eine Liste verlangen, wurde eine allgemeine Klasse zur Verwaltung dieser Listen verwirklicht.

2.1.2 Wahl der Methoden

2.1.2.1 Banner

- **public Banner()**
  Constructor. Hier wird der Zeichen Canvas initialisiert.
- **public static void main(String args[])**
  Das Hauptprogramm. Hier werden die benötigten Daten geladen, und das Frame initialisiert

2.1.2.2 Banners

- **Banners(String pfad, String filename)**
  Constructor. Ließt die auszugebenden Zeilen aus dem angegebenen File ein und legt sie in einer Liste ab.
- **public void reset()**
  Setzt die Liste data auf ersten Eintrag zurück.
- **public String getString()**
  Gibt den String (Bannerzeile), des aktuellen Bans zurück.
  Returns: Den aktuellen String.
- **public int getSize()**
  Gibt den Vergrößerungsfaktor, des aktuellen Bans zurück.
  Returns: Den aktuellen Vergrößerungsfaktor.
- **public boolean nextElement()**
  Geht in der Liste auf das nächste Element, wenn das aktuelle noch nicht das Letzte ist und liefert true zurück. Wenn das aktuelle Element das letzte ist, liefert die Methode false.
  Returns: Gibt true zurück, wenn ein nächstes Element existiert hat.
- **public boolean eol()**
  Liefert *true* zurück, wenn das Ende der Liste erreicht ist.
- **public int getMaxWidth()**
  liefert die Länge (in Pixel) des größten auszugebenden Strings in Abhängigkeit vom Vergrösserungfaktor.

- **public int getMaxHeight()**
  Liefert die maximale Höhe aller Strings.

2.1.2.3 MyChars

- **MyChars(String pfad) throws MyException**
  Constructor. läd die "*.def" aus dem angegebenen Verzeichnis
- **private void readDefFile(String pfad, String file)**
  Wertet die im File gegebenen Zeilen aus und Speichert sie in data
  Parameters:
  pfad - Gibt das Verzeichnis an, in dem sich das File befindet
  file - Gibt das zu ladende File an
- **public void drawMyString(Graphics g, String s, int size, int x, int y)**
  Gibt den übergebenen String an der übergebenen XY-Position aus.
  Parameters:
  s - Der auszugebende String.
  size - Der Vergrößerungsfaktor.
  x - Die Startposition x.
  y - Die Startposition y.
- **public void findMyChar(Graphics g, char c, int size, int x, int y)**
  Sucht den übergebenen Char in der Liste 'data'.
  Parameters:
  c - Das auszugebende Zeichen
  size - Der Vergrößerungsfaktor.
  x - Die Startposition x.
  y - Die Startposition y.
- **private void drawMyChar(Graphics g, MagList c, int size, int x, int y)**
  Geht die Zeichenanweisungen durch, und zeichnet den Char.
  Parameters:
  c - Liste mit den Zeichenanweisungen.
  size - Der Vergrößerungsfaktor.
  x - Die Startposition x.
  y - Die Startposition y.
- **private void drawKurve(Graphics g, int d[], int size, int x, int y)**
  Zeichnet einen Bogen. Dabei wird zuerst ein Bogen in der Vordergrundfarbe, und dann ein zweiter um eine Einheit kleinern Bogen und der Hintergrundfarbe gezeichnet.
  Parameters:
  d[] - Feld mit den Größenangaben für fillArc().
  size - Der Vergrößerungsfaktor.
  x - Die Startposition x.
  y - Die Startposition y.
  private void drawStrich(Graphics g, int d[],int size, int x, int y)
  Zeichnet einen Strich, wo bei zwischen Strich nach unten und Strich nach Rechts unterschieden wird.
  Parameters:
  d[] - Feld mit den Größenangaben für fillPolygon().
  size - Der Vergrößerungsfaktor.
  x - Die Startposition x.
  y - Die Startposition y.
- **public void setForeBackColor(Color f, Color b)**
  Setzt die Vorder- und Hintergrundfarbe. Dieser werden von der Methode drawKurve benötigt. Wenn die Übergabeparameter null sind, werden die Standardwerte (schwarz/weiß) gesetz.
  Parameters:
  f - Die Vordergrundfarbe.
  b - Die Hintergrundfarbe.

2.1.2.4 MagList

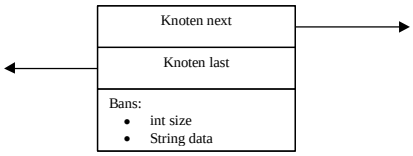
- **public MagList()**
  Konstruktor. Hier werden die Hilfsnodes zum Verwalten der Daten initialisiert.
- **public void add(Object x)**
  Fügt ein neues Element in die Liste mit der größe des übergebenen Objektes ein.
- **public boolean eol()**
  Überprüft, ob das Ende der Liste erreicht wurde.
  Returns: true, wenn das Ende der Liste erreicht, sonst false.
- **public void gotoNext()**
  Setzt den Hilfspointer now auf das nächste Element in der Liste.
- **public void gotoLast()**
  Setzt den Hilfspointer now auf das vorherige Element in der Liste.
- **public void reset()**
  Setzt den Hilfspointer now zurück auf den Start der Liste.
- **public Object getData()**
  Returns: Liefert das gespeicherte Objekt, der aktuellen Listenposition zurück.
- **public void destroy()**
  Setzt alle Hilfnodes auf null zurück -> Löschen der Liste.

2.1.3 Wahl der Datentypen

Die Klassen **MyChars** und **Banners** verlangen nach eigenen Datentype, zur Aufbewahrung der Daten. Da diese in einer Liste abgelegt werden, wurde sie gleich als Element einer Liste dargestellt.

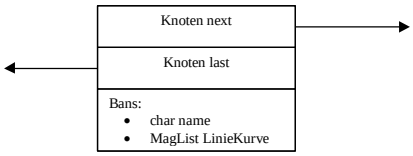
2.1.3.1 Banners

Bans-Elemente in einer Liste:



2.1.3.2 MyChars

Chars-Elemente in einer Liste, wobei wiederum eine Liste mit den Zeichenanweisungen enthalten ist:



2.2 Pseudocode

Initialisieren der Konstaten und Variablen
„Banner"-File einlesen und in geeigenter Form (*Bans*) sichern:

- Solange das Fileende nicht erreicht, die eingelesene Zeile in *Bans* ablegen.

„*.def"-Files einlesen und in geeigneter Form (*Chars*) sichern:

- Solange ein „*.def"-File gefunden wird, dieses öffnen:
  Solange das Fileende nicht erreicht, die eingelesenen Zeilen in *Chars* ablegen.

GUI (Frame) initialisieren (Abmessungen, Titel, Fensterschließer, ...)

- Scollpane und darinliegenden Canvas initialisieren, wobei die Abmessungen des Canvas erst berechnet werden (kann aber nicht kleiner als der Frame sind).

In der Paintmethode des Canvas:

- Solange *Bans* vorhanden sind, wird der String in der gewünschten Größe ausgegeben.

2.3 Source Code

2.3.1 Banner.java

```
/**
 * Bannerprogramm: Ließt Strings aus einem File ein und gibt diese in der vorher geladenen Schrift aus
 *
 * Die Strings sind im File "banner.txt" enthalten das nach folgenden Schema aufgebaut ist:
 *    Datensatz: "10 Das ist der Text" (Die Zahl vor dem Text stellt den Vergrößerungsfaktor dar)
 *
 * Die Schriftart setzt sich aus mehreren "*.def"-Files zusammen, wo bei "A.def" für den Buchstaben A steht, usw.
 * Ein "*.def" File kann nun folgende Inhalte haben, aus denen sich ein Buchstabe zusammensetzt:
 *    1.) "Strich <x1> <y1> <x2> <y2>"
 *    2.) "Kurve <x> <y> <Außenradius> <Anfangswinkel> <Bogenwinkel>"
 */

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

public class Banner extends Frame
    {
    private static MyChars c;
    private static Banners b;
    final static int STARTWIDTH = 400, STARTHEIGHT = 300;
    final static String PFAD = "f:\\daten\\uni\\1.semester\\eprog\\abgabe2\\beispiel2\\letters\\", FILE = "banner.txt";

    /**
     * Constructor.
     * Hier wird der Zeichen Canvas initialisiert.
     */
    public Banner ()
          {
          setLayout (new BorderLayout());
            OutputCanvas output = new OutputCanvas ();
            ScrollPane scrollpane = new ScrollPane ();
            scrollpane.add (output);
            add (scrollpane);
            scrollpane.doLayout ();

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

    /**
     * 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
          {
          /**
           * Konstruktor.
           * Hier wird die größe des Canvas berechnet & gesetzt
           */
          OutputCanvas ()
              {
              int w = b.getMaxWidth () + 30, h = b.getMaxHeight () + 30;
              if (w < STARTWIDTH)
                    w = STARTWIDTH;
              if (h < STARTHEIGHT)
                    h = STARTHEIGHT;
              setSize (w, h);
              }

          /**
           * Methode: paint
           * Hier werden die einzelnen Zeilen des auszugebenden Textes gelesen und an die Methode drawMyString (..)
           * aus der Classe MyChars übergeben.
           **/
          public void paint (Graphics g)
              {
              int i = 30;
              for (b.reset (); !b.eol (); b.nextElement ())
                    {
                    i += 2 * b.getSize ();
                    c.drawMyString (g, b.getString (), b.getSize (), 30, i);
                    i += 9 * b.getSize ();
                    }
              }
          }

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

```
    /**
     * Das Hauptprogramm.
     * Hier werden die benötigten Daten geladen, und das Frame initialisiert
     */
    public static void main(String args[])
          {
          // Einlesen der *.def Files mit den Anweisungen zum Zeichnen der Zeichen
          c = new MyChars (PFAD);

          // Einlesen des Files mit den auszugebenden Texten
          b = new Banners (PFAD, FILE);

          Frame f = new Banner ();
          f.setTitle ("Banner");
          f.setSize (STARTWIDTH, STARTHEIGHT);
          f.setVisible (true);
          f.addWindowListener (new Fensterschliesser());

          // Übergibt die Zeichnungsfarben an die Klasse Chars
          c.setForeBackColor (f.getForeground (), f.getBackground ());
          }
    }
```

2.3.2 Banners.java

```
/**
 * Klasse: Banners.
 * Hier werden die Bans gesichert und verwaltet.
 */
public class Banners
    {
    private MagList data;

    /**
     * Klasse: Bans.
     * In diesem Format werden die einzelen Bannertexte und deren Vergrößerungsfaktor abgelegt.
     */
    private class Bans
          {
          int size;
          String data;

          Bans (int s, String d)
              {
              size = s;
              data = d;
              }
          }

    /**
     * Constructor.
     * Ließt die auszugebenden Zeilen aus dem angegebenen File ein und legt sie in einer Liste ab.
     */
    Banners (String pfad, String filename)
          {
          data = new MagList ();
          String s = null;
          ReadFile file;
          int size = 0, t;
          boolean fund = false;

          // versucht das gewünschte File im angegebenen Pfad zu öffenen
          file = new ReadFile (pfad, filename);

          // Solange Daten vorhanden Zeilen einlesen und auswerten
          while ((s = file.readLine()) != null)
              {
              fund = true;
              if ((t = s.indexOf (' ')) != 0)
                    {
                    try {size = Integer.parseInt (s.substring (0, t));}
                    catch (NumberFormatException e) {}
                    s = s.substring (t+1);
                    if ((size > 0) && (s.length() > 0))
                        data.add (new Bans (size, s));
                    else System.out.println ("Ungültige Zeile gefunden: " + s);
                    }
              else System.out.println ("Fehler beim Lesen der Zeile: " + s);
              }

          // Schließt das File wieder, falls  eines geöffnet wurde
          if (fund) file.close ();
          }

    /**
     * Methode: reset.
     * Setzt die Liste data auf ersten Eintrag zurück.
     */
    public void reset ()
          {
          data.reset ();
          }
```

```
    /**
     * Methode: getString.
     * Gibt den String (Bannerzeile), des aktuellen Bans zurück.
     * @return Den aktuellen String.
     */
    public String getString ()
          {
          return ((Bans)data.getData()).data;
          }

    /**
     * Methode: getSize.
     * Gibt den Vergrößerungsfaktor, des aktuellen Bans zurück.
     * @return Den aktuellen Vergrößerungsfaktor.
     */
    public int getSize ()
          {
          return ((Bans)data.getData()).size;
          }

    /**
     * Methode: nextElement.
     * Geht in der Liste auf das nächste Element, wenn das aktuelle noch nicht das Letzte ist und liefert true zurück.
     * Wenn das aktuelle Element das letzte ist, liefert die Methode false.
     * @return Gibt true zurück, wenn ein nächstes Element existiert hat.
     */
    public boolean nextElement ()
          {
          if (!data.eol())
              {
              data.gotoNext ();
              return true;
              }
          else return false;
          }

    public boolean eol ()
          {
          return data.eol ();
          }

    /**
     * Methode: getMaxWidth.
     * liefert die Länge (in Pixel) des größten auszugebenden Strings in Abhängigkeit vom Vergrösserungfaktor.
     **/
    public int getMaxWidth ()
          {
          int w = 0;
          for (reset (); !eol (); nextElement ())
              if ((getString().length() * getSize() * 6) > w)
                    w = getString().length () * getSize() * 6;
          return w;
          }

    /**
     * Methode: getMaxHeight.
     * liefert maximale Höhe aller Strings.
     **/
    public int getMaxHeight ()
          {
          int h = 0;
          for (reset (); !eol (); nextElement ())
              h += getSize() * 11;
          return h;
          }
    }
```

2.3.3 MyChars.java

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

/**
 * Klasse: MyChars.
 * Hier werden die Informationen über die einzelnen Zeichen gespeichert & verwaltet.
 */
public class MyChars
    {
    private MagList data;
    Color foreGroundColor, backGroundColor;

    /**
     * Klasse: Chars.
     * In diesem Format werden die einzelen Zeichen aus den *.def-Files abgelegt.
     * Ein Eintrag besteht aus der Bezeichnung des Zeichens:
     *   char name
     * und einer einer Liste mit den Anweisungen zum Zeichnen des Zeichens
     *   MagList chardata.
     */
    private class Chars
          {
          char name;
          MagList chardata;

          Chars (char s, MagList d)
              {
              name = s;
              chardata = d;
              }
          }
```

```
    /**
     * Klasse: LinieKurve
     * In diesem Format werden die einzelnen Anweisungen zum zeichnen von Kurven bzw. Linie gespeichert, wobei
     *    typ == false ... Strich
     *  typ == true .... Kurve
     */
    private class LinieKurve
          {
          boolean typ;
          int abme[];

          LinieKurve (boolean t, int a[])
              {
              typ = t;
              abme = a;
              }
          }

    /**
     * Constructor.
     * läd die "*.def" aus dem angegebenen Verzeichnis
     */
    MyChars (String pfad)
          {
          data = new MagList ();

          // versucht Directory zu öffnen
          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].substring (f[i].length() - 4, f[i].length()).equalsIgnoreCase (".def")) && (f[i].length() == 5))
                    readDefFile (pfad, f[i]);
          }

    /**
     * Methode: readDefFile
     * Wertet die im File gegebenen Zeilen aus und Speichert sie in data
     * @param pfad Gibt das Verzeichnis an, in dem sich das File befindet
     * @param file Gibt das zu ladende File an
     */
    private void readDefFile (String pfad, String file)
          {
          String s = null, sk = null;
          MagList lines = new MagList ();
          boolean error = false;

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

          // Ließt solange Zeilen aus dem File ein bis das Fileende erreicht ist
          while ((s = deffile.readLine()) != null)
              {
              int h[] = new int[5];
              StringTokenizer t = new StringTokenizer (s, " ");
              sk = t.nextToken();

              // Wenn die eingelesene Zeile eine Strichanweisung enthält
              if ((t.countTokens () == 4) && (sk.equals ("Strich")))
                    {
                    for (int i = 0; i < 4; i++)
                        {
                        try {h[i] = Integer.parseInt (t.nextToken ());}
                        catch (NumberFormatException e) {error = true;}
                        if (h[i] > 6) error = true;
                        }
                    if ((h[0] > 4) || (h[2] > 4)) error = true;
                    lines.add (new LinieKurve (false, h));
                    }
              // Wenn die eingelesene Zeile eine Kurvenanweisung enthält
              else if ((t.countTokens () == 5) && (sk.equals ("Kurve")))
                    {
                    for (int i = 0; i < 5; i++)
                        try {h[i] = Integer.parseInt (t.nextToken ());}
                        catch (NumberFormatException e) {error = true;}
                    lines.add (new LinieKurve (true, h));
                    if ((h[0] > 4) || (h[1] > 6) || (h[2] > 3)) error = true;
                    }
              // Wenn die eingelesene Zeile keine gültige Anweisung enthält, Fehlerwert 'true' setzten
              else error = true;
              for (int i = 0; i < 5; i++)
                    if ((h[i] < 0) || (h[i] > 360)) error = true;
              }

          // Gibt Fehlermeldung aus, falls beim Einlesen ein Fehler aufgetreten ist
          // Sonst werden die eingelesenen Daten gespeichert
          if (error || (deffile == null)) System.out.println ("Beim Einlesen des Files \"" + file + "\" ist ein Fehler
aufgetreten");
          else data.add (new Chars (file.charAt (0), lines));
          }
```

```
/**
 * Methode drawMyString.
 * gibt den übergebenen String an der übergebenen XY-Position aus.
 * @param s Der auszugebende String.
 * @param size Der Vergrößerungsfaktor.
 * @param x Die Startposition x.
 * @param y Die Startposition y.
 */
public void drawMyString (Graphics g, String s, int size, int x, int y)
      {
      for (int i = 0; i < s.length (); i++)
          {
          findMyChar (g, s.charAt (i), size, x, y);
          x += size*6;
          }
      }

/**
 * Methode: findMyChar.
 * Sucht den übergebenen Char in der Liste 'data'.
 * Wird kein Char gefunden, wird das Programm ohne Fehlermeldung fortgesetzt.
 * @param c Das auszugebende Zeichen.
 * @param size Der Vergrößerungsfaktor.
 * @param x Die Startposition x.
 * @param y Die Startposition y.
 */
public void findMyChar (Graphics g, char c, int size, int x, int y)
      {
      for (data.reset (); !data.eol (); data.gotoNext ())
          if (((Chars)data.getData ()).name == c)
                drawMyChar (g, (((Chars)data.getData()).chardata), size, x, y);
      }

/**
 * Methode: drawMyChar.
 * Geht die Zeichenanweisungen durch, und zeichnet den Char.
 * @param c Liste mit den Zeichenanweisungen.
 * @param size Der Vergrößerungsfaktor.
 * @param x Die Startposition x.
 * @param y Die Startposition y.
 */
private void drawMyChar (Graphics g, MagList c, int size, int x, int y)
      {
      for (c.reset (); !c.eol (); c.gotoNext ())
          // Wenn true = Kurve, sonst Strich
          if (((LinieKurve)c.getData()).typ)
                drawKurve (g, ((LinieKurve)c.getData()).abme, size, x, y);
          else
                drawStrich (g, ((LinieKurve)c.getData()).abme, size, x, y);
      }

/**
 * Methode: drawKurve.
 * Zeichnet einen Bogen. Dabei wird zuerst ein Bogen in der Vordergrundfarbe, und
 * dann ein zweiter um eine Einheit kleinern Bogen und der Hintergrundfarbe gezeichnet.
 * @param d[] Feld mit den Größenangaben für fillArc().
 * @param size Der Vergrößerungsfaktor.
 * @param x Die Startposition x.
 * @param y Die Startposition y.
 */
private void drawKurve (Graphics g, int d[], int size, int x, int y)
      {
      int r = d[2]*size;

      g.setColor(foreGroundColor);
      g.fillArc(x+d[0]*size-r,y+d[1]*size-r,r*2,r*2,d[3],d[4]);
      g.setColor(backGroundColor);
      r -= size;
      g.fillArc(x+d[0]*size-r,y+d[1]*size-r,r*2,r*2,d[3],d[4]);
      g.setColor(foreGroundColor);
      }

/**
 * Methode: drawStrich.
 * Zeichnet einen Strich, wo bei zwischen Strich nach unten und Strich nach Rechts unterschieden wird.
 * @param d[] Feld mit den Größenangaben für fillPolygon().
 * @param size Der Vergrößerungsfaktor.
 * @param x Die Startposition x.
 * @param y Die Startposition y.
 */
private void drawStrich (Graphics g, int d[], int size, int x, int y)
      {
      int xpoints [] = {0, 0, 0, 0}, ypoints [] = {0, 0, 0, 0};

      if (d[2]-d[0] > d[3]-d[1])
          {
          // Strich nach Rechts
          xpoints[0] = xpoints[3] = d[0]*size + x;
          xpoints[1] = xpoints[2] = (d[2]+1) * size + x;
          ypoints[0] = d[1]*size + y;
          ypoints[1] = d[3]*size+ y;
          ypoints[2] = (d[3]+1)*size + y;
          ypoints[3] = (d[1]+1)*size + y;
          }
```

```
       else
              {
              // Strich nach Unten
              xpoints[0] = d[0]*size + x;
              xpoints[1] = (d[0]+1)*size + x;
              xpoints[3] = d[2]*size + x;
              xpoints[2] = (d[2]+1)*size + x;
              ypoints[0] = ypoints[1] = d[1]*size + y;
              ypoints[2] = ypoints[3] = (d[3]+1)*size + y;
              }

          // Zeichnet das Polygon (den Strich)
          g.fillPolygon (xpoints, ypoints, 4);
          }

    /**
     * Methode: setForeBackColor.
     * Setzt die Vorder- und Hintergrundfarbe. Dieser werden von der Methode drawKurve benötigt.
     * Wenn die Übergabeparameter null sind, werden die Standardwerte (schwarz/weiß) gesetz.
     * @param f Die Vordergrundfarbe.
     * @param b Die Hintergrundfarbe.
     */
    public void setForeBackColor (Color f, Color b)
          {
          if ((foreGroundColor = f) == null) foreGroundColor = Color.black;
          if ((backGroundColor = b) == null) backGroundColor = Color.white;
          }
    }
```

2.3.4 MagList.java

```
/**
 * Klasse: MagList.
 * Speichert und Verwaltet dynamische Datenstrukturen in einer doppelt verketteten Liste.
 */
public class MagList
    {
    // Hilfsnodes, zur Verwaltung und Ansteuerung der Liste
    private Node start, now, end;

    /**
     * Konstruktor.
     * Hier werden die Hilfsnodes zum Verwalten der Daten initialisiert.
     */
    public MagList ()
          {
          now = null;
          start = null;
          end = null;
          }

    /**
     * Methode: add.
     * Fügt ein neues Element in die Liste mit der größe des übergebenen Objektes ein.
     */
    public void add (Object x)
          {
          // Wenn kein Element initialisiert ist: neues Element erzeugen und Hilfsnodes setzen
          if (start == null)
              {
              start = new Node (x, null, null);
              end = now = start;
              }
          // Wenn schon mind. ein Element initialisiert ist: neues Element erzeugen
          else
              {
              Node T = new Node (x, now.next, now);
              now.next = T;
              now = T;
              if (now.next == null)
                    end = T;
              }
          }

    /**
     * Methode: eol.
     * Überprüft, ob das Ende der Liste erreicht wurde.
     * @return true, wenn das Ende der Liste erreicht, sonst false.
     */
    public boolean eol ()
          {
          return now == null;
          }

    /**
     * Methode: gotoNext.
     * Setzt den Hilfspointer now auf das nächste Element in der Liste.
     */
    public void gotoNext ()
          {
          now = now.next;
          }
```

```
    /**
     * Methode: gotoLast.
     * Setzt den Hilfspointer now auf das vorherige Element in der Liste.
     */
    public void gotoLast ()
          {
          now = now.last;
          }

    /**
     * Methode: reset.
     * Setzt den Hilfspointer now zurück auf den Start der Liste.
     */
    public void reset ()
          {
          now = start;
          }

    /**
     * Methode: getData.
     * @return Liefert das gespeicherte Objekt, der aktuellen Listenposition zurück.
     */
    public Object getData ()
          {
          return now.data;
          }

    /**
     * Methode: destroy.
     * Setzt alle Hilfnodes auf null zurück -> Löschen der Liste.
     */
    public void destroy ()
          {
          start = null;
          now = null;
          end = null;
          }

    /**
     * Klasse: Node.
     * In diesem Format werden die einzelnen Elemente der Liste abgelegt.
     */
    private class Node
          {
          Object data;
          Node next;
          Node last;

          Node (Object d, Node n, Node l)
              {
              data = d;
              next = n;
              last = l;
              }
          }
    }
```