Übungsaufgabe 3
Übung IT/EP 188.156

Bernhard Fiser
0109815

Beschreibung
Das Programm 'Banner' liest Zeichendefinititonen aus einzelnen Dateien für jedes Zeichen von 'A' bis 'Z' aus und stellt anschließen einen Text in der Schriftart der Zeichendefinititonen dar.

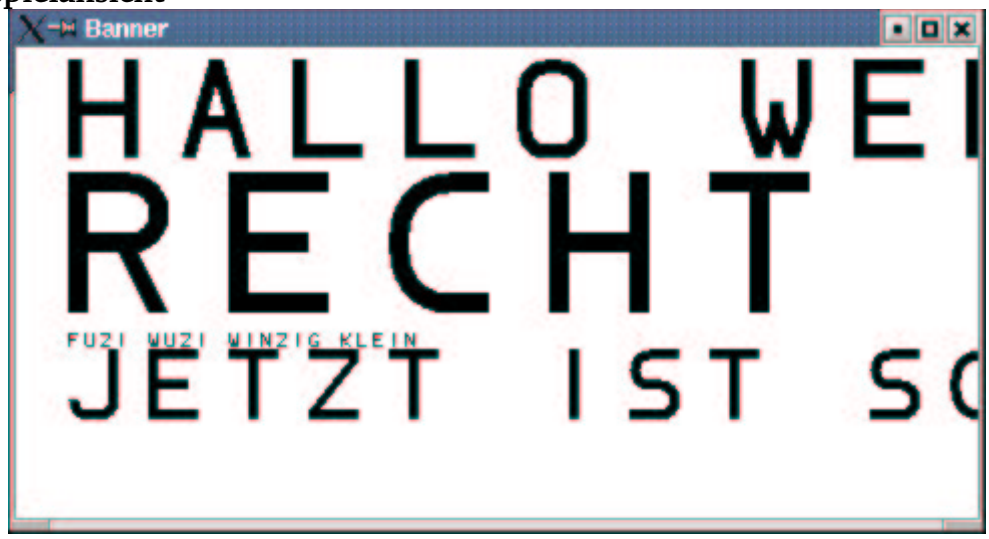
Das Problem wurde gelöst, indem für jedes einzelne Zeichen eine LinkedList angelegt wird, wobei jedes Element der LinkedList ein einzelnes Zeichenelement darstellt. Die so entstandenen Listen werden anschließend in einer Hashmap zum schnellen auffinden der Buchstaben abgespeichert.

Ein einzelnes Element kann vom Typ 'Strich' oder 'Kurve' sein, und stellt dementsprechend funktional unterschiedliche Methoden zum Zeichnen zur Verfügung. Die beiden Typen sind von einer abstrakten Klasse erweitert (extended), um die gleichen Methoden zum Zeichen zur Verfügung stellen zu können.

Die Bannertexte selbst werden ebenfalls in eine LinkedList aufgenommen, wobei eine eigene Klasse für einen einzelnen Bannertext verwendet wird. Diese Klasse beinhaltet nur den Text und den zugehörigen Vergrößerungsfaktor des Textes.

Beim öffnen des Fenster wird nun jeder einzelne Text aus der Bannertextliste genommen, danach jeder Buchstabe aus dem Text einzeln herausgenommen und dessen Zeichenmethode aufgerufen.

Beispielansicht



Pseudocode

- START (Übergabeparameter: <Verzeichnis> <bannertexte>)
- Schleife von Zeichen 'A' bis 'Z'
- Zeichendefinititonsdatei öffnen
- neues Zeichenelement erstellen
- Schleife über alle Zeilen der Datei
- Zeile lesen und parsen
- In Zeichenelement aufnehmen
- Bei Fehler Programm beenden
- Zeichendefinitionsdatei schließen
- Zeichen in Hashmap aufnehmen
- Bannerdatei öffnen
- Schleife über alle Zeilen der Datei
- Bannerelement erstellen
- Zeile lesen und parsen
- Bannerinformation in Bannerelement aufnehmen
- Bei Fehler Programm beenden
- Bannerelement in Bannertextliste aufnehmen
- Bannerdateu schließen
- Fenster öffnen
- Aufruf der Zeichenmethode durch das OS
- Y−Startposition festlegen
- Schleife über alle Bannertexte
- Bannertext und Vergrößerungsfaktor auslesen
- X−Startposition festlegen
- Schleife über alle Zeichen des Textes
- Zeichen auslesen
- Zeichendefinitionselemente aus Hashmap auslesen
- Schleife über alle Zeichendefinitionselemente
- Element zeichnen
- X−Position erhöhen
- Y−Position erhöhen
- ENDE (warten bis Fenster gesclossen wird)

Sourcecode Banner.java

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

/** Hauptprogramm, um Zeichdaten aus Files einzulesen,
  * und in einem Fenster graphisch wieder auszugeben.
  *
  * @author Bernhard Fiser 0109815
  */
public class Banner
{
	/* Hashmap zur Speicherung der Zeichendaten */
	static HashMap		hmap;
	/* Liste zur Speicherung der Bannertexte */
	static LinkedList	belem;

	/** Hauptprogramm */
	public static void main(String argv[])
	{
		char		c;
		CharDef			cd;
		CharElement	ce;
		ReadFile	f;
		String		line;
		int		i, e;
		StringTokenizer st;
		String		t;
		BannerElement	be;


		/* Falls nicht genügend Parameter übergeben wurden,
		 * Fehler ausgeben und Programm beenden. */
		if (argv.length < 2)
		{
			System.err.println("*** usage: java Banner <data-directory>
<textfile>");
			System.exit(1);
		}

		/* Hashmap initialisieren */
		hmap = new HashMap();
		/* Schleife über alle Großbuchstaben von A bis Z */
		for (c = 'A'; c <= 'Z'; c++)
		{
			/* Information auf stdout ausgeben */
			System.out.print("info: reading character " + c + " ... ");

			/* Zeilenzähler initialisieren*/
			i = 0;
			/* Zeichendatenobjekt initialisieren mit entsprechendem
			 * Zeichen c */
			cd = new CharDef(c);
			/* File des Zeichens öffnen */
			f = new ReadFile(argv[0], c + ".def");
			/* Zeilen der Datei einzeln auslesen */
			while ((line = f.readLine()) != null)
			{
				/* Zeilenzähler erhöhen */
				i++;
				/* Stringtokenizer initialisieren, Trennzeichen ' ' */
				st = new StringTokenizer(line, " ");
				/* Fehlercodevariable auf 0 setzen */
				e = 0;
				/* Falls zumindes ein Element existiert ... */
				if (st.hasMoreTokens())
				{
					/* ... erstes Element lesen */
					t = st.nextToken();
					/* Fehlercode je nach erstem Element auf einen
Wert setzen */
					if (t.compareToIgnoreCase("strich") == 0)
						e = 1;
					else if (t.compareToIgnoreCase("kurve") == 0)
						e = 2;
					else
						e = -1;
				}

				ce = null;
				switch (e)
				{
					/* Falls Fehlercode e == 1, dann Linienelement
initialisieren */
					case 1 :
						ce = new CharLineElement();
						break;
```

```
                                        /* Falls Fehlercode e == 2, dann Kurvenelement
initialisieren */
                                        case 2 :
                                                ce = new CharCurveElement();
                                                break;

                                        /* Alle anderen Fehlercodes sind unzulässig,
geben eine Fehlermeldung aus
                                         * und beenden das Programm */
                                        default :
                                                System.err.println("*** error: syntax
error in file " + c + ".def line " + i);
                                                System.exit(1);
                                }

                                /* aus Datei gelesener String in Zeichenelement
übernehmen */
                                if (!ce.parseLine(line))
                                {
                                        /* Falls übernahme nicht möglich, dann
Fehlermeldung ausgeben und
                                         * Programm beenden */
                                        System.err.println("*** parse error: syntax error
in file " + c + ".def line " + i);
                                        System.exit(1);
                                }
                                /* andernfalls dieses Zeichenelement dem Zeichen
hinzufügen, d.h. alles OK */
                                cd.add(ce);
                        }
                        /* Alle Zeilen des Zeichens gelesen, Datei schließen */
                        f.close();
                        /* Zeichen in Hashmap aufnehmen */
                        hmap.put(cd.getCharObject(), cd);
                        /* Erfolgsmeldung auf Bildschirm ausgeben */
                        System.out.println("done");
                }

                /* LinkedList erstellen, zur Ablage der Bannertexte */
                belem = new LinkedList();

                System.out.print("reading textfile " + argv[1] + " ... ");
                /* File öffnen und Zeilenzähler auf 0 setzen */
                i = 0;
                f = new ReadFile(argv[0], argv[1]);
                /* Zeilen einzeln auslesen */
                while ((line = f.readLine()) != null)
                {
                        /* Zeilenzähler erhöhen */
                        i++;
                        /* Neues BannerElement erstellen */
                        be = new BannerElement();
                        /* Gelesene Zeile parsen und in Bannerelement übernehmen */
                        if (!be.parseLine(line))
                        {
                                /* Falls nicht erfolgreich Fehlermeldung ausgeben und
Programm beenden */
                                System.err.println("*** error: syntax error in file " +
argv[1] + " line " + i);
                                System.exit(1);
                        }
                        /* Alles OK, Bannerelement in Liste aufnehmen */
                        belem.add(be);
                }
                /* Datei schließen und Erfolgsmeldung ausgeben */
                f.close();
                System.out.println("done");

                /* Fenster öffnen und darstellen */
                new Wnd(hmap, belem);
                }
}
```

Sourcecode BannerElement.java

```
import java.util.*;

/** Klasse zur Speicherung der Daten eines Bannertextes
  *
  * @author Bernhard Fiser 0109815
  */
public class BannerElement
{
	private int    zoom;
	private String text;

	/** Initialisierung des Bannerelements */
	public BannerElement()
	{
		this.zoom = 1;
		this.text = "";
	}

	/** Zoomfaktor zurückliefern
	  * @return int zoom : Zoomfaktor */
	public int getZoom()
	{
		return(this.zoom);
	}

	/** Text des Bannerelements zurückliefern
	  * @return String txt : Bannertext */
	public String getText()
	{
		return(this.text);
	}

	/** Methode um einen String in ein Bannerelement zu
	  * konvertieren.
	  * @return false : falls ein Fehler bei der Konvertierung aufgetreten ist
	  * @return true : falls Konvertierung erfolgreich */
	public boolean parseLine(String s)
	{
		StringTokenizer st;
		int             i;
		String          t;

		/* Stringtokenizer initialisieren */
		i = 0;
		st = new StringTokenizer(s, " ");
		while(st.hasMoreTokens())
		{
			t = st.nextToken();
			switch(i)
			{
				/* Erstes Element enthält den Zoomfaktor */
				case 0 :
					/* falls Integerkovertierung nicht möglich,
					 * Fehler zurückgeben */
					try { this.zoom = Integer.parseInt(t); }
					catch (NumberFormatException e) { return(false);
}
					break;

				/* Die restlichen Element wieder verketten */
				default :
					this.text += t + " ";
			}
			/* Elementzähler erhöhen */
			i++;
		}
		/* Sollten weniger als 2 Elemente im String gewesen sein,
		 * dann Fehler zurückgeben */
		if (i < 2)
			return(false);
		/* Andernfalls alles OK */
		return(true);
	}
}
```

Sourcecode CharCurveElement.java

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

/** Klasse zur Speicherung eines Kurvenelements
  * eines Zeichens.
  *
  * @author Bernhard Fiser 0109815
  */
public class CharCurveElement extends CharElement
{
	private int	coord[];

	/** Initialisierung des Kurvenelements */
	public CharCurveElement()
	{
		int	i;

		/* Koordinatenfeld initialisieren und auf 0 setzen */
		coord = new int[5];
		for (i = 0; i < 5; i++)
			coord[i] = 0;
	}

	/** Daten aus dem String s in das Koordinationfeld übernehmen */
	public boolean parseLine(String s)
	{
		StringTokenizer	st;
		int		i;
		String		t;

		/* StringTokenizer inititalisieren,
		 * Trennzeichen ' ' */
		i = 0;
		st = new StringTokenizer(s, " ");
		while (st.hasMoreTokens())
		{
			t = st.nextToken();
			switch (i)
			{
				/* Das erste Element bestimmt den Typ.
				 * Falls nicht vom Type 'kurve', dann Fehler
				 * zurückgeben */
				case 0 :
					if (t.compareToIgnoreCase("kurve") != 0)
						return(false);
					break;

				/* Die nächsten 5 Elemente bestimmen die Koordinaten
				 * x, y, Radius, Anfangswinkel und Bogenwinkel. */
				case 1 :
				case 2 :
				case 3 :
				case 4 :
				case 5 :
					/* In Integer umkonvertieren, und Fehler
zurückgeben
					 * falls Konvertierung nicht möglich */
					try { this.coord[i - 1] = Integer.parseInt(t); }
					catch (NumberFormatException e) { return(false);
}
					/* Bei einer negativen Koordinate ebenfalls einen
					 * Fehler zurückgeben */
					if (this.coord[i - 1] < 0)
						return(false);
					break;

				/* Sollten mehr als diese 6 Elemente existieren,
				 * dann Fehler zurückgeben */
				default :
					return(false);
			}
			/* Element- (Token-) zähler erhöhen */
			i++;
		}
		/* Sollten weniger als 6 Elemente (Tokens) im String enthalten sein,
		 * dann Fehler zurückgeben */
		if (i < 6)
			return(false);
		/* Andernfalls alles OK */
		return(true);
	}

	/** Dieses Element graphisch darstellen */
	public void draw(int x, int y, int zoom, Graphics g)
	{
		int	xs, ys, w;
```

```
                /* Startkoordinaten des Rechtecks in dem sich der Bogen befindet
                 * berechnen. Siehe Angabe und JDK Dokumentation 'fillArc' */
                xs = (this.coord[0] - this.coord[2]) * zoom + x;
                ys = (this.coord[1] - this.coord[2]) * zoom + y;
                w = this.coord[2] * 2 * zoom;

                /* Bogen zeichnen */
                g.setColor(Color.black);
                g.fillArc(xs, ys, w, w, this.coord[3], this.coord[4]);

                /* Rechteck korrigieren, innen darf es nicht gefüllt sein, d.h.
                 * dieser Bereich wird nochmals weiß übermalt */
                xs += zoom;
                ys += zoom;
                w -= zoom * 2;

                /* inneren Bogen zeichen */
                g.setColor(Color.white);
                g.fillArc(xs, ys, w, w, this.coord[3], this.coord[4]);
        }

}
```

Sourcecode CharDef.java

```
import java.util.*;

/** Klasse zur Speicherung aller Elemente
  * eines Zeichens
  *
  * @author Bernhard Fiser 0109815
  */

public class CharDef extends LinkedList
{
        private Character      c;

        /** Erzeugt ein neues Element und initialisiert
          * das Zeichen des Elements */
        public CharDef(char c)
        {
                this.c = new Character(c);
        }

        /** Liefert das Zeichen dieses Elements zurück
          */
        public Character getCharObject()
        {
                return(this.c);
        }
}
```

Sourcecode CharElement.java

```
import java.awt.*;

/** Abstrakte Klasse zur Speicherung
  * eines graphischen Elements eines Zeichens
  *
  * @author Bernhard Fiser 0109815
  */
public abstract class CharElement
{
        /** Defniert die Methode zur Datenübernahme in das Element */
        abstract boolean parseLine(String s);

        /** Definiert die Methode um das Element graphisch darzustellen */
        abstract void draw(int x, int y, int zoom, Graphics g);
}
```

Sourcecode CharLineElement.java

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

/** Klasse zur Speicherung eines Linienelements
  * eines Zeichens.
  *
  * @author Bernhard Fiser 0109815
  */
public class CharLineElement extends CharElement
{
	private int	coord[];

	/** Initialisiert das LinienElement */
	public CharLineElement()
	{
		int	i;

		/* Koordinationfeld erzeugen und auf 0 setzen */
		coord = new int[4];
		for (i = 0; i < 4; i++)
			coord[i] = 0;
	}

	/** Daten aus dem String s in das Koordinationfeld übernehmen */
	public boolean parseLine(String s)
	{
		StringTokenizer	st;
		int		i;
		String		t;

		/* StringTokenizer inititalisieren,
		 * Trennzeichen ' ' */
		i = 0;
		st = new StringTokenizer(s, " ");
		while (st.hasMoreTokens())
		{
			t = st.nextToken();
			switch (i)
			{
				/* Das erste Element bestimmt den Typ.
				 * Falls nicht vom Type 'strich', dann Fehler
				 * zurückgeben */
				case 0 :
					if (t.compareToIgnoreCase("strich") != 0)
						return(false);
					break;
				/* Die nächsten 4 Elemente bestimmen die Koordinaten
				 * x1, y1, x2 und y2. */
				case 1 :
				case 2 :
				case 3 :
				case 4 :
					/* In Integer umkonvertieren, und Fehler
zurückgeben
					 * falls Konvertierung nicht möglich */
					try { this.coord[i - 1] = Integer.parseInt(t); }
					catch (NumberFormatException e) { return(false);
}
					/* Bei einer negativen Koordinate ebenfalls einen
					 * Fehler zurückgeben */
					if (this.coord[i - 1] < 0)
						return(false);
					break;
				/* Sollten mehr als diese 5 Elemente existieren,
				 * dann Fehler zurückgeben */
				default :
					return(false);
			}
			/* Element- (Token-) zähler erhöhen */
			i++;
		}
		/* Sollten weniger als 5 Elemente (Tokens) im String enthalten sein,
		 * dann Fehler zurückgeben */
		if (i < 5)
			return(false);
		/* Andernfalls alles OK */
		return(true);
	}

	/** Dieses Element graphisch darstellen */
	public void draw(int x, int y, int zoom, Graphics g)
	{
		/* Koordinatenfelder für die Funktion fillPolygon */
		int xc[] = new int[5];
		int yc[] = new int[5];
		int i;
```

```
                /* Winkel steiler als 45° */
                if (this.coord[2] - this.coord[0] < this.coord[3] - this.coord[1])
                {
                        /* Berechnung der Koordinatenpunkte. Details siehe Übungsangabe
*/
                        xc[0] = (this.coord[0] + 1) * zoom;
                        yc[0] = this.coord[1] * zoom;

                        xc[1] = (this.coord[2] + 1) * zoom;
                        yc[1] = (this.coord[3] + 1) * zoom;

                        xc[2] = this.coord[2] * zoom;
                        yc[2] = yc[1];

                        xc[3] = this.coord[0] * zoom;
                        yc[3] = yc[0];
                }
                /* Winkel flacher als 45° */
                else
                {
                        /* Berechnung der Koordinatenpunkte. Details siehe Übungsangabe
*/
                        xc[0] = this.coord[0] * zoom;
                        yc[0] = this.coord[1] * zoom;

                        xc[1] = (this.coord[2] + 1) * zoom;
                        yc[1] = this.coord[3] * zoom;

                        xc[2] = xc[1];
                        yc[2] = (this.coord[3] + 1) * zoom;

                        xc[3] = xc[0];
                        yc[3] = (this.coord[1] + 1) * zoom;
                }

                /* Endpunkt auf Anfangspunkt setzen */
                xc[4] = xc[0];
                yc[4] = yc[0];

                /* Offset hinzufuegen (absolute Position des Elements auf dem
Bildschirm) */
                for (i = 0; i < 5; i++)
                {
                        xc[i] += x;
                        yc[i] += y;
                }

                /* Zeichnen */
                g.setColor(Color.black);
                g.fillPolygon(xc, yc, xc.length);
        }
}
```

Sourcecode Wnd.java

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

/** Klasse zum darstellen der Bannertexte und Programm beenden beim Fensterschließen
  *
  * @author Bernhard Fiser 0109815
  */
public class Wnd extends Frame
{
	/* Hashmap zur Speicherung der Zeicheninformationen */
	private HashMap		hmap;
	/* LinkedList zur Speicherung der Bannertexte */
	private LinkedList	tlist;

	/** Fenster öffnen und Instanzvariablen initialisieren */
	public Wnd(HashMap h, LinkedList l)
	{
		/* Fenster erstellen und Titeltext setzen */
		super("Banner");
		/* Instanzvariablen setzen */
		this.hmap = h;
		this.tlist = l;
		/* Handler für das Fensterschließen setzen */
		addWindowListener(new WndClose());
		/* Fenstergröße und -farbe setzen */
		setSize(500, 500);
		setBackground(Color.white);
		/* Fenster anzeigen */
		show();
	}

	/** "Fensterschließ-Handler"
	  * Dieser wird aufgerufen falls das Fenster geschlossen wird */
	class WndClose extends WindowAdapter
	{
		public void windowClosing(WindowEvent e)
		{
			/* Meldung ausgeben und Programm beenden */
			System.out.println("Window closed and exit");
			System.exit(0);
		}
	}

	/** Grafik zeichnen.
	  * Diese Methode wird aufgerufen, falls der Inhalt des Fensters
	  * neu gezeichnet werden muß */
	public void paint(Graphics g)
	{
		int		i,
				xoff,
				yoff;
		char		c;
		CharElement	ce;
		CharDef			cd;
		ListIterator	li, tli;
		String		txt;
		int		zoom;
		BannerElement	be;

		/* Listeniterator der Bannertexte initialisieren */
		tli = tlist.listIterator(0);
		/* Absolute Y-Startposition setzen */
		yoff = 50;
		/* Schleife über alle Bannertexte aus der Liste */
		while (tli.hasNext())
		{
			/* Bannerelement aus Liste auslesen */
			be = (BannerElement) tli.next();
			/* Bannertext und Zoomfaktor auslesen */
			txt = be.getText();
			zoom = be.getZoom();
			/* Absolute X-Startposition setzen */
			xoff = 30;
			/* Schleife über alle Zeichen des Bannertextes */
			for (i = 0; i < txt.length(); i++)
			{
				/* Zeichenelementliste aus Hashmap auslesen */
				if ((cd = (CharDef) hmap.get(new
Character(txt.charAt(i)))) != null)
				{
					/* Falls Zeichen in Hashmap exisistiert, Iterator
					 * über die ELementliste erstellen */
					li = cd.listIterator(0);
					/* Schleife über alle Elemente des Zeichens */
					while(li.hasNext())
```

```
                                        {
                                                /* Zeichenelement aus Liste auslesen */
                                                ce = (CharElement) li.next();
                                                /* Zeichenelement zeichnen */
                                                ce.draw(xoff, yoff, zoom, g);
                                        }
                                }
                                /* Absolute X-Position erhöhen für nächstes Zeichen */
                                xoff += 8 * zoom;
                        }
                        /* Absolute Y-Position erhöhen für nächsten Bannertext */
                        yoff += 8 * zoom;
                }
        }
}
```

Sourcecode ReadFile.java

```
import java.io.*;

public class ReadFile
{
	private BufferedReader		in;
	private boolean			fehler;

	public ReadFile(String path, String name) {
		try {
			in = new BufferedReader(new FileReader(new File(path, name)));
			fehler = false;
			}
		catch (IOException e) {
			fehler = true;
			System.out.println("Fehler beim Öffnen: " + e);
			}
		}

	public String readLine() {
		String s = null;
		if (fehler)
			return null;
		try {
			s = in.readLine();
			}
		catch (IOException e) {
			System.out.println("Fehler beim Lesen: " + e);
			}
		return s;
		}

	public void close() {
		try {
			fehler = true;
			in.close();
			}
		catch (IOException e) {
			System.out.println("Fehler beim Schließen: " + e);
			}
		}

	/* Testprogramm */
	public static void main(String args[]) {
		String path = "/anydir/ReadFile";
		ReadFile file = new ReadFile(path, "test.txt");

		String s;
		while (null != (s = file.readLine()))
			System.out.println(s);

		file.close();
		}
	}
```