RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET
INFORMATIK I
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Dr. Jürgen Giesl
Darius Dlugosz, Thomas v.d. Maßen, Antje Nowack

Übung Informatik I – Programmierung – Blatt 11

(Lösungsvorschlag)

Hinweis: Die Quelltexte sind hier nur auszugsweise aufgeführt. Die wichtigsten Änderungen und die „Hot Spots“ sollten aber deutlich werden. Die vollständigen Quelltexte sind dann in den Java-Quelltextdateien zu finden.

Aufgabe 1)

```
package figurenanzeige.figuren;

/* Die Klasse realisiert ein geometrisches Objekt und stellt spezifische Attribute
 * und Operationen (Methoden) für solch ein Objekt bereit.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 14.01.2002
 */

import figurenanzeige.IO;

public abstract class GeoObjekt {
      protected double flaecheninhalt;
      protected double umfang;
      protected String linienfarbe;
      protected String fuellfarbe;
      protected int x;
      protected int y;

      /**
       * Standardkonstruktor, der alle Attribute initialisiert.
       */
      public GeoObjekt() {
            flaecheninhalt = 0;
            umfang = 0;
            linienfarbe = "";
            fuellfarbe = "";
            x = 0;
            y = 0;
      }

      /**
       * Konstruktor der das Objekt mit einer x- und y-Position initialisiert
       * @param x x-Koordinate des Objekts
       * @param y y-Koordinate des Objekts
       */
      public GeoObjekt(int x, int y) {
            flaecheninhalt = 0;
            umfang = 0;
            linienfarbe = "";
            fuellfarbe = "";
            this.x = x;
            this.y = y;
      }
```

```
	/**
	 * Die Methode <i>setPosition</i> legt die Koordinaten des Objekts fest.
	 * @param x x-Koordinate des Objekts
	 * @param y y-Koordinate des Objekts
	 */
	public void setPosition(int x, int y) {
		this.x = x;
		this.y = y;
	}

	public int getX() {
		return x;
	}

	public int getY() {
		return y;
	}

	/**
	 * Die Methode <i>berechneFlaecheninhalt</i> ist abstrakt definiert, da noch zu
wenige Informationen bekannt sind, um diesen zu berechnen.
	 * @return Flaecheninhalt
	 */
	public abstract double berechneFlaecheninhalt();

	/**
	 * Die Methode <i>berechneUmfang</i> ist abstrakt definiert, da noch zu wenige
Informationen bekannt sind, um diesen zu berechnen.
	 * @return Umfang
	 */
	public abstract double berechneUmfang();

	/**
	 * Die Methode <i>setLinienfarbe</i> legt die Linienfarbe des Objekts fest.
	 * @param farbe Linienfarbe
	 */
	public void setLinienfarbe(String farbe) {
		linienfarbe = farbe;
	}

	/**
	 * Die Methode <i>getLinienfarbe</i> liefert die Linienfarbe
	 * des geometrischen Objekts zurück.
	 * @return Linienfarbe
	 */
	public String getLinienfarbe() {
		return linienfarbe;
	}

	/**
	 * Die Methode <i>setFuellfarbe</i> legt die Fuellfarbe des Objekts fest.
	 * @param farbe Füllfarbe
	 */
	public void setFuellfarbe(String farbe) {
		fuellfarbe = farbe;
	}

	/**
	 * Die Methode <i>getFuellfarbe</i> liefert die Fuellfarbe
	 * des geometrischen Objekts zurück.
	 * @return Fuellfarbe
	 */
	public String getFuellfarbe() {
		return fuellfarbe;
```

```
	}


	/**
	 * Die Methode <i>aenderung</i> liest die Attribute
	 * des Geometrischen Objekts durch den Benutzer ein
	 */
	public void aenderung() {
		System.out.print("Geben Sie bitte die x-Position ein: ");
		int xpos = IO.Eingabe();
		System.out.print("Geben Sie bitte die y-Position ein: ");
		int ypos = IO.Eingabe();
		setPosition(xpos, ypos);
		System.out.print("Geben Sie bitte die Linienfarbe ein: ");
		String farbe = IO.TextEingabe();
		setLinienfarbe(farbe);
		System.out.print("Geben Sie bitte die Fuellfarbe ein: ");
		farbe = IO.TextEingabe();
		setFuellfarbe(farbe);
	}
}
```

```
package figurenanzeige;

/* Diese Klasse stellt ein Menu bereit, welches dem Benutzer erlaubt verschiedene
 * graphische Figuren anzulegen und in einem Array zu speichern. Diese Figuren werden
 * dann graphisch auf dem Bildschirm in einem Fenster dargestellt.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 14.01.2002
 */

import figurenanzeige.figuren.*;
import figurenanzeige.gui.Anzeige;

public class Figurendarstellung {

	private static int figurenanzahl = 0;
	private static GeoObjekt[] figuren = new GeoObjekt[10];

	public static void druckeMenu() {
		System.out.println("*** Figurendarstellung ***");
		System.out.println("1) Rechteck einfuegen");
		System.out.println("2) Quadrat einfuegen");
		System.out.println("3) Ellipse einfuegen");
		System.out.println("4) Kreis einfuegen");
		System.out.println("-------------------------");
		System.out.println("5) Zeichnen");
	}

	public static void figurHinzufuegen(GeoObjekt g) {
		if (figurenanzahl < figuren.length) {
			figuren[figurenanzahl] = g;
			figurenanzahl = figurenanzahl + 1;
		}
	}


	public static void main (String[] args) {

		/* Menüauswahl */
```

```
            int auswahl = 0;
            /* Hilfsvariablen */
            String dummy;


            for (int i = 0; i < figuren.length; i++) {
                  figuren[i] = null;
            }


            /* Zeige das Menue solange, bis das Programm beendet wird */
            while (auswahl != 5) {
                  druckeMenu();
                  auswahl = IO.Eingabe();
                  switch (auswahl) {
                        case 1: { Rechteck r = new Rechteck();
                                    r.aenderung();
                                    figurHinzufuegen(r);
                                    } break;
                        case 2: { Quadrat q = new Quadrat();
                                    q.aenderung();
                                    figurHinzufuegen(q);
                                  } break;
                        case 3: { Ellipse e = new Ellipse();
                                    e.aenderung();
                                    figurHinzufuegen(e);
                                  } break;
                        case 4: { Kreis k = new Kreis();
                                    k.aenderung();
                                    figurHinzufuegen(k);
                                  } break;
                        case 5: { Anzeige a = new Anzeige(figuren); } break;
                        default: { /* Default-Fall, wenn eine ungültige Auswahl
getroffen wurde */
                                    System.out.println("Ungueltige Auswahl!");    }
                  }
            }
      }
}
```

Aufgabe 2)

Generell sollte Software modular aufgebaut sein, damit sie aus einzelnen Teilen zusammengesetzt werden kann. Diese einzelnen Module können von verschiedenen Personen entworfen, realisiert, getestet und gewartet werden. Die zugrunde liegenden Implementierungen können leicht ersetzt werden und können somit auch in unterschiedlichen Zusammenhängen, sprich in anderen Programmen wiederverwendet werden. Gerade beim Test von Softwarekomponenten ergeben sich große Vorteile für die Modularisierung. Somit können einzelne Bausteine separat getestet und gewartet werden.
Zudem sollten in einem Paket logisch-zusammenhängende Bausteine eines Programms zusammengefasst werden. Somit kann man unterschiedliche Verantwortlichkeiten sauber trennen. Beispielsweise die grundlegende Funktionalität eines Systems und ihre Darstellung. Klar-definierte Schnittstellen machen es möglich, dass Module von fremden Personen oder Anbietern benutzt oder auch erweitert werden können.

Der direkter Zugriff auf Attribute von Klassen in unterschiedlichen Paketen ist nur dann möglich, falls das Attribut mit dem „public“-Modifizierer deklariert wurde. Ein Zugriff auf ein als „protected“ deklariertes Attribut ist nur dann möglich falls die aufrufende Klasse eine Unterklasse der Klasse ist, welche das Attribut definiert.

Zudem ist die entsprechende Klasse, die das Attribut definiert, mittels der import-Anweisung zu importieren oder der Paketname voranzustellen.

Wird eine strenge Kapselung der internen Informationen einer Klasse angestrebt (dies sollte immer der Fall sein), so sind die Attribute als „private“ zu deklarieren. Um die Attributwerte zu lesen und zu verändern sind somit geeignete Selektoren zu definieren.

Aufgabe 3)

```
package kontoverwaltung.konto;

import kontoverwaltung.liste.*;

public class Kontoliste {

	private Liste kliste;

	public Kontoliste() {
		kliste = new Liste();
	}

	public void fuegeVorneEin(Konto k) {
		kliste.fuegeVorneEin(k);
	}

	public Element suche(Konto k) {
		return kliste.suche(k);
	}

	public void drucke() {
		kliste.drucke();
	}

	public Konto kontoAnPosition(int i) {
		Element e = kliste.getKopf();
		for (int j = 1; j < i; j++) {
			e = e.getNext();
		}
		return (Konto)e.getWert();
	}
}
```

```
package kontoverwaltung.konto;

/* Die Klasse realisiert ein einfach-verkette Liste um Buchungselemente
 * zu speichern. Ebenso werden Operationen (Methoden) für die Arbeit auf
 * dieser Liste zur Verfügung gestellt. Einige Operationen werden an
 * die allgemeine Liste delegiert.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 14.01.2002
 */

import kontoverwaltung.liste.*;

public class Buchungsliste {

	// Enthält die allgemeine Liste
	private Liste bliste;
```

```
	private int anzahlBuchungen;

	/* Standardkonstrukor */
	public Buchungsliste() {
		bliste = new Liste();
		anzahlBuchungen = 0;
	}

	/* Fügt eine Buchung in die Liste ein.
	 * @param Buchung
	 */
	public void fuegeSortiertEin(Buchung b) {
		// Delegiere an allgemeine Liste
		bliste.fuegeSortiertEin(b);
		anzahlBuchungen = anzahlBuchungen + 1;
	}

	/* Such eine Buchung und liefert das Listenelement zurück
	 * @param Buchung
	 * @return Listenelement, welches die Buchung enthält (oder null)
	 */
	public Element suche(Buchung b) {
		return bliste.suche(b);
	}

	/* Liefert die Anzahl der gespeicherten Buchungen in der Liste
	 * @return Anzahl der Buchungen in der Liste
	 */
	public int getAnzahlBuchungen() {
		return anzahlBuchungen;
	}

	/* Berechnet den Saldo mittels einer Hilfsmethode
	 * @return Saldo
	 */
	public double berechneSaldo() {
		double saldo = 0.0;
		Element e = bliste.getKopf();
		while (e != null) {
			Buchung b = (Buchung)e.getWert();
			saldo = saldo + b.getBetrag();
			e = e.getNext();
		}
		return saldo;
	}

	/*	Gibt die gespeicherten Buchungen von vorne nach hinten mittels einer
Hilfsmethode aus */
	public void drucke() {
		bliste.drucke();
	}
}
```

```
package kontoverwaltung.konto;
/* Die Klasse realisiert eine Buchung und stellt spezifische Attribute
 * und Operationen (Methoden) für eine Buchung bereit.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 14.01.2002
 */

import kontoverwaltung.liste.Vergleichbar;

public class Buchung implements Vergleichbar {

	private String beschreibung = "";
	private double betrag = 0.0;
	private Datum datum = new Datum();

	/* Setzt die Beschreibung der Buchung auf den String b */
	public void setBeschreibung(String b) {
		beschreibung = b;
	}

	/* Liefert die Beschreibung der Buchung zurück */
	public String getBeschreibung() {
		return beschreibung;
	}

	/* Setzt den Betrag der Buchung auf den übergebenen Wert b */
	public void setBetrag(double b) {
		betrag = b;
	}

	/* Liefert den Betrag der Buchung zurück */
	public double getBetrag() {
		return betrag;
	}

	/* Setzt das Datum der Buchung auf das Datum d */
	public void setDatum(Datum d) {
		datum = d;
	}

	/* Liefert das Datum der Buchung zurück */
	public Datum getDatum() {
		return datum;
	}

	public String toString() {
		return ((this.getDatum()).toString() + " " + this.getBeschreibung() + " " +
this.getBetrag() + "\n");
	}

	/* Prüft zwei Buchungen auf ihre Inhaltsgleichheit */
	public boolean gleich(Vergleichbar zuvergleichen) {
		if (zuvergleichen instanceof Buchung) {
			Buchung b = (Buchung)zuvergleichen;
			// Vergleiche Beschreibung, Betrag und Datum
			if ((this.beschreibung.equals(b.beschreibung)) &&
				(this.betrag == b.betrag) &&
				(this.datum.gleich(b.datum))) {
					return true;
			}
			else {
				return false;
```

```
                }
            }
            else {
                System.out.println("Kein Buchungsvergleich!");
                return false;
            }
        }

        /* Prüft ob das aktuelle Buchungsobjekt zeitlich vor dem übergebenen
Buchungsobjekt liegt */
        public boolean kleiner(Vergleichbar zuvergleichen) {
            if (zuvergleichen instanceof Buchung) {
                Buchung b = (Buchung)zuvergleichen;
                // Prüft, das Datum der beiden Buchungen
                if (this.datum.kleiner(b.datum)) {
                    return true;
                }
                else {
                    return false;
                }
            }
            else {
                System.out.println("Kein Buchungsvergleich!");
                return false;
            }
        }
}
```