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 9

(Lösungsvorschlag)

Aufgabe 1)

```
/* Die Klasse realisiert ein Konto und stellt spezifische Attribute
 * und Operationen (Methoden) für ein Konto bereit bereit.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 18.12.2001
 */

public class Konto {

	private final int maxBuchungen = 100;

	private String inhaber;
	private String kontonr;
	private String blz;

	private Buchungsliste buchungen = new Buchungsliste();

	/* Speicherung der Anzahl der instantiierten Konten */
	private static int anzKonten = 0;

	public Konto() {
		init();
		anzKonten = anzKonten + 1;
	}

	public Konto(String inhaber, String kontonr, String blz) {
		this.inhaber = inhaber;
		this.kontonr = kontonr;
		this.blz = blz;
		anzKonten = anzKonten + 1;
	}

	/* Initialisierung eines neuen Kontos */
	public void init() {
		inhaber = "";
		kontonr = "";
		blz = "";
	}

	/* Setzt den Inhaber auf den übergebenen String i */
	public void setInhaber(String i) {
		inhaber = i;
	}

	/* Liefert den Inhaber des Kontos zurück */
	public String getInhaber() {
		return inhaber;
	}
```

```
	/* Setzt die Kontonummer auf den übergebenen String k */
	public void setKontonr(String k) {
		kontonr = k;
	}

	/* Liefert die Kontonummer des Kontos zurück */
	public String getKontonr() {
		return kontonr;
	}

	/* Setzt die Bankleitzahl auf den übergebenen String b */
	public void setBLZ(String b) {
		blz = b;
	}

	/* Liefert die Bankleitzahl des Kontos zurück */
	public String getBLZ() {
		return blz;
	}

	/* Fügt eine Buchung in die Buchungsliste hinzu */
	public void buchungHinzufuegen(Buchung b) {
		buchungen.fuegeSortiertEin(b);

	}

	/* Listet alle gespeicherten Buchungen auf dem Bildschirm auf */
	public void druckeBuchungsliste() {
		Datum d;
		Buchung b;
		double saldo;
		// Audruck der Buchungsliste von vorne nach hinten
		if (buchungen.getAnzahlBuchungen() > -1) {
			System.out.println("Inhaber: "+ inhaber + " BLZ: " +
				blz + " Kontonummer: " + kontonr);
			System.out.println("*** Gespeicherte Buchungen ***");
			// Rufe drucke()-Methode der Klasse Buchungsliste auf
			buchungen.drucke();
			System.out.println("Saldo: " + berechneSaldo());

			// Und jetzt rückwärts !!!
			System.out.println("*** Gespeicherte Buchungen ***");
			buchungen.druckerueck();
		}
	}

	/* Berechnet den Saldo des Kontos */
	public double berechneSaldo() {
		return buchungen.berechneSaldo();
	}

	/* Gibt die Anzahl der instantiierten Konten zurück */
	public static int anzahlKonten() {
		return anzKonten;
	}
}
```

```
/* Die Klasse realisiert ein doppelt-verkette Liste um Buchungselemente
 * zu speichern. Ebenso werden Operationen (Methoden) für die Arbeit auf
 * dieser Liste zur Verfügung gestellt.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 18.12.2001
 */
public class Buchungsliste {

	private Buchungselement kopf;
	private Buchungselement schluss;
	private int anzahlBuchungen;

	/* Erzeugt eine neue initialisierte Buchungsliste */
	public Buchungsliste() {
		kopf = schluss = null;
		anzahlBuchungen = 0;
	}

	/* Fügt eine Buchung sortiert an der richtigen Stelle ein */
	public void fuegeSortiertEin(Buchung b) {
		/* Aufruf der Hilfsmethode der der Listenkopf übergeben wird */
		kopf = fuegeSortiertEin(b, null, kopf);
		anzahlBuchungen = anzahlBuchungen + 1;
	}

	/* Fügt rekursiv eine Buchung in die Liste ein */
	private Buchungselement fuegeSortiertEin(Buchung b, Buchungselement vorgaenger,
Buchungselement element) {
		/* Element wird an das Ende der Liste angehängt */
		if (element == null) {
			/* Nuer Schluss wird gesetzt */
			schluss = new Buchungselement(b, vorgaenger, null);
			return schluss;
		}
		/* Element wird sortiert in die Liste eingefügt */
		else if (kleinerAls(b.getDatum(), (element.getBuchung()).getDatum())) {
			Buchungselement neu = new Buchungselement(b, vorgaenger, element);
			/* Setzen des Verweises auf den Vorgänger */
			element.prev = neu;
			return neu;
		}
		/* Gehe ein Element weiter und prüfe auf Einfügestelle */
		else {
			element.next = fuegeSortiertEin(b, element, element.next);
			return element;
		}
	}

	/* Vergleicht zwei Daten miteinander und gibt true zurück, falls
	   das Datum d1 vor dem Datum d2 liegt */
	private boolean kleinerAls(Datum d1, Datum d2) {
		/* Überprüfung des Jahres */
		if (d1.getJahr() > d2.getJahr()) {
			return false;
		}
		else if (d1.getJahr() < d2.getJahr()) {
			return true;
		}
		else {
			/* Wenn die Jahre gleich sind, dann Überprüfung des Monats */
			if (d1.getMonat() > d2.getMonat()) {
				return false;
			}
```

```
                else if (d1.getMonat() < d2.getMonat()) {
                      return true;
                }
                else {
                      /* Wenn auch die Monate gleich sind, dann Überprüfung des Tags
*/
                      if (d1.getTag() > d2.getTag()) {
                            return false;
                      }
                      else {
                            return true;
                      }
                }
          }
     }

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

     /* Berechnet den Saldo mittels einer Hilfsmethode */
     public double berechneSaldo() {
          return berechneSaldo(kopf);
     }

     /* Rekursiver durchlauf durch die Liste und Akkumulation des Saldos */
     private double berechneSaldo(Buchungselement kopf) {
          if (kopf != null) {
                return (kopf.getBuchung().getBetrag() + berechneSaldo(kopf.next));
          }
          else return 0;
     }

     /*    Gibt die gespeicherten Buchungen von vorne nach hinten mittels einer
Hilfsmethode aus */
     public void drucke() {
          System.out.println(durchlaufe(kopf));
     }

     /* Durchläuft rekursiv die Buchungsliste und gibt alle Buchungen aus */
     private String durchlaufe(Buchungselement kopf) {
          if (kopf != null) {
                Buchung b = kopf.getBuchung();
                return ((b.getDatum()).toString() + " " + b.getBeschreibung() + " " +
b.getBetrag()) + "\n" + durchlaufe(kopf.next);
          }
          else return "";
     }

     /* Gibt die gespeicherten Buchungen von hinten nach vorne mittels einer
Hilfsmethode aus */
     public void druckerueck() {
          System.out.println(durchlauferueck(schluss));
     }

     /* Durchläuft die Buchungsliste von hinten nach vorne und gibt alle Buchungen aus
*/
     private String durchlauferueck(Buchungselement schluss) {
          if (schluss != null) {
                Buchung b = schluss.getBuchung();
                return ((b.getDatum()).toString() + " " + b.getBeschreibung() + " " +
b.getBetrag()) + "\n" + durchlauferueck(schluss.prev);
          }
```

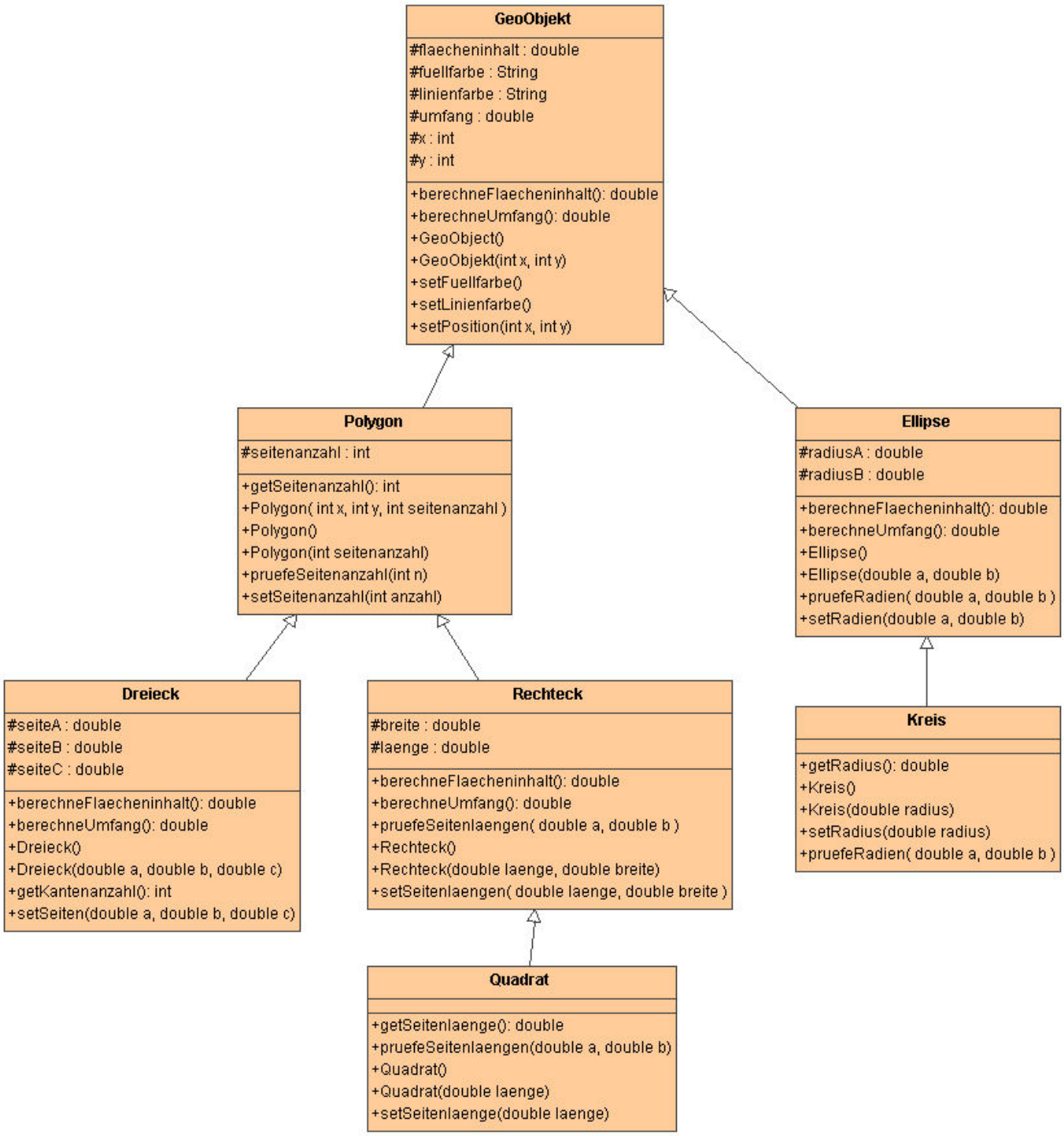
```
            else return "";
      }
}



/* Die Klasse realisiert ein Buchungselement einer
 * doppelt-verketten linearen Liste.
 * Autor: Thomas von der Maßen
 * Umgebung: JDK 1.3.1, Windows 2000
 * Erstellt: 18.12.2001
 */
public class Buchungselement {

      private Buchung b;
      protected Buchungselement next;
      protected Buchungselement prev;

      /* Erzeugt ein neues Buchungselement und inititalisiert die Verweise mit null */
      public Buchungselement(Buchung b) {
            this.b = b;
            this.next = null;
            this.prev = null;
      }

      /* Erzeugt ein neues Buchungselement und setzt die Verweise auf die übergebenen
        Vorgänger und Nachfolger */
      public Buchungselement(Buchung b, Buchungselement prev, Buchungselement next) {
            this.b = b;
            this.prev = prev;
            this.next = next;
      }

      /* Liefert die Buchung des Listenelements zurück */
      public Buchung getBuchung() {
            return b;
      }

      /* Liefert das nächste Buchungselement zurück */
      public Buchungselement getNext() {
            return next;
      }

      /* Liefert das vorhergehende Buchungselement zurück */
      public Buchungselement getPrev() {
            return prev;
      }

      /* Setzt den Nachfolger des Buchungselements auf next */
      public void setNext(Buchungselement next) {
            this.next = next;
      }

      /* Setzt den Vorgänger des Buchungselements auf prev */
      public void setPrev(Buchungselement prev) {
            this.prev = prev;
      }
}
```

Aufgabe 2)

Zeilennummer	Gültig	Begründung
4	Ja	Instantiierung von adr als neues Objekt der Klasse Adresse unter Verwendung des Konstruktors `public Adresse(String name, String strasse, int plz, String ort)`
5	Ja	Instantiierung von padr als neues Objekt der Klasse PrivatAdresse unter Verwendung des Konstruktors `public PrivatAdresse()`
6	Ja	Instantiierung von gadr als neues Objekt der Klasse GeschaeftsAdresse unter Verwendung des Konstruktors `public GeschaeftsAdresse()`
7		
8	Nein	Inkompatible Typen. Einem Objekt einer Unterklasse kann nicht einem anderen Objekt einer anderen Unterklasse (beide Objekte haben die gleiche Oberklasse) zugewiesen werden.
9	Nein	Inkompatible Typen. Das Objekt gadr der Klasse GeschaeftsAdresse kann nicht zu einem Objekt seiner Oberklasse umgewandelt (casting) werden.
10	Nein	Inkompatible Typen. Einem Objekt der Unterklasse kann nicht ein Objekt der Oberklasse zugewiesen werden.
11	Ja	Es wird das Attribut „name“ der Klasse PrivatAdresse gesetzt, welches von der Oberklasse „Adresse“ geerbt wurde.
12	Ja	Es wird das Attribut „plz“ der Klasse PrivatAdresse gesetzt, welches von der Oberklasse „Adresse“ geerbt wurde.
13	Ja	Es wird das Attribut „gebDatum der Klasse PrivatAdresse gesetzt, welches von der Oberklasse „Adresse“ geerbt wurde.
14		
15	Ja	Einem Objekt der Oberklasse wird ein Objekt der Unterklasse zugewiesen. Die spezifischen Attribute und Methoden der Unterklasse „PrivatAdresse“ sind jedoch auf dem Objekt adr nicht zugreifbar.
16	Ja	Es wird das Attribut „telefon“ der Klasse GeschaeftsAdresse gesetzt, welches von der Oberklasse „Adresse“ geerbt wurde.
17	Ja	Es wird auf das Attribut „name“ der Klasse Adresse zugegriffen.
18	Nein	Das Attribut „telefon“ ist in der Klasse Adresse nicht definiert und ist somit auch nicht zugreifbar.
19		
20	Ja	Dem Objekt a der Klasse Adresse wird eine Instanz der Klasse GeschaeftsAdresse zugewiesen. Die spezifischen Attribute und Methoden der Unterklasse „GeschaeftsAdresse“ sind jedoch auf dem Objekt a nicht zugreifbar.
21	Nein	Das Attribut „telefon“ ist in der Klasse Adresse nicht definiert und ist somit auch nicht zugreifbar.
22	Ja	Es wird auf das Attribut „plz“ der Klasse GeschaeftsAdresse zugegriffen.
23	Nein	Inkompatible Typen. Das Attribut „name“ der Klasse GeschaeftsAdresse erwartet ein Objekt der Klasse Name und keinen String
24	Ja	Es wird auf das Attribut „name“ der Klasse PrivatAdresse zugegriffen, welches von der Oberklasse Adresse geerbt wurde.
25	Nein	Inkompatible Typen. Das Attribut „plz“ des Objekts a der Klasse Adresse erwartet einen Wert vom Typ int und nicht vom Typ String.
26	Nein	Inkompatible Typen. Das Attribut „name“ des Objekts a der Klasse Adresse erwartet einen String und kein Objekt der Klasse Name.
27	Ja	Dem Attribut „name“ des Objekts gadr der Klasse GeschaeftsAdresse wird ein neues Objekt der Klasse Name zugewiesen. Dies geschieht durch Aufruf des Konstruktors `public Name(String anrede, String vorname, String nachname)`

Aufgabe 3

a)



b)
siehe Quellcode-Dateien

c)

```
public static void statistik(GeoObjekt[] feld) {
            GeoObjekt o;

            /* Instantiierung eine neuen Feldes, welches die jeweilige Anzahl an
geometrischen Figuren speichert.
             * 0 - # geometrische Objekte, 1 - # Polygone, 2- # Rechtecke, 3 - #
Quadrate, 4 - # Dreiecke,
             * 5 - # Ellipsen, 6 - # Kreise
             */
            int[] anzahl = new int[7];
            double gesamtflaeche = 0;

            // Initialisierung
            for (int i = 0; i < anzahl.length; i++) {
                  anzahl[i] = 0;
            }

            for (int i = 0; i < feld.length; i++) {
                  o = feld[i];
                  /* Prüfe die enthaltenen Objekte auf deren Typ hin.
                   * Dabei ist die Reihenfolge wichtig, da ein Objekt einer Unterklasse
auch
                   * ein Objekt der Oberklasse ist.
                   */
                  if (o instanceof Quadrat) {
                        // wenn Objekt ein Quadrat
                        anzahl[3]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else if (o instanceof Rechteck) {
                        // wenn Objekt ein Rechteck
                        anzahl[2]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else if (o instanceof Dreieck) {
                        // wenn Objekt ein Dreieck
                        anzahl[4]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else if (o instanceof Polygon) {
                        // wenn Objekt ein Polygon
                        anzahl[1]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else if (o instanceof Kreis) {
                        // wenn Objekt ein Kreis
                        anzahl[6]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else if (o instanceof Ellipse) {
                        // wenn Objekt eine Ellipse
                        anzahl[5]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else if (o instanceof GeoObjekt) {
                        // wenn Objekt ein geometrisches Objekt
                        anzahl[0]++;
                        gesamtflaeche = gesamtflaeche + o.berechneFlaecheninhalt();
                  }
                  else
                        System.out.println("Kein GeoObjekt");
            }
```

```
            System.out.println("Anzahl der jeweiligen geometrischen Objekte: ");
            System.out.println("GeoObjekte: " + anzahl[0]);
            System.out.println("Polygone  : " + anzahl[1]);
            System.out.println("Rechtecke : " + anzahl[2]);
            System.out.println("Quadrate  : " + anzahl[3]);
            System.out.println("Dreiecke  : " + anzahl[4]);
            System.out.println("Ellipsen  : " + anzahl[5]);
            System.out.println("Kreise    : " + anzahl[6]);
            System.out.println();
            System.out.println("Gesamtflaeche: " + gesamtflaeche);
      }
```

d)
siehe Quellcode-Dateien