Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 10 – Klassen und Objekte II

(Stand 22.03.2019)

Vorgabe für die Aufgaben 1 – 5:

In den Aufgaben 1 – 5 geht es um die Entwicklung von Programmen mit graphischen Oberflächen. Die Java-JDK-Klassenbibliothek stellt mit dem AWT-Framework eine Sammlung von Klassen zur Erstellung graphischer Oberflächen zur Verfügung. Folgende Ausschnitte der Klassen sind dabei für die Aufgaben von Bedeutung:

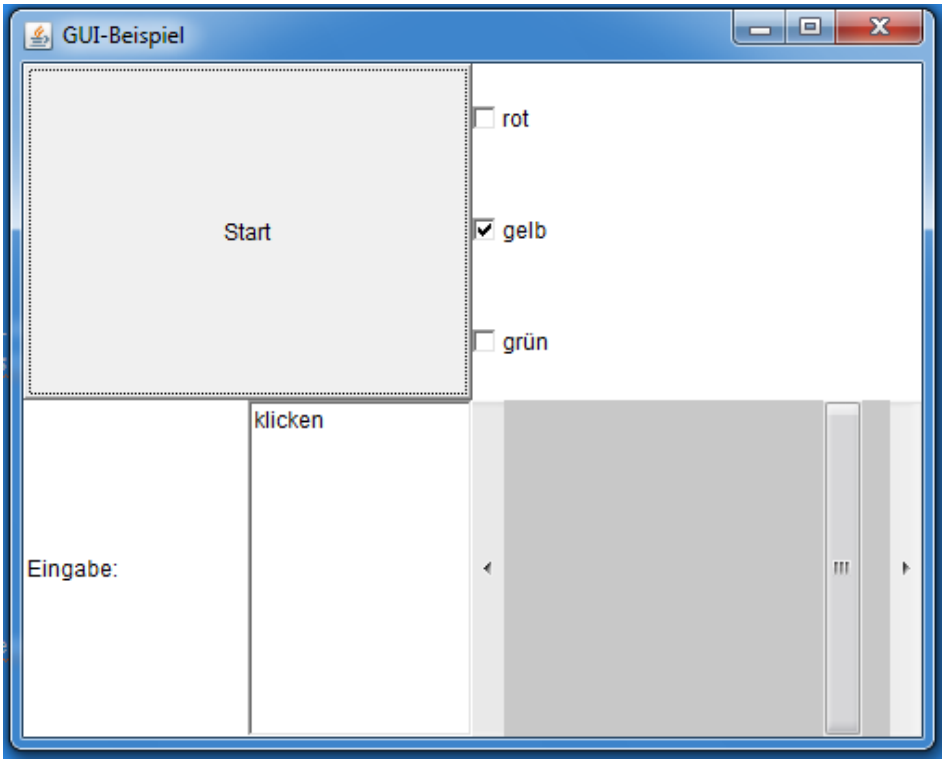
```
// realisiert ein Fenster auf dem Bildschirm
class Frame {
  Frame(String title)

  // setzen von Groesse und Bildschirmposition
  void setSize(int width, int height)
  void setLocation(int x, int y)

  // zuordnen eines Layout-Managers; dieser ordnet die dem Fenster
  // zugeordneten GUI-Komponenten tabellenartig an
  void setLayout(GridLayout layoutManager)

  // Methoden, mit denen dem Fenster bestimmte GUI-Komponenten zugeordnet
  // werden koennen
  void add(Panel comp)
  void add(Label comp)
  void add(Button comp)
  void add(Checkbox comp)
  void add(TextField comp)
  void add(Scrollbar comp)

  // muss mit true aufgerufen werden, um das Fenster sichtbar zu machen
  void setVisible(boolean visible)
}

// realisiert einen speziellen Layout-Manager, der für eine tabellenartige
// Ausrichtung der Elemente sorgt
class GridLayout
    GridLayout(int rows, int cols)
}

// realisiert einen nicht sichtbaren Container, der
// GUI-Komponenten aufnehmen und verwalten kann
class Panel {
  Panel()

  // zuordnen eines Layout-Managers; dieser ordnet die dem Panel
  // zugeordneten GUI-Komponenten tabellenartig an
  void setLayout(GridLayout layoutManager)
```

```
  // Methoden, mit denen dem Panel bestimmte GUI-Komponenten zugeordnet
  // werden koennen
  void add(Panel comp)
  void add(Label comp)
  void add(Button comp)
  void add(Checkbox comp)
  void add(TextField comp)
  void add(Scrollbar comp)
}

// zur Darstellung von Texten
class Label {
  Label()
  Label(String label)
  void setLabel(String label)
  String getLabel()
}

// zur Darstellung von Buttons
class Button {
  Button()
  Button(String label)
  void setLabel(String label)
  String getLabel()
}

// zur Darstellung von Checkboxen
class Checkbox {
  Checkbox(String label)
  Checkbox(String label, boolean checked)
  void setLabel(String label)
  boolean getState()
}

// zur Darstellung von Texteingabefeldern
class TextField {
  TextField()
  TextField(String text)
  void setText(String text)
  String getText()
}

// zur Darstellung von Scrollbars
class Scrollbar {
  Scrollbar()
  void setOrientation(int orientation) // 0 = horizontal, 1 = vertikal
  void setValue(int value) // zwischen 0 und 100
  int getValue()
}
```

Um die Klassen des AWT-Frameworks nutzen zu können, müssen Sie in Ihren Programmen ganz oben folgende Zeile einfügen:

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

Das folgende Programm demonstriert die Nutzung der AWT-Klassen an einem Beispiel. Nach dem Start erscheint auf dem Bildschirm das folgende Fenster:

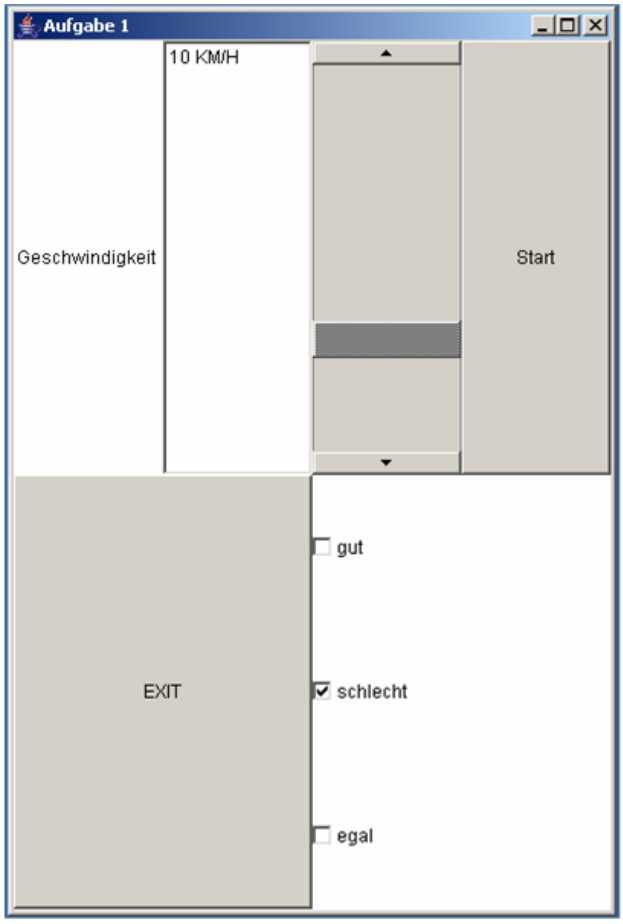



```
import java.awt.*;

public class GUIExample {

    public static void main(String[] args) {

        // Erzeugung eines Fensters
        Frame fenster = new Frame("GUI-Beispiel");

        // Festlegen von Position und Groesse
        fenster.setLocation(10, 20);
        fenster.setSize(500, 400);

        // dem Fenster wird ein Layout-Manager zugeordnet, der die dem
        // Fenster zugeordneten GUI-Komponenten in einer 2x2-Tabelle
        // darstellt
        GridLayout fensterLayout = new GridLayout(2, 2);
        fenster.setLayout(fensterLayout);

        // oben links wird dem Fenster ein Button zugeordnet
        Button start = new Button("Start");
        fenster.add(start);

        // oben rechts wird dem Fenster ein Panel mit 3 Checkboxen
        // zugeordnet, die untereinander stehen
        Checkbox box1 = new Checkbox("rot");
        Checkbox box2 = new Checkbox("gelb");
        box2.setState(true);
        Checkbox box3 = new Checkbox("grün");
        Panel checkboxPanel = new Panel();
        GridLayout checkboxPanelLayout = new GridLayout(3, 1);
        checkboxPanel.setLayout(checkboxPanelLayout);
        checkboxPanel.add(box1);
        checkboxPanel.add(box2);
        checkboxPanel.add(box3);
```

```
        fenster.add(checkboxPanel);

        // unten links wird dem Fenster ein Panel zugeordnet, das links
        // ein Label und rechts ein Texteingabefeld enthaelt
        Label label = new Label("Eingabe:");
        TextField eingabe = new TextField("klicken");
        Panel eingabePanel = new Panel();
        GridLayout eingabePanelLayout = new GridLayout(1, 2);
        eingabePanel.setLayout(eingabePanelLayout);
        eingabePanel.add(label);
        eingabePanel.add(eingabe);
        fenster.add(eingabePanel);

        // unten rechts wird dem Fenster ein horizontaler Scrollbar
        // zugeordnet
        Scrollbar bar = new Scrollbar();
        bar.setOrientation(0);
        bar.setValue(83);
        fenster.add(bar);

        // das Fenster wird sichtbar gemacht
        fenster.setVisible(true);
    }

}
```

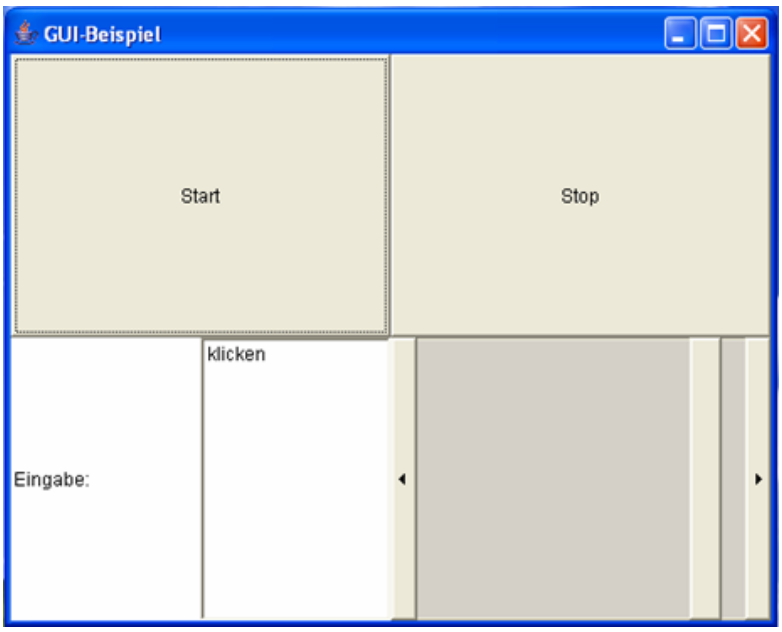
Aufgabe 1:

Entwickeln Sie auf der Basis der vorgestellten Java-AWT-Klassen ein Java-Programm, das folgende GUI erzeugt:



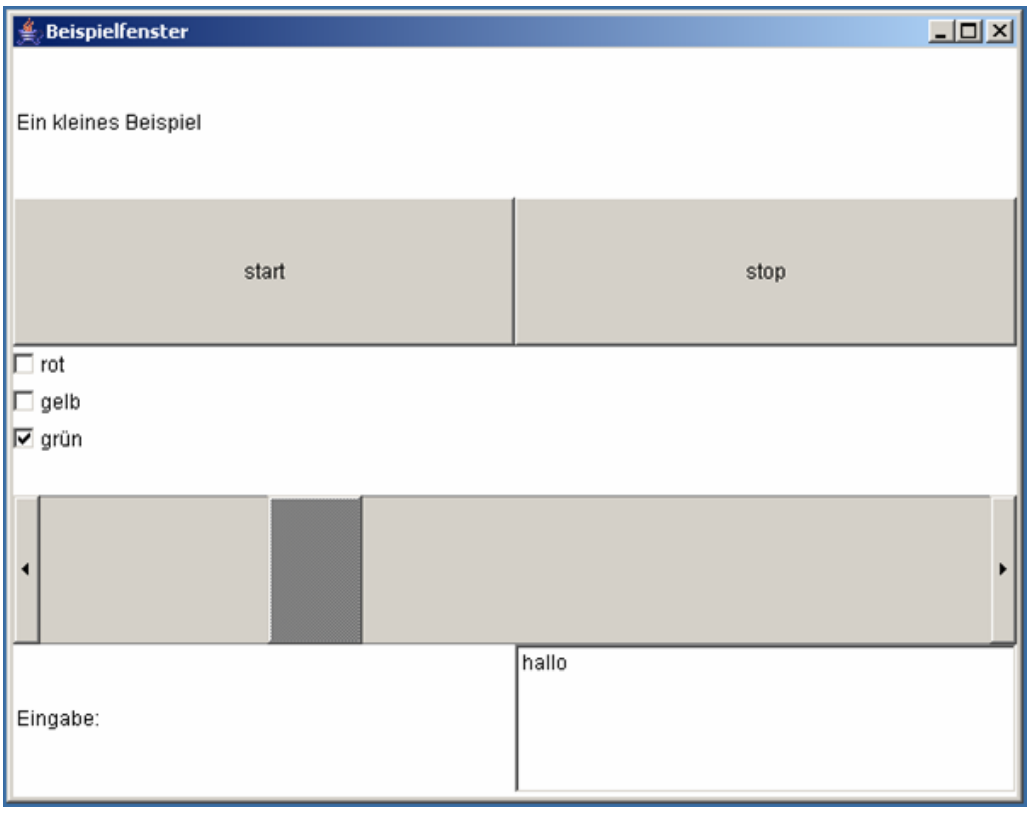
Aufgabe 2:

Entwickeln Sie auf der Basis der vorgestellten Java-AWT-Klassen ein Java-Programm, das folgende GUI erzeugt:



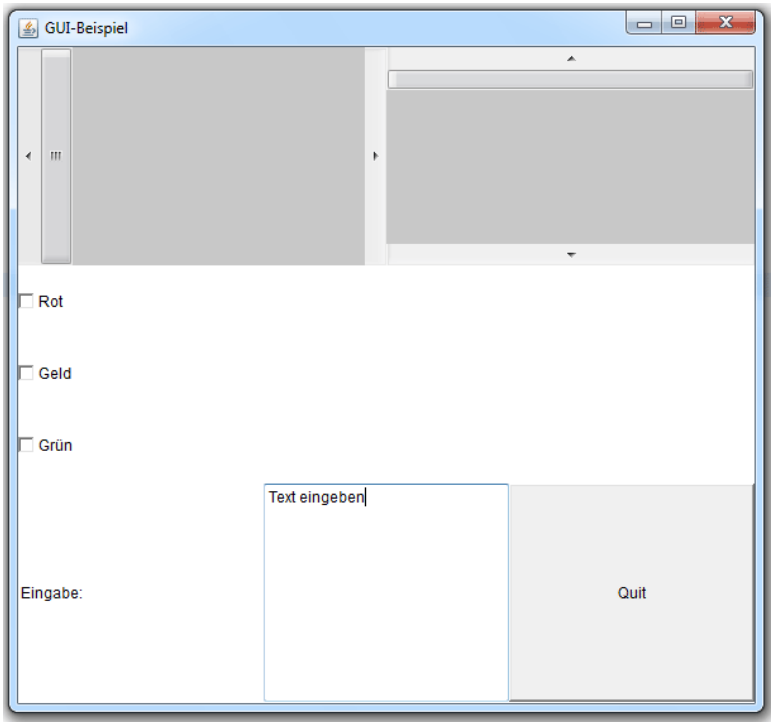
Aufgabe 3:

Entwickeln Sie auf der Basis der vorgestellten Java-AWT-Klassen ein Java-Programm, das folgende GUI erzeugt:



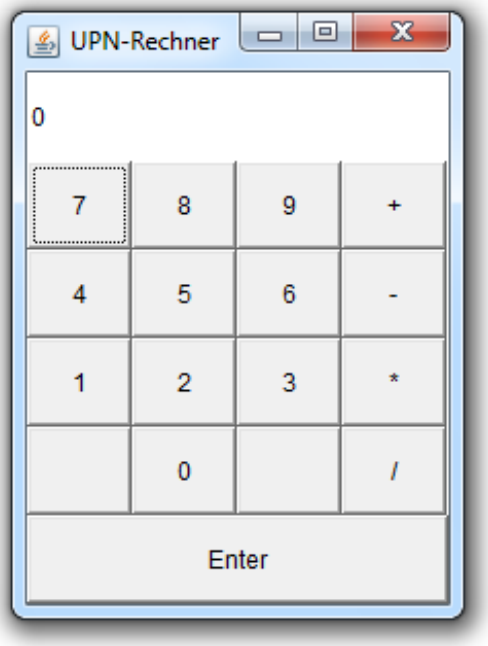
Aufgabe 4:

Entwickeln Sie auf der Basis der vorgestellten Java-AWT-Klassen ein Java-Programm, das folgende GUI erzeugt:



Aufgabe 5:

Bilden Sie auf der Basis Java-AWT-Klassen exakt folgende Taschenrechner-GUI nach:

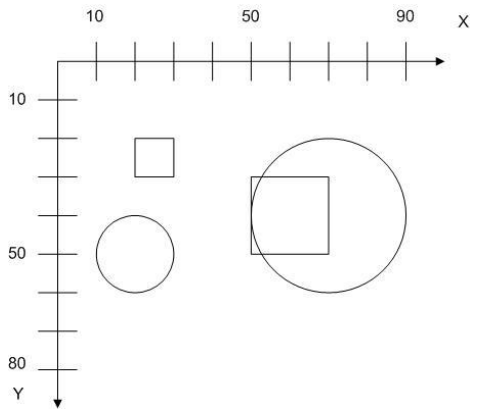


Aufgabe 6:

Schauen Sie sich die folgenden beiden Klassen an. Sie repräsentieren Kreise bzw. Quadrate, die auf einen Bildschirm mit einem rechtwinkligen Koordinatensystem gezeichnet werden können

```
class Kreis {
     // zeichnet einen Kreis mit dem Mittelpunkt bei Punkt (x=0/y=0) und
     // einem Radius von 10 Einheiten
     Kreis()

     // verschiebt den Kreis um den als Parameter uebergebenen Wert in der
     // Horizontalen
     void verschiebeHorizontal(int einheiten)

     // verschiebt den Kreis um eine Einheit in der Vertikalen
     void verschiebeVertikal()

     // vergroessert den Radius des Kreises um den als Parameter
     // uebergebenen Wert
     void vergroessere(int einheiten)

     // liefert die x-Koordinate des Kreises
     int getX()

     // liefert die y-Koordinate des Kreises
     int getY()
}

class Quadrat {
     // zeichnet ein Quadrat mit der linken oberen Ecke bei Punkt
     // (x=0/y=0) und einer Hoehe und Breite von 50 Einheiten
     Quadrat()

     // verschiebt das Quadrat so, dass dessen linke obere Ecke
     // anschliessend auf den Mittelpunkt des als Parameter
     // uebergebenen Kreises faellt
     void verschiebeNach(Kreis kreis)

     // vergroessert die Hoehe und Breite des Quadrates um den als
     // Parameter uebergebenen Wert
     void vergroessere(int einheiten)
}
```

Aufgabe: Schreiben Sie auf der Basis dieser Klassen ein Programm, dass die skizzierte Bildschirmausgabe erzeugt:



Aufgabe 7:

Schauen Sie sich die beiden folgenden Klassen an:

```
class Kreis {

	double PI = 3.1415;
	double radius;

	Kreis(double r) {
		radius = r;
	}

	void vergroessern(double wert) {
		radius += wert;
	}

	double liefereFlaeche() {
		return radius * radius * PI;
	}
}

// realisiert ein Quadrat mit einem internen die Raender des
// Quadrats berührenden Kreis
class QuadratMitKreis {

	Kreis kreis;
	double groesse;

	QuadratMitKreis(double g) {
		kreis = new Kreis(g / 2);
		groesse = g;
	}

	void vergroessern(double wert) {
		kreis.vergroessern(wert / 2);
		groesse += wert;
	}

	double liefereFlaeche() {
		return groesse * groesse;
	}

	double liefereFlaecheOhneKreis() {
		return liefereFlaeche() - kreis.liefereFlaeche();
	}
}
```

Aufgabe: Schreiben Sie durch Benutzung dieser Klassen ein Java-Programm, das folgendes tut:

- Zunächst wird ein QuadratMitKreis-Objekt mit einer Größe erzeugt, die vom Benutzer abgefragt wird. Achten Sie darauf, dass der Nutzer "ordentliche" Werte eingibt.
- Anschließend werden in einer Schleife, solange die Fläche des Quadrats kleiner als 100 ist, jeweils die Fläche des Quadrates und die Fläche des Quadrates abzüglich der Kreisfläche ausgegeben sowie das Quadrat um den Wert 1 vergrößert.

Im Folgenden wird ein Beispiel für eine mögliche Ausgabe des Programms gegeben (in <> die Eingaben des Benutzers):

```
Quadratgroesse: <8>
Quadratflaeche = 64.0
Quadratflaeche ohne Kreisflaeche = 13.736
Quadratflaeche = 81.0
Quadratflaeche ohne Kreisflaeche = 17.385
```

Aufgabe 8:

In dieser Aufgabe geht es darum, die Größe der Oberflächen zweier volumenmäßig ungefähr gleichgroßer 3-dimensionaler Behälterobjekte miteinander zu vergleichen. Schauen Sie sich dazu die folgenden Klassen an:

```
class Wuerfel {
	double kantenLaenge;

	Wuerfel(double k) {
		kantenLaenge = k;
	}

	double getOberflaeche() {
		return 6.0 * kantenLaenge * kantenLaenge;
	}

	double getVolumen() {
		return kantenLaenge * kantenLaenge * kantenLaenge;
	}

	void veraendern(double inkrement) {
		kantenLaenge += inkrement;
	}
}

class Kugel {
	double PI = 3.1415;

	double radius;

	Kugel(double r) {
		radius = r;
	}

	double getOberflaeche() {
		return 4.0 * PI * radius * radius;
	}

	double getVolumen() {
		return 4.0 / 3.0 * PI * radius * radius * radius;
	}

	void veraendern(double inkrement) {
		radius += inkrement;
	}
}
```

Aufgabe: Schreiben Sie durch Benutzung dieser Klassen ein Java-Programm, das folgendes tut:

- Zunächst werden ein Wuerfel- und ein Kugel-Objekt mit jeweils einer Größe (Kantenlänge bzw. Radius) erzeugt, die zuvor vom Benutzer abgefragt wird. Achten Sie darauf, dass der Nutzer "ordentliche" Werte eingibt.
- Anschließend wird das Volumen der beiden Objekte auf den Bildschirm ausgegeben
- Danach wird ein Inkrementwert *inkrement* vom Benutzer abgefragt.
- Anschließend werden in einer Schleife die Volumen der beiden Objekte "angeglichen", und zwar auf folgende Art und Weise: Das anfangs volumenmäßig größere Objekt wird in jedem Schleifendurchlauf um den Inkrementwert verkleinert (Methode *veraendern*) und das anfangs volumenmäßig kleinere Objekt wird um den Inkrementwert vergrößert. Die Schleife wird solange durchlaufen, bis das anfangs volumenmäßig größere Objekt volumenmäßig kleiner ist als das anfangs volumenmäßig kleinere Objekt.
- Zum Schluss werden Volumen und Oberfläche der beiden Objekte auf den Bildschirm ausgegeben.

Im Folgenden wird ein Beispiel für eine mögliche Ausgabe des Programms gegeben (in <> die Eingaben des Benutzers):

```
Kantenlaenge: <10.0>
Radius: <9.0>
Wuerfel-Volumen (Anfang): 1000.0
Kugel-Volumen (Anfang): 3053.5
Inkrement: <0.001>
Wuerfel-Volumen: 1612.3
Wuerfel-Oberfläche: 824.9
Kugel-Volumen: 1612.1
Kugel-Oberflaeche: 664.8
```

Aufgabe 9:

Gegeben ist folgende Klasse `Zahl`:

```
class Zahl {

	int wert;

	Zahl() {
		wert = (new java.util.Random()).nextInt(1000);
	}

	void inkr() {
		wert++;
	}

	void dekr() {
		wert--;
	}

	boolean equals(Zahl zahl) {
		return wert == zahl.wert;
	}

	int compareTo(Zahl zahl) {
```

```
            return wert - zahl.wert;
      }

      String nachString() {
            return "" + wert;
      }

}
```

Aufgabe: Schreiben Sie ein Programm, das zwei Zahl-Objekte erzeugt. Anschließend soll das größere Objekt solange verkleinert werden (`dekr`) und das kleiner Objekt solange vergrößert werden (`inkr`), bis das zunächst größere Objekt kleiner oder gleich dem zunächst kleineren Objekt ist.

Aufgabe 10:

Gegeben seien die beiden folgenden Klassen zur Darstellung und Bearbeitung von runden Glasböden und Trinkgläsern, die ein Trinkglas durch jeweils einen Glasboden und durch eine Füllstands-Angabe darstellen:

```
class Glasboden {
      double PI = 3.1415;

      double radius;

      Glasboden(double r) {
            radius = r;
      }

      void verkleinern(double x) {
            // verkleinert den Radius des Glasboden-Objekts um x
            radius = radius - x;
      }

      double flaeche() {
            // liefert die Flaeche des Glasboden-Objekts
            return PI * radius * radius;
      }

      double umfang() {
            // liefert den Umfang der Glasboden-Objekts
            return 2 * PI * radius;
      }

      String nachString() {
            // liefert die String-Darstellung des Glasboden-Obj.
            return "B(r=" + radius + ")";
      }
}

class TrinkGlas {
      Glasboden boden;

      double fuellStand;

      TrinkGlas(double fuellStand, Glasboden boden) {
            this.fuellStand = fuellStand;
            this.boden = boden;
      }
```

```
    void verkleinern(double x) {
          boden.verkleinern(x);
          fuellStand = fuellStand - x;
    }

    double innenFlaeche() {
          return boden.umfang() * fuellStand + boden.flaeche();
    }

    double fuellMenge() {
          return boden.flaeche() * fuellStand;
    }

    String nachString() {
          return "G(b=" + boden + ",s=" + fuellStand + ")";
    }

}
```

Aufgabe: Implementieren Sie auf der Grundlage dieser Klassen ein Java-Programm *TesteTrinkGlas*, in dem zunächst ein Trinkglas aus einem Glasboden mit vom Benutzer eingelesenen Radius- und Füllstandstandwerten erzeugt werden soll. Danach sollen in einer Schleife das Trinkglas jeweils um den Wert 5 verkleinert werden und das aktuelle Trinkglas, seine bedeckte Innenfläche und seine Füllmenge ausgegeben werden. Die Schleife soll nur durchlaufen werden, falls bzw. solange für die Innenfläche I und die Füllmenge M des Trinkglases gilt: I < M/8. Testen Sie das Programm bspw. mit einem Radius von 100 und einem Füllstand von 50.

Aufgabe 11:

Wir haben in der Vorlesung gelernt, dass Objekte im Umfeld der objektorientierten Programmierung Gegenstände, Sachverhalte, Lebewesen, Dinge, etc. repräsentieren. Sie besitzen Eigenschaften und Verhaltensweisen. Klassen stellen Baupläne für Objekte dar. Überlegen Sie sich **fünf** verschiedene Objekte aus ihrem tagtäglichen Umfeld und skizzieren Sie eine entsprechende Klasse. Erzeugen Sie Objekte und greifen Sie auf die Attribute und Methoden zu. Beispiel: Bücher

```
public class Buch {
  int isbn;
  String titel, autor;
  float preis;

  public Buch(String titel, String autor) {}
  public void veroeffentlichen(int isbn) {}
  public void anbieten(float preis) {}
  public void kaufen(int anzahl) {}
  public float preisErmitteln(int anzahl) {}
  public void verleihen(String person) {}
}

Buch hamster = new Buch("Programmieren spielend gelernt", "D. Boles");
hamster.veroeffentlichen(3519022974);
hamster.anbieten(39,90);
hamster.kaufen(8);
```

Aufgabe 12:

Der *Body-Mass-Index* (*BMI*) ist eine Maßzahl für die Bewertung des Körpergewichts eines Menschen. Der BMI berechnet sich aus dem Körpergewicht [kg] dividiert durch das Quadrat der Körpergröße [m^2]. Die Formel lautet: BMI = Körpergewicht / (Körpergröße in m)^2. Die Einheit des BMI ist demnach kg/m^2.

Der "wünschenswerte" BMI hängt vom Alter ab. Folgende Tabelle zeigt BMI-Werte für verschiedene Altersgruppen:

Alter	**BMI**
<= 24 Jahre	19-24
25-34 Jahre	20-25
35-44 Jahre	21-26
45-54 Jahre	22-27
55-64 Jahre	23-28
>= 65 Jahre	24-29

Liegt der BMI unterhalb der angegebenen Werte hat die Person Untergewicht, darüber Übergewicht.

Definieren Sie analog zu dem Beispiel in der Unterrichtseinheit 10 eine Klasse `Mensch`, für deren Objekte abgefragt werden kann, ob der entsprechende Mensch Unter-, Normal- oder Übergewicht hat. Grundlage ist dabei diesmal der BMI. Testen Sie Ihre Klasse an dem folgenden Beispielprogramm:

```
public class Aufgabe12 {
	public static void main(String[] args) {
		Mensch person = new Mensch(IO.readInt("Alter: "),
				IO.readDouble("Gewicht (kg) eingeben: "),
				IO.readDouble("Groesse (m) eingeben: "));

		if (person.hatUebergewicht()) {
			IO.println("Na, du alter Fettsack!");
		} else if (person.hatUntergewicht()) {
			IO.println("Na, du Hungerhaken!");
		} else {
			IO.println("Idealer Gewichtsbereich!");
		}
		IO.println("Dein BMI ist " + person.getBMI());
	}
}
```

Aufgabe 13:

Schauen Sie sich das folgende Programm an:

```
class Rabatte {

	public static void main(String[] args) {
		double preis = 100.0; // Euro
		double kleinRabatt = 10.0; // Prozent
		double grossRabatt = 15.0; // Prozent
```

```
            // Erzeugen eines Haendlers mit Preis- und
            // Rabattinformationen
            Haendler haendler =
              new Haendler(preis, kleinRabatt, grossRabatt);

            int anzahlKaeufe = IO.readInt("Anzahl Kaeufe: ");
            for (int i = 0; i < anzahlKaeufe; i++) {
                  int menge = IO.readInt("Zu kaufende Menge: ");
                  // eine bestimmte Menge beim Haendler einkaufen;
                  // geliefert werden die jeweiligen Kosten
                  double kosten = haendler.kaufen(menge);
                  IO.println("Anfallende Kosten: " + kosten);
            }

            // Ausgabe der vom Haendler gesamt erzielten Einnahmen
            IO.println("Einnahmen: " +
                        haendler.liefereGesamteEinnahmen());

            // Ausgabe der vom Haendler gesamt gewaehrten Rabatte;
            // (also "Rabattverluste")
            IO.println("Rabatte: " +
                        haendler.liefereGesamtGegebeneRabatte());
      }
}
```

In diesem Programm wird zunächst ein Händler erzeugt, der ein einzelnes Produkt zu einem angegebenen Preis verkauft und dabei seinen Kunden Mengenrabatte einräumt:

- Bei einem Kaufvorgang bis zu einer Menge von 5 Produkten (einschließlich) wird kein Rabatt gewährt.
- Bei einem Kaufvorgang zwischen 6 und 10 Produkten (einschließlich) wird ein "kleinRabatt" (in Prozent) gewährt.
- Bei einem Kaufvorgang über 10 Produkten wird ein "grossRabatt" (in Prozent) gewährt.

Anschließend werden unter diesen Bedingungen eine bestimmte Anzahl an Kaufvorgängen durchgeführt und die jeweils anfallenden Kosten ausgegeben. Zum Schluss werden die gesamten Einnahmen sowie die insgesamt gewährten Rabatte des Händlers ausgegeben.

Aufgabe: Implementieren Sie eine Klasse *Haendler*, so dass sich das obige Programm compilieren lässt und die beschriebene Semantik erfüllt wird. Sie dürfen das gegebene Programm natürlich nicht verändern.

Im Folgenden wird ein Beispiel für eine mögliche Ausgabe des Programms gegeben (in <> die Eingaben des Benutzers):

```
Anzahl Kaeufe: <3>
Zu kaufende Menge: <5>
Anfallende Kosten: 500.0
Zu kaufende Menge: <7>
Anfallende Kosten: 630.0
Zu kaufende Menge: <12>
Anfallende Kosten: 1020.0
Einnahmen: 2150.0
Rabatte: 250.0
```

Aufgabe 14:

Ein Wörterbuch im Sinne der folgenden Aufgabe ist dadurch charakterisiert, dass es mehrere Begriffe enthält, denen jeweils eine einzelne Übersetzung zugeordnet ist.

Teilaufgabe (a): Implementieren Sie eine Klasse `Woerterbuch` mit folgenden Methoden:

- Einen Konstruktor, der die maximale Anzahl an zu speichernden Begriffen als Parameter übergeben bekommt.
- Eine Methode `einfuegen` mit zwei `String`-Parametern, nämlich einem Begriff und einer Übersetzung. Existiert der übergebene Begriff bereits im Wörterbuch, soll die alte Übersetzung durch die neue überschrieben werden.
- Eine Methode `getUebersetzung`, die einen String als Parameter übergeben bekommt und die im Wörterbuch gespeicherte Übersetzung dieses Begriffs als Funktionswert (`String`) liefert.

Teilaufgabe (b): Schreiben Sie mit Hilfe der Klasse `Woerterbuch` ein kleines Programm, in dem ein Benutzer zunächst eine Menge an Begriffen und Übersetzungen eingeben kann und er sich anschließend zu einzelnen einzugebenden Begriffen die gespeicherten Übersetzungen ausgeben lassen kann.

Aufgabe 15:

Definieren Sie eine Klasse *Wuerfel* mit einer Methode *wuerfeln*. Die Methode soll Zufallszahlen zwischen 1 und 6 erzeugen und liefern. Nutzen Sie dazu die Funktion Math.random(), die zufällig double-Werte zwischen 0 und 1 (ausschließlich) generiert (`double zufall = Math.random();`)

Schreiben Sie weiterhin ein Programm, in dem zunächst ein Würfel erzeugt wird. Dieser wird anschließend 100.000 Mal gewürfelt. Geben Sie danach auf den Bildschirm aus, wie oft die einzelnen Zahlen gewürfelt worden sind.

Aufgabe 16:

Definieren Sie eine Klasse *LottoMaschine* mit einer Methode *naechsteZahl*. Die Methode soll Zufallszahlen zwischen 1 und 49 erzeugen und liefern. Aber Achtung: Wie bei einer richtigen Lottoziehung darf jede Zahl nur einmal vorkommen. Nutzen Sie dazu die Funktion Math.random(), die zufällig double-Werte zwischen 0 und 1 (ausschließlich) generiert (`double zufall = Math.random();`)

Schreiben Sie weiterhin ein Programm, das die Ziehung der Lottozahlen (6 Zahlen plus eine Zusatzzahl) simuliert und die Zahlen auf den Bildschirm ausgibt.

Aufgabe 17:

In dieser Aufgabe geht es um die Verschlüsselung von Texten. Klartexte über einem Klartextalphabet werden durch die Anwendung von Verschlüsselungsalgorithmen in Geheimtexte über einem Geheimtextalphabet überführt. Verschlüsselungsverfahren sollen gewährleisten, dass nur Befugte bestimmte Botschaften lesen können. Das

Chiffrieren ist ein Verschlüsselungsverfahren, bei dem jeder Buchstabe in einem Text durch einen anderen Buchstaben ersetzt wird.

In dieser Aufgabe sollen Sie eine Klasse *CaesarVerschluesselung* implementieren, die einen Konstruktor und zwei Methoden definiert:

```
class CaesarVerschluesselung{
     CaesarVerschluesselung(int verschiebung)
     String chiffrieren(String botschaft);
     String dechiffrieren(String botschaft);
}
```

Die Methode `chiffrieren` soll eine entsprechende Umsetzung der übergebenen Zeichen vornehmen. Die Methode `dechiffrieren` bildet die umgekehrte Funktion.

Die Caesar-Verschlüsselung beruht dabei auf einem Geheimtextalphabet, das um eine bestimmte *Stellenzahl n* gegenüber dem Klartextalphabet verschoben ist. Beispiel für n = 3: a -> d, b -> e, c -> f, ..., w -> z, x -> a, y -> b, z -> c. Chiffriert werden sollen hier nur Kleinbuchstaben. Alle anderen Zeichen sollen unverändert zurückgegeben werden. Implementieren Sie neben den beiden Methoden einen Konstruktor, dem die *Stellenzahl* als Parameter übergeben wird (Schlüssel).

Hinweis: Die Klasse java.lang.String stellt eine Methode `char charAt(int index)` zur Verfügung, die den Charakter an der index-ten Stelle liefert.

Implementieren Sie weiterhin ein Testprogramm (= Aufruf aller Methoden) für die Klasse *CaesarVerschluesselung*.

Aufgabe 18:

In dieser Aufgabe geht es um die Verschlüsselung von Texten. Klartexte über einem Klartextalphabet werden durch die Anwendung von Verschlüsselungsalgorithmen in Geheimtexte über einem Geheimtextalphabet überführt. Verschlüsselungsverfahren sollen gewährleisten, dass nur Befugte bestimmte Botschaften lesen können. Konkret geht es in dieser Aufgabe um die Vignere-Verschlüsselung.

Die Vignere-Verschlüsselung basiert auf dem so genannten Vignere-Quadrat:

```
Klar   a b c d e f g h i j k l m n o p q r s t u v w x y z

0      a b c d e f g h i j k l m n o p q r s t u v w x y z
1      b c d e f g h i j k l m n o p q r s t u v w x y z a
2      c d e f g h i j k l m n o p q r s t u v w x y z a b
3      d e f g h i j k l m n o p q r s t u v w x y z a b c
…
11     l m n o p q r s t u v w x y z a b c d e f g h i j k
…
24     y z a b c d e f g h i j k l m n o p q r s t u v w x
25     z a b c d e f g h i j k l m n o p q r s t u v w x y
```

Zeile 1 des Quadrates enthält ein Geheimtextalphabet mit einer Caesar-Verschiebung von 1; Zeile 2 des Quadrates enthält ein Geheimtextalphabet mit einer Caesar-Verschiebung von 2; usw.

Bei der Vignere-Verschlüsselung geht man so vor, dass man jeden Buchstaben einer geheim zu haltenden Botschaft anhand einer anderen Zeile des Vignere-Quadrates verschlüsselt. Um die Botschaft wieder entschlüsseln zu können, muss der Empfänger allerdings wissen, welche Zeile des Vignere-Quadrates für den jeweiligen Buchstaben benutzt wurde. Deshalb tauschen Sender und Empfänger vorher ein Schlüsselwort aus.

Nehmen wir einmal an, die zu verschlüsselnde Botschaft sei „`truppenabzugnachosten`“ und das Schlüsselwort sei „`licht`“. Zunächst wird das Schlüsselwort über die Botschaft geschrieben und so lange wiederholt, bis jeder Buchstabe der Botschaft mit einem Buchstaben des Schlüsselwortes verknüpft ist.

```
Schlüsselwort     lichtlichtlichtlichtl
Klartext          truppenabzugnachosten

Geheimtext        ezwwipvcisfoehvswuaxy
```

Der Geheimtext wird dann folgendermaßen erzeugt: Um den ersten Buchstaben „`t`“ zu verschlüsseln, stellen wir zunächst fest, dass über ihm der Buchstabe „`l`“ steht, der wiederum auf eine bestimmte Zeile des Vignere-Quadrates verweist. Die mit „`l`“ beginnenden Reihe 11 enthält das Geheimtextalphabet, das wir benutzen, um den Stellvertreter des Klarbuchstabens „`t`“ zu finden. Also folgen wir der Spalte unter „`t`“ bis zum Schnittpunkt mit der Zeile 11 bzw. „`l`“, und dort befindet sich der Buchstabe „`e`“ im Geheimtext. Genauso gehen wir mit allen weiteren Buchstaben der Botschaft vor.

Die Entschlüsselung eines Zeichen eines Geheimtextes erfolgt auf umgekehrten Weg: Anhand des aktuellen Schlüsselzeichens wird die aktuelle Zeile des Quadrates bestimmt. Aus der Spalte des Zeichens leitet sich dann das Klartextzeichen ab.

Konkrete Aufgabe: Implementieren Sie eine Klasse `Vignere`, die es ermöglicht, Klartexte zu ver- und entschlüsseln. Das folgende Grundgerüst sei dabei vorgegeben. Verschlüsselt werden sollen nur Kleinbuchstaben. Alle anderen Zeichen sollen unverändert bleiben.

```
public class Vignere {

	// enthaelt nur Kleinbuchstaben!
	private String schluesselwort;

	// speichert das Vignere-Quadrat
	private char[][] quadrat = null;

	// schluessel darf nur Kleinbuchstaben enthalten!
	public Vignere(String schluessel) {
		schluesselwort = schluessel;
		// initialisieren des Vignere-Quadrates
		quadrat = new char[26][26];
		for (int i = 0; i < 26; i++) {
			for (int j = 0; j < 26; j++) {
				quadrat[i][j] = (char) ('a' + (i + j) % 26);
			}
		}
	}

	// liefert zu einem uebergebenen Klartext unter Nutzung des im
	// Attribut schluesselwort gespeicherten Schluesselwortes den
```

```
    // entsprechenden Geheimtext
    public String verschluesseln(String klartext) {

          // muss von Ihnen implementiert werden!

    }

    // liefert zu einem uebergebenen Geheimtext unter Nutzung des
    // im Attribut schluesselwort gespeicherten Schluesselwortes
   // den entsprechenden Klartext
    public String entschluesseln(String geheimtext) {

         // muss von Ihnen implementiert werden!

    }
}
```

Aufgabe 19:

Sicher haben Sie in Ihrer Kindheit mal das Spiel *Superhirn* (*Mastermind*) gespielt. Es ist ein Spiel für zwei Personen. Der eine Spieler wählt zu Beginn einen vierstelligen Farbcode aus, wobei sechs verschiedene Farben zur Verfügung stehen, die auch mehrfach verwendet werden können. Der andere Spieler versucht den Code herauszufinden. Als Hilfestellung bekommt er nach jedem Zug die Information, wie viele Stifte er in Farbe und Position richtig gesetzt hat, und wie viele Stifte zwar die richtige Farbe haben, aber an einer falschen Position stehen. Ein Treffer in Farbe und Position wird durch einen schwarzen Stift angezeigt, ein lediglich in der Farbe übereinstimmender Stift wird durch einen weißen Stift angezeigt (siehe auch http://de.wikipedia.org/wiki/Mastermind).

Die 6 Farben ersetzen wir in dieser Aufgabe durch die Ziffern 1 bis 6.

Implementieren Sie eine Klasse *MasterMind* mit folgenden Methoden:

```
class MasterMind {

  // erzeugt per Zufall 4-stellige Zahl mit Ziffern 1 bis 6
  MasterMind()

  // liefert die Anzahl an schwarzen Stiften bezogen auf die
  // uebergebenen Ziffern;
  // das Array hat eine Laenge von 4 und enthaelt nur Ziffern
  // 1 bis 6
  int anzahlSchwarzeStifte(int[] ziffern)

  // liefert die Anzahl an weissen Stiften bezogen auf die
  // uebergebenen Ziffern;
  // das Array hat eine Laenge von 4 und enthaelt nur Ziffern
  // 1 bis 6
  int anzahlWeisseStifte(int[] ziffern)

}
```

Nutzen Sie im Konstruktor die Funktion Math.random(), die zufällig double-Werte zwischen 0 und 1 (ausschließlich) generiert (`double zufall = Math.random();`)

Implementieren Sie mit Hilfe der Klasse *MasterMind* das MasterMind-Spiel, in dem ein menschlicher Spieler gegen den Computer spielt. Der Computer wählt dabei den Code aus und der Mensch muss diesen entsprechend der Spielregeln erraten.

Aufgabe 20:

Hintergrund: Eine Bank führt die ersten sechs Buchungen bei jedem Girokonto kostenlos durch. Für die nächsten zehn Buchungen berechnet sie 30 Cent pro Buchung, und für jede weitere Buchung 20 Cent.

Implementierung: Schauen Sie sich das folgende objektorientierte Java-Programm an. In einer Schleife liest es jeweils eine Anzahl an durchzuführenden Buchungen ein, registriert diese für das Konto, fragt beim Konto die bisher angefallenen Kontoführungsgebühren ab und gibt diese auf dem Bildschirm aus:

```
class OOGebuehren {
     public static void main(String[] args) {
           Girokonto meinKonto = new Girokonto();
           while (true) {
                 // Anzahl an neuen Buchungen einlesen
                 int buchungen;
                 do {
                       buchungen = IO.readInt("Neue Anzahl Buchungen: ");
                 } while (buchungen < 0);

                 // vermerkt weitere Buchungen
                 meinKonto.neueBuchungen(buchungen);

                 // berechnet die seit Kontoeroeffnung
                 // angefallenen Kontogebuehren
                 int gebuehren =
                     meinKonto.berechneKontofuehrungsgebuehren();

                 // aktuelle Kontofuehrungsgebuehren ausgeben
                 IO.println("Gebuehren = " + gebuehren / 100 + ","
                              + gebuehren % 100
                             + " EUR");
           }
     }
}
```

Aufgabe: Implementieren die dazugehörige Klasse `Girokonto`!

Aufgabe 21:

Schauen Sie sich das folgende objektorientierte Java-Programm an. Es simuliert den Kauf in Einkaufshops, zum einen ein Buch-Shop, zum anderen ein DVD-Shop. In den Shops wird jeweils nur ein Produkt verkauft (Buch bzw. DVD). Anfangs werden der Buchpreis sowie der DVD-Preis eingelesen. Der Benutzer kann nun in einer Schleife wahlweise Bücher oder DVDs kaufen. Die gewünschten Mengen werden dem jeweiligen Händler mitgeteilt. Nachdem der Benutzer den Kaufvorgang abgeschlossen hat, werden die Gesamteinnahmen der beiden Shops auf den Bildschirm ausgegeben.

```
public class UE23Aufgabe11 {

     public static void main(String[] args) {
           double buchPreis = IO.readDouble("Buchpreis: ");
           double dvdPreis = IO.readDouble("DVD-Preis: ");
```

```
            Shop buchShop = new Shop(buchPreis);
            Shop dvdShop = new Shop(dvdPreis);

            char weiter = 0;
            do {
                char auswahl = IO.readChar("Buch oder DVD kaufen (b/d)?");
                int anzahl = IO.readInt("Anzahl Produkte: ");
                if (auswahl == 'b') {
                    buchShop.kaufen(anzahl);
                } else {
                    dvdShop.kaufen(anzahl);
                }

                weiter = IO.readChar("weiter einkaufen(j/n): ");
            } while (weiter == 'j');

            double einnamen = buchShop.liefereEinnahmen();
            System.out.println("Einnahmen des Buch-Shops = " + einnamen);

            einnamen = dvdShop.liefereEinnahmen();
            System.out.println("Einnahmen des DVD-Shops = " + einnamen);
        }

    }
```

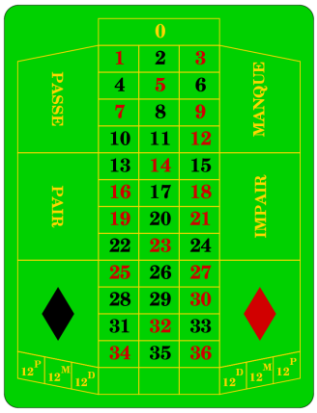
Aufgabe: Implementieren die dazugehörige Klasse `Shop`!

Beispiel für Programmablauf (Benutzereingaben in <>):

```
Buchpreis: <26.90>
DVD-Preis: <12.00>
Buch oder DVD kaufen (b/d)?<b>
Anzahl Produkte: <2>
weiter einkaufen(j/n): <j>
Buch oder DVD kaufen (b/d)?<d>
Anzahl Produkte: <3>
weiter einkaufen(j/n): <j>
Buch oder DVD kaufen (b/d)?<b>
Anzahl Produkte: <4>
weiter einkaufen(j/n): <n>
Einnahmen des Buch-Shops = 161.39999999999998
Einnahmen des DVD-Shops = 36.0
```

Aufgabe 22:

Definieren Sie eine Klasse `RouletteRad` mit einer Methode `drehen`. Diese soll eine zufällige Zahl zwischen 0 und 36 ermitteln.



(aus Wikipedia)

Für die zuletzt erdrehte Zahl sollen abgefragt werden können: die Zahl, Passe, Manque, Pair, Impair, Red, Black. Sehen Sie geeignete Methoden vor. Schreiben Sie ein kleines Testprogramm.

Aufgabe 23:

Sie wollen Konservendosen herstellen und müssen dazu bestimmte Werte der Dosen berechnen. Sei *u* der Umfang einer Dose in Zentimetern, *h* die Höhe einer Dose in Zentimetern und *PI* die Konstante mit dem Wert 3.141592. Dann berechnet sich

- Der Durchmesser des Dosenbodens $d_{boden} = u/PI$
- Die Fläche des Dosenbodens $f_{boden} = PI * (d_{boden}/2)^2$
- Die Mantelfläche der Dose $f_{mantel} = u * h$
- Die Gesamtfläche der Dose $f_{gesamt} = 2 * f_{boden} + f_{mantel}$
- Das Volumen der Dose $v = f_{boden} * h$

Definieren Sie eine Klasse *Konservendose* mit folgenden Methoden:

- Einem Konstruktor, dem der Umfang und die Höhe der Dose als Parameter übergeben werden.
- Methoden zum Ändern und Abfragen von Umfang und Höhe der Dose
- Methoden zum Abfragen von Durchmesser und Fläche des Dosenbodens
- Methoden zum Abfragen von Mantelfläche, Gesamtfläche und Volumen der Dose

Aufgabe 24:

Definieren Sie eine Klasse *Quader* mit folgenden Methoden:

- Einem Konstruktor, dem die Breite, Höhe und Tiefe des Quaders als Parameter übergeben werden.
- Methoden zum Ändern und Abfragen von Breite, Höhe und Tiefe des Quaders
- Methoden zum Abfragen von Gesamtkantenlänge, Gesamtfläche und Volumen des Quaders

Aufgabe 25:

Beim Spiel *Mäxchen* würfelt ein Spieler zwei Würfel. Würfelt er eine 1 und eine 2, ein so genanntes *Mäxchen*, bekommt er 1000 Punkte. Würfelt er einen Pasch, d.h. zwei gleiche Zahlen, erhält er das Hundertfache der entsprechenden Zahl (also bspw. 400 bei zwei 4en) als Punkte, ansonsten ist die Punktzahl 10 * höhere Augenzahl + niedrigere Augenzahl. Der Wurf 3, 5 hat also bspw. den Wert 53.

Implementieren Sie eine Funktion `maexchen`, die zwei gewürfelte Zahlen als Parameter übergeben bekommt und die erzielten Punkte als Funktionswert liefert.

Schreiben Sie dann ein Programm, in dem die Funktion `maexchen` mit jeweils zwei zufällig erzeugten Zahlen zwischen 1 und 6 so oft aufgerufen wird, bis 100.000 Punkte erreicht sind. Anschließend soll der prozentuale Anteil der Mäxchen-Würfe als `double`-Wert ausgegeben werden.

Nutzen Sie zum Würfeln die folgende Klasse:

```
class MaxWuerfel {
	int wuerfeln() {
		return (int) (Math.random() * 6) + 1;
	}
}
```

Aufgabe 26:

In dieser Aufgabe geht es um die objektorientierte Implementierung eines einfachen Geldspielautomaten. In einen solchen Automaten soll man Geld einwerfen, Spiele mit dem eingeworfenen Geld spielen und sich bestehende Guthaben auszahlen lassen können.

Definieren Sie eine Klasse **`GeldspielAutomat`**. Ein Geldspielautomat speichert ein Guthaben (int-Attribut für volle EUR-Beträge). Definieren und implementieren Sie dann folgende Methoden:

- Einen Default-Konstruktor: Anfangs ist das Guthaben gleich 0.
- Eine Methode **`geldEinwerfen`**, die als Parameter einen int-Wert für den entsprechend einzuwerfenden EUR-Betrag besitzt. Das Guthaben des Automaten soll entsprechend dem Parameterwert erhöht werden.
- Eine Getter-Methode für das Guthaben des Automaten.
- Eine Methode **`auszahlen`**, die das aktuelle Guthaben des Automaten als Rückgabewert liefert und intern auf 0 setzt.
- Eine Methode **`spielen`**. In der Methode sollen zwei Zufallszahlen zwischen 0 und 5 generiert werden. Sind die Zahlen gleich, hat der Spieler gewonnen und das aktuelle Guthaben wird verdreifacht. Sind die Zahlen ungleich, hat der Spieler verloren und das aktuelle Guthaben ist verloren. Im Gewinnfall liefert die Methode *true*, ansonsten *false*.

Ein Testprogramm könnte folgendermaßen aussehen:

```
GeldspielAutomat automat = new GeldspielAutomat();
// ...
automat.geldEinwerfen(IO.readInt("Einwerfen: "));
// ...
if (automat.spielen()) {
	System.out.println("gewonnen");
} else {
	System.out.println("verloren");
}
IO.println("Guthaben = " + automat.getGuthaben());
// ...
int geld = automat.auszahlen();
// ...
```