```java
/**
 * L&ouml;sung zu Aufgabe 46 (WiSe 2017/2018).
 * (Erg#&auml;nze: Modellierung ...)
 * @author [hier erg&auml;nzen]
 */
public class FidgetSpinnerArm {
    // Vereinbare die folgenden Attribute:
    // Durchmesser des ersten Halbstifts
    private double d1,
    // L&auml;nge des ersten Halbstifts.
    l1,
    // Durchmesser des zweiten Halbstifts.
    d2,
    // L&auml;nge des zweiten Halbstifts.
    l2;
    // Die nachfolgende Deklarationen nutzen Konstrukte aus Kapitel 11 und 12
    // (static/final), um oeffentlich sichtbare (public) Konstanten zu vereinbaren.
    public static final double APX_PI              = 3.14;
    public static final double GOLD_DICHTE         = 19.32;
    public static final double APX_WURZEL_FUENF    = 2.24;

    // Die Logik hinter dem nachfolgenden Konstruktor wird erst am
    // Donnerstag in der Vorlesung (Folien 9.66f.) erklaert. Aus diesem
    // Grund wird die Implementierung hier bereits vorgegeben.
    /**
     * Konstruktor f&uuml;r den Fidget-Spinner-Arm.
     * Dieser Konstruktor erwartet vier nicht-negative Werte, die den
     * Durchmesser und die L&auml;nge des ersten bzw. zweiten Zylinders
     * beschreiben. Diese Werte m&uuml;ssen beim Erzeugen eines Arms
     * &uuml;bergeben werden und sind danach weder ver&auml;nderbar noch auslesbar.
     * Als Einheit aller Ma&szlig;e wird "cm" angenommen.
     * @param d1 Durchmesser des ersten Zylinders.
     * @param l1 L&auml;nge des ersten Zylinders.
     * @param d2 Durchmesser des zweiten Zylinders.
     * @param l2 L&auml;nge des zweiten Zylinders.
     */
    public FidgetSpinnerArm(double d1, double l1, double d2, double l2) {
        if (d1 < 0) {
            //  Entspricht vom Prinzip her dem aus Racket bekannten
            //  (error ...), d.h. es wird eine Fehlermeldung ausgeloest.
            throw new IllegalArgumentException("Durchmesser d1 darf nicht negativ sein.");
        }
        if (l1 < 0) {
            throw new IllegalArgumentException("Laenge l1 darf nicht negativ sein.");
        }
        if (d2 < 0) {
            throw new IllegalArgumentException("Durchmesser d2 darf nicht negativ sein.");
```

```
48          }
49          if (l2 < 0) {
50              throw new IllegalArgumentException("Laenge l2 darf nicht
  negativ sein.");
51          }
52          // Speichere den Wert des Parameters d1 im Objekt-Attribut d1.
53          this.d1 = d1;
54          // Speichere den Wert des Parameters l1 im Objekt-Attribut l1.
55          this.l1 = l1;
56          // Speichere den Wert des Parameters d2 im Objekt-Attribut d2.
57          this.d2 = d2;
58          // Speichere den Wert des Parameters l2 im Objekt-Attribut l2.
59          this.l2 = l2;
60      }
61
62      /**
63       * Runden einer gegebenen Zahl auf eine gegebene Anzahl an
  Nachkommastellen.
64       * @param x Zu rundende Zahl.
65       * @param n Anzahl der erwuenschten Nachkommastellen.
66       * @return Auf <I>n</I> Nachkommastellen abgerundeter Wert von <I>x</
  I>.
67       */
68      double runde(double x, int n) {
69          if (n<0) {
70              throw new IllegalArgumentException("Anzahl der
  Nachkommastellen darf nicht negativ sein.");
71          }
72          double pow = Math.pow(10, n);
73          return ((long)(x * pow))/pow;
74      }
75      /**
76       * Berechent den Betrag einer Zahl.
77       * @param n Zahl für die der Betrag berechnet werden soll.
78       * @return Betrag von n
79       */
80      double abs(double n) {
81          return n < 0 ?
82                      n * -1 :    //true
83                      n;          //false
84      }
85
86      /**
87       * Approximative Berechung der Fl&auml;che eine Kreisscheibe f&uuml;r
  einen gegebenen Durchmesser.
88       * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
  verwendet.
89       * @param d Durchmesser der Kreisscheibe.
90       * @return Approximierte Fl&auml;che der Kreisscheibe.
91       */
92      double berechneKreisflaeche(double d) {
93          if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
  eintragen */) {
```

```
 94             throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
 95         }
 96         return APX_PI*(d/2)*(d/2);
 97     }
 98
 99     /**
100      * Approximative Berechung des Volumens eines Kegels f&uuml;r einen
    gegebenen
101      * Durchmesser, der auch der H&ouml;he entspricht.
102      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
103      * @param d Durchmesser des Kegels.
104      * @return Approximiertes Volumen des Kegels.
105      */
106     double berechneKegelvolumen(double d) {
107         if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
108             throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
109         }
110         return 1/3.0*berechneZylindervolumen(d, d);
111     }
112
113     /**
114      * Approximative Berechung des Volumens eines Zylinders f&uuml;r
    einen gegebenen Durchmesser
115      * und eine gegebene L&auml;nge.
116      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
117      * @param d Durchmesser der Zylinders.
118      * @param l L&auml;nge des Zylinders
119      * @return Approximiertes Volumen des Zylinders.
120      */
121     double berechneZylindervolumen(double d, double l) {
122         if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
123             throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
124         }
125         if (l < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
126             throw new IllegalArgumentException("Laenge darf nicht negativ
    sein.");
127         }
128         return berechneKreisflaeche(d)*l;
129     }
130
131     /**
132      * Approximative, ungerundete Berechung des Volumens des Arms.
133      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
134      * @return Approximiertes Volumen des Arms.
```

```
135      */
136     double berechneUngerundetesVolumen() {
137         return berechneKegelvolumen(d1)+berechneZylindervolumen(d1,
    l1)+berechneZylindervolumen(d2, l2);
138     }
139 
140     /**
141      * Approximative Berechung der Manelfl&auml;che eines Kegels f&uuml;r
    einen gegebenen
142      * Durchmesser, der auch der H&ouml;he entspricht.
143      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
144      * @param d Durchmesser des Kegels.
145      * @return Approximierte Mantelfl&auml;che des Kegels.
146      */
147     double berechneKegelmantelflaeche(double d) {
148         if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
149             throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
150         }
151 
152         return APX_PI * (d/2) * d * (APX_WURZEL_FUENF / 2.0);
153     }
154     /**
155      * Berechnet die Oberfl&auml;che eines Ringes.
156      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
157      * @param d1 Durchmaesser eines der beiden Kreise die den Ring
    beschraenken.
158      * @param d2 Durchmaesser des anderen Kreise die den Ring
    beschraenken.
159      * @return
160      */
161     double berechneRingflaeche(double d1, double d2) {
162         if (d1 < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
163             throw new IllegalArgumentException("Durchmesser d1 darf nicht
    negativ sein.");
164         }
165         if (d2 < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
166             throw new IllegalArgumentException("Durchmesser d2 darf nicht
    negativ sein.");
167         }
168 
169         return abs(berechneKreisflaeche(d1)-berechneKreisflaeche(d2));
170     }
171 
172     /**
173      * Approximative Berechung der Oberfl&auml;che eines Kegels f&uuml;r
    einen gegebenen
174      * Durchmesser, der auch der H&ouml;he entspricht.
```

```
175     * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
176     * @param d Durchmesser des Kegels.
177     * @return Approximierte Oberfl&auml;che des Kegels.
178     */
179    double berechneKegeloberflaeche(double d) {
180        if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
181            throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
182        }
183
184        return berechneKegelmantelflaeche(d) + berechneKreisflaeche(d);
185    }
186    /**
187     * Approximative Berechung der Mantelfl&auml;che eines Zylinders.
188     * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
189     * @param d Durchmesser des Zylinders.
190     * @param l L&auml;enge des Zylinders
191     * @return Approximierte Mantelfl&auml;che des Zylinders.
192     */
193    double berechneZylindermantelflaeche(double d,double l) {
194        if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
195            throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
196        }
197        if (l < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
198            throw new IllegalArgumentException("Laenge darf nicht negativ
    sein.");
199        }
200        return APX_PI*d*l;
201    }
202    /**
203     * Approximative Berechung der Oberfl&auml;che eines Zylinders
    f&uuml;r einen gegebenen
204     * Durchmesser und eine gegebene L&auml;nge.
205     * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
206     * @param d Durchmesser der Zylinders.
207     * @param l L&auml;nge des Zylinders
208     * @return Approximierte Oberfl&auml;che des Zylinders.
209     */
210    double berechneZylinderoberflaeche(double d, double l) {
211        if (d < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
212            throw new IllegalArgumentException("Durchmesser darf nicht
    negativ sein.");
213        }
214        if (l < 0 /* true loeschen, Test auf ungueltigen Parameter
    eintragen */) {
```

```
215             throw new IllegalArgumentException("Laenge darf nicht negativ
    sein.");
216         }
217 
218         return 2*berechneKreisflaeche(d)+berechneZylindermantelflaeche(d,
    l);
219     }
220 
221     /**
222      * Approximative Berechung der Oberfl&auml;che des Armes.
223      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
224      * @return Approximierte Oberfl&auml;che des Armes.
225      */
226     double berechneUngerundeteOberflaeche() {
227         return berechneKegelmantelflaeche(d1) +         //Kegel
228                 berechneZylindermantelflaeche(d1, l1) + //Erster Zylinder
229                 berechneRingflaeche(d1, d2) +           //Schnittflaeche
    zwischen dem erstem und dem zweitem Zylinder
230                 berechneZylindermantelflaeche(d2, l2) +
    berechneKreisflaeche(d2); //Zweiter Zylider
231     }
232 
233     /**
234      * Approximative Berechnung der Masse des Armes.
235      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
236      * @return Approximierte Masse des Armes.
237      */
238     double berechneUngerundeteMasse() {
239         return 0.0;
240     }
241 
242     /**
243      * Approximative, auf drei Nachkommastellen gerundete Berechnung des
    Volumens
244      * des Arms.
245      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
246      * @return Approximatives, auf drei Nachkommastellen gerundetes
    Volumen.
247      */
248     public double berechneGerundetesVolumen() {
249         return runde(berechneUngerundetesVolumen(), 3);
250     }
251 
252     /**
253      * Approximative, auf zwei Nachkommastellen gerundete Berechnung der
    Oberfl&auml;che
254      * des Arms.
255      * Bei der Berechnung wird als Approximation von &Pi; der Wert 3.14
    verwendet.
256      * @return Approximatives, auf zwei Nachkommastellen gerundetes
```

```
   Volumen.
257      */
258     public double berechneGerundeteOberflaeche() {
259         return runde(berechneUngerundeteOberflaeche(), 2);
260     }
261 
262     /**
263      * Approximative, auf zwei Nachkommastellen gerundete Berechnung der
   Masse (in g)
264      * des Stifts bei einer Dichte von 19,32 g/cm<SUP>3</SUP>.
265      * Bei der Berechnung wird als Approximation von PI der Wert 3,14
   verwendet.
266      * @return Approximative, auf zwei Nachkommastellen gerundete Masse
   in Gramm.
267      */
268     public double berechneGerundeteMasse() {
269         return berechneUngerundetesVolumen()*GOLD_DICHTE;
270     }
271 
272 }
273 
```