

```
1 package oop6_dat6_u;
2
3 import java.util.Comparator;
4
5 /**
6  *
7  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
8  */
9 public class SurfaceComparator implements Comparator<IMyObject>
10 {
11
12     public int compare(IMyObject c1, IMyObject c2)
13     {
14         return (c1.getSurface() - c2.getSurface());
15     }
16
17
18 }
```

```

1
2 package oop6_dat6_u;
3
4 /**
5  *
6  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
7  */
8
9 public class Sphere implements Comparable<IMyObject>, IMyObject
10 {
11
12     private int number; //Nummer der Kugel
13     private int s1; //Radius
14
15     public Sphere(int no, int a)
16     {
17         this.number = no;
18         this.s1 = a;
19     }
20
21     /**
22      * Berechnet das Volumen der Kugel
23      * @return Volumen
24      */
25     public int getVolume()
26     {
27         return (4/3 * (int) Math.PI * s1 * s1 * s1);
28     }
29
30     /**
31      * Berechnet die Oberfläche des Kubus
32      * @return Oberfläche
33      */
34     public int getSurface()
35     {
36         return (4 * (int) Math.PI * s1 * s1);
37     }
38
39     /**
40      * Liefert die grösste Seitenlänge als Ergebnis
41      * @return grösse Seitenlänge
42      */
43     public int getMaxDimension()
44     {
45         return s1;
46     }
47
48     /**
49      * String der Kugel
50      * @return Beschreibung der Kugel
51      */
52     @Override
53     public String toString()
54     {
55         /*
56          return ("-----\n"+
57             "SPHERE " + number + "\n" +
58             "Radius: " + s1 + "\n" +
59             "Volume = " + this.getVolume() + "\n" +
60             "Surface = " + this.getSurface()+ "\n");
61          */
62         return ("SPHERE " + number + " | " +
63             "Radius: " + s1 + " | " +
64             "Volume = " + this.getVolume() + " | " +
65             "Surface = " + this.getSurface()+ "\n");
66     }
67
68     /**
69      * Kubus vergleichen
70      * @param other Kugel
71      * @return Kubus gleich oder nicht
72      */
73     @Override
74     public boolean equals(Object other)
75     {
76         if (this == other) // 1. Test auf Identität
77         {
78             return true;
79         }
80         if (other == null) // 2. Test auf null
81         {
82             return false;
83         }
84         if (other.getClass() != this.getClass())// 3. Test auf Vergleichbarkeit
85         {
86             return false;
87         }
88         if ( ! (this.getVolume() == ((Sphere) other).getVolume()))// 4. Vergleich relev.
felder
89         {
90             return false;
91         }
92         return true;
93     }
94
95     /**
96      * Hash-Code wird bassierend auf der (eindeutigen) Nummer definiert
97      * @return Hash-Code
98      */
99     @Override
100    public int hashCode()

```

```

101 {
102     return number;
103 }
104
105 /**
106  * Kubus wird auf ihre Volumen verglichen
107  * @param c Cube zum Vergleichen
108  * @return Differenz des Volumen
109  */
110 public int compareTo(IMyObject c)
111 {
112     return (this.getVolume() - c.getVolume());
113 }
114
115
116 /**
117  * kleine "test" Methode
118  * @param args
119  */public static void main(String[] args)
120 {
121     Sphere s1 = new Sphere(1, 1);
122     Sphere s2 = new Sphere(2, 10);
123     Sphere s3 = new Sphere(3, 30);
124
125     System.out.println(s1.toString());
126     System.out.println(s2.toString());
127     System.out.println(s3.toString());
128
129     System.out.println("s1 == s2 ? : " + s1.equals(s2));
130     System.out.println("s2 == s3 ? : " + s2.equals(s3));
131     System.out.println("s1 > s2 ? : " + s1.compareTo(s2));
132     System.out.println("s2 > s3 ? : " + s2.compareTo(s3));
133 }
134 }

```

```

1
2 package oop6_dat6_u;
3
4 import java.awt.*;
5 import java.awt.event.*;
6 import javax.swing.*;
7 import java.util.Arrays;
8 import java.util.Comparator;
9
10 /**
11  * SolidWorks
12  *
13  * @author H. Diethelm, Hochschule Luzern
14  * @version 6.4.2008
15  */
16 public class SolidWorks extends JFrame implements ActionListener
17 {
18     private IMyObject[] solids;
19     private String fileName = "soliddata.txt";
20     // GUI Elemente erzeugen
21     private JButton bRead = new JButton("Read File '" + fileName + "'");
22     private JButton bSort1 = new JButton("Sort 'volume'");
23     private JButton bSort2 = new JButton("Sort 'surface'");
24     private JButton bSort3 = new JButton("Sort 'dimension'");
25     private JTextArea outputArea = new JTextArea();
26
27     public SolidWorks ()
28     {
29         super("SolidWorks");
30         setSize(450, 650);
31         setResizable(false);
32         //center on screen
33         setLocationRelativeTo(null);
34         setDefaultCloseOperation(EXIT_ON_CLOSE);
35
36         Container cp = getContentPane();
37         JPanel p1 = new JPanel();
38
39         p1.add(bRead);
40         cp.add(p1, BorderLayout.NORTH);
41
42         outputArea.setEditable(false);
43         cp.add(new JScrollPane(outputArea), BorderLayout.CENTER);
44
45         JPanel p2 = new JPanel();
46         p2.add(bSort1); p2.add(bSort2); p2.add(bSort3);
47         cp.add(p2, BorderLayout.SOUTH);
48
49         setVisible(true);
50
51         // Listener registrieren
52         bRead.addActionListener(this);
53         bSort1.addActionListener(this);
54         bSort2.addActionListener(this);
55         bSort3.addActionListener(this);
56     }
57
58     private void doOutput ()
59     {
60         outputArea.setText("");
61         for (int i = 0; i < solids.length; i++){
62             outputArea.append(solids[i].toString());
63         }
64     }
65
66     public void actionPerformed(ActionEvent e)
67     {
68         if (e.getSource() == bRead){
69             solids = SolidFileIO.readSolids(fileName);
70             doOutput();
71         }
72         if ((e.getSource() == bSort1) && (solids != null)){
73             // volume
74             Arrays.sort(solids);
75             doOutput();
76         }
77         if ((e.getSource() == bSort2) && (solids != null)){
78             // surface
79             Comparator<IMyObject> comparator = new SurfaceComparator();
80             Arrays.sort(solids, comparator);
81             doOutput();
82         }
83         if ((e.getSource() == bSort3) && (solids != null)){
84             // dimension
85             Comparator<IMyObject> comparator = new MaxDimensionComparator();
86             Arrays.sort(solids, comparator);
87             doOutput();
88         }
89     }
90
91     public static void main(String[] args)
92     {
93         SolidWorks sw = new SolidWorks();
94     }
95
96 }

```

```

1 package oop6_dat6_u;
2
3 import java.io.BufferedReader;
4 import java.io.FileNotFoundException;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.ArrayList;
8 import java.util.StringTokenizer;
9
10 /**
11  *
12  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
13  */
14 public class SolidFileIO
15 {
16     private static int number;
17
18     public static IMyObject[] readSolids(String fileName)
19     {
20         ArrayList<IMyObject> list = new ArrayList<IMyObject>();
21         int s1, s2, s3, sTemp;
22
23         try {
24             FileReader aFileReader = new FileReader(fileName);
25             BufferedReader aBufferedReader = new BufferedReader(aFileReader);
26
27             String line = aBufferedReader.readLine();
28             StringTokenizer sT;
29             while (line != null) { // while not EOF
30                 sT = new StringTokenizer(line, " ");
31                 String s0 = sT.nextToken();
32
33                 // Cube
34                 if (s0.equals("C")) {
35                     s1 = Integer.parseInt(sT.nextToken());
36                     s2 = Integer.parseInt(sT.nextToken());
37                     s3 = Integer.parseInt(sT.nextToken());
38                     //aufsteigend sortieren s1, s2, s3
39                     if (s1 > s2) {
40                         sTemp = s1;
41                         s1 = s2;
42                         s2 = sTemp;
43                     }
44                     if (s1 > s3) {
45                         sTemp = s1;
46                         s1 = s3;
47                         s3 = sTemp;
48                     }
49                     if (s2 > s3) {
50                         sTemp = s2;
51                         s2 = s3;
52                         s3 = sTemp;
53                     }
54                     list.add(new Cube(number, s1, s2, s3));
55                     number ++;
56                 }
57
58                 // Kugel - Sphere
59                 if (s0.equals("S")) {
60                     s1 = Integer.parseInt(sT.nextToken());
61                     list.add(new Sphere(number, s1));
62                     number ++;
63                 }
64
65                 // Zylinder - Cylinder
66                 if (s0.equals("Y")) {
67                     s1 = Integer.parseInt(sT.nextToken());
68                     s2 = Integer.parseInt(sT.nextToken());
69
70                     //aufsteigend sortieren s1, s2
71                     if (s1 > s2) {
72                         sTemp = s1;
73                         s1 = s2;
74                         s2 = sTemp;
75                     }
76                     list.add(new Cylinder(number, s1, s2));
77                     number ++;
78                 }
79
80                 line = aBufferedReader.readLine();
81             }
82             aFileReader.close();
83
84         } catch (FileNotFoundException ex) {
85             System.out.println("ERROR:");
86             ex.printStackTrace();
87         } catch (IOException ex) {
88             System.out.println("ERROR:");
89             ex.printStackTrace();
90         } catch (Exception ex) {
91             System.out.println("ERROR:");
92             ex.printStackTrace();
93         } finally {
94             return list.toArray(new IMyObject[1]); // siehe auch API-Dokumentation
95         }
96     }
97
98     /**
99     * mini Test
100     */

```

```
102     *param args
103     */
104     public static void main(String[] args)
105     {
106         SolidFileIO a = new SolidFileIO();
107         IMyObject[] c = a.readSolids("soliddata.txt");
108         for (IMyObject myObject : c) {
109             System.out.println(myObject.toString());
110         }
111     }
112 }
113 }
```

```
1
2 package oop6_dat6_u;
3
4 import java.util.Comparator;
5
6 /**
7  *
8  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
9  */
10 public class MaxDimensionComparator implements Comparator<IMyObject> {
11     public int compare(IMyObject c1, IMyObject c2)
12     {
13         return (c1.getMaxDimension() - c2.getMaxDimension());
14     }
15 }
```

```
1 package oop6_dat6_u;
2
3 /**
4  *
5  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
6  */
7 public interface IMyObject
8 {
9
10     int getVolume();
11     int getSurface();
12     int getMaxDimension();
13     String toString();
14     boolean equals(Object other);
15     int hashCode();
16 }
```



```

1 package oop6_dat6_u;
2
3 /**
4  *
5  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
6  */
7 public class Cylinder implements Comparable<IMyObject>, IMyObject
8 {
9
10 private int number; //Nummer des Zylinders
11 private int s1; //Durchmesser
12 private int s2; //Höhe
13
14 public Cylinder(int no, int a, int b)
15 {
16     this.number = no;
17     this.s1 = a;
18     this.s2 = b;
19 }
20
21 /**
22  * Berechnet das Volumen
23  * @return Volumen
24  */
25 public int getVolume()
26 {
27     return (s1 * s1 * (int) Math.PI * s2);
28 }
29
30 /**
31  * Berechnet die Oberfläche des Kubus
32  * @return Oberfläche
33  */
34 public int getSurface()
35 {
36     return (2 * (int) Math.PI * s1 * (s1 + s2));
37 }
38
39 /**
40  * Liefert die grösste Seitenlänge als Ergebnis
41  * @return grösse Seitenlänge
42  */
43 public int getMaxDimension()
44 {
45     if (s1 > s2) {
46         return s1;
47     } else {
48         return s2;
49     }
50 }
51
52 /**
53  * String der Kugel
54  * @return Beschreibung der Kugel
55  */
56 @Override
57 public String toString()
58 {
59     /*
60     return ("-----\n"+
61     "CYLINDER " + number + "\n" +
62     "Radius: " + s1 + "\n" +
63     "Höhe: " + s2 + "\n" +
64     "Volume = " + this.getVolume() + "\n" +
65     "Surface = " + this.getSurface() + "\n");
66     */
67     return ("CYLINDER " + number + " | " +
68     "Radius: " + s1 + " | " +
69     "Höhe: " + s2 + " | " +
70     "Volume = " + this.getVolume() + " | " +
71     "Surface = " + this.getSurface() + "\n");
72 }
73
74 /**
75  * vergleichen
76  * @param other
77  * @return Kubus gleich oder nicht
78  */
79 @Override
80 public boolean equals(Object other)
81 {
82     if (this == other) // 1. Test auf Identität
83     {
84         return true;
85     }
86     if (other == null) // 2. Test auf null
87     {
88         return false;
89     }
90     if (other.getClass() != this.getClass())// 3. Test auf Vergleichbarkeit
91     {
92         return false;
93     }
94     if ( ! (this.getVolume() == ((Cylinder) other).getVolume()))// 4. Vergleich relev.
95     {
96         return false;
97     }
98     return true;
99 }
100

```

```

101 /
102 * Hash-Code wird basierend auf der (eindeutigen) Nummer definiert
103 * @return Hash-Code
104 */
105 @Override
106 public int hashCode()
107 {
108     return number;
109 }
110
111 /**
112 * wird auf ihre Volumen verglichen
113 * @param c zum Vergleichen
114 * @return Differenz des Volumen
115 */
116 public int compareTo(IMyObject c)
117 {
118     return (this.getVolume() - c.getVolume());
119 }
120
121 /**
122 * kleine "test" Methode
123 * @param args
124 */
125 public static void main(String[] args)
126 {
127     Cylinder y1 = new Cylinder(1, 10, 10);
128     Cylinder y2 = new Cylinder(2, 20, 10);
129     Cylinder y3 = new Cylinder(3, 30, 5);
130
131     System.out.println(y1.toString());
132     System.out.println(y2.toString());
133     System.out.println(y3.toString());
134
135     System.out.println("y1 == y2 ? : " + y1.equals(y2));
136     System.out.println("y2 == y3 ? : " + y2.equals(y3));
137     System.out.println("y1 > y2 ? : " + y1.compareTo(y2));
138     System.out.println("y2 > y3 ? : " + y2.compareTo(y3));
139 }
140 }

```

```

1 package oop6_dat6_u;
2
3 /**
4  *
5  * @author Felix Rohrer <felix.rohrer@stud.hslu.ch>
6  */
7 public class Cube implements Comparable<IMyObject>, IMyObject
8 {
9
10 private int number; //Nummer des Kubus
11 private int s1, s2, s3; //Die 3 Seiten des Kubus
12
13 public Cube(int no, int a, int b, int c)
14 {
15     this.number = no;
16     this.s1 = a;
17     this.s2 = b;
18     this.s3 = c;
19 }
20
21 /**
22  * Berechnet das Volumen des Kubus
23  * @return Volumen
24  */
25 public int getVolume()
26 {
27     return (s1 * s2 * s3);
28 }
29
30 /**
31  * Berechnet die Oberfläche des Kubus
32  * @return Oberfläche
33  */
34 public int getSurface()
35 {
36     return (2 * (s1 * s2 + s1 * s3 + s2 * s3));
37 }
38
39 /**
40  * Liefert die grösste Seitenlänge als Ergebnis
41  * @return grösse Seitenlänge
42  */
43 public int getMaxDimension()
44 {
45     int res;
46     if (s1 > s2) {
47         if (s1 > s3) {
48             res = s1;
49         } else {
50             res = s3;
51         }
52     } else {
53         if (s2 > s3) {
54             res = s2;
55         } else {
56             res = s3;
57         }
58     }
59     return res;
60 }
61
62 /**
63  * String des Kubus
64  * @return Beschreibung des Kubus
65  */
66 @Override
67 public String toString()
68 {
69     /*
70     return ("-----\n"+
71         "CUBE " + number + "\n" +
72         "Sides: " + s1 + " " + s2 + " " + s3 + "\n" +
73         "Volume = " + this.getVolume() + "\n" +
74         "Surface = " + this.getSurface()+ "\n");
75     */
76     return ("CUBE " + number + " | " +
77         "Sides: " + s1 + " " + s2 + " " + s3 + " | " +
78         "Volume = " + this.getVolume() + " | " +
79         "Surface = " + this.getSurface()+ "\n");
80 }
81
82 /**
83  * Kubus vergleichen
84  * @param other Kubus
85  * @return Kubus gleich oder nicht
86  */
87 @Override
88 public boolean equals(Object other)
89 {
90     if (this == other) // 1. Test auf Identität
91     {
92         return true;
93     }
94     if (other == null) // 2. Test auf null
95     {
96         return false;
97     }
98     if (other.getClass() != this.getClass())// 3. Test auf Vergleichbarkeit
99     {
100        return false;
101    }

```

```

102         if ( ! (this.getVolume() == ((Cube) other).getVolume()) ) 4. Vergleich relev.
felder
103     {
104         return false;
105     }
106     return true;
107 }
108
109 /**
110  * Hash-Code wird basierend auf der (eindeutigen) Nummer definiert
111  * @return Hash-Code
112  */
113 @Override
114 public int hashCode()
115 {
116     return number;
117 }
118
119 /**
120  * Kubus wird auf ihre Volumen verglichen
121  * @param c Cube zum Vergleichen
122  * @return Differenz des Volumen
123  */
124 public int compareTo(IMyObject c)
125 {
126     return (this.getVolume() - c.getVolume());
127 }
128
129
130 /**
131  * kleine "test" Methode
132  * @param args
133  */
134 public static void main(String[] args)
135 {
136     Cube c1 = new Cube(1, 1, 2, 3);
137     Cube c2 = new Cube(2, 10, 20, 30);
138     Cube c3 = new Cube(3, 30, 20, 10);
139
140     System.out.println(c1.toString());
141     System.out.println(c2.toString());
142     System.out.println(c3.toString());
143
144     System.out.println("c1 == c2 ? : " + c1.equals(c2));
145     System.out.println("c2 == c3 ? : " + c2.equals(c3));
146     System.out.println("c1 > c2 ? : " + c1.compareTo(c2));
147     System.out.println("c2 > c3 ? : " + c2.compareTo(c3));
148 }

```