Programmieren 2
Übung Semesterwoche 6

1. Aufgabe: Klasse Cube

```
public class Cube implements Comparable<Cube>
{
    protected int number; // Nummer des Kubus
    int s1, s2, s3; // Die 3 Seiten des Kubus.

    public static void main(String[] args)
    {
        Cube c = new Cube(1,1,2,3);
        System.out.print(c.toString());
    }

    public Cube(int no, int a, int b, int c) // Die 3 Seiten des Kubus
    {
        number = no;
        s1 = a;
        s2 = b;
        s3 = c;
    }

    // Berechnet das Volumen des Kubus
    public int getVolume()
    {
        return s1*s2*s3;
    }

    // Berechnet die Oberfläche des Kubus
    public int getSurface()
    {
        return 2*(s1*s2) + 2*(s1*s3) + 2*(s2*s3);
    }

    // Liefert die grösste Seitenlänge als Ergebnis
    public int getMaxDimension()
    {
        int returnvalue = 0;

        if(s1>=s2 && s1>=s3)
        {
            returnvalue = s1;
        }
        else if(s2>=s1 && s2>=s3)
        {
            returnvalue = s2;
        }
        else if(s3>=s1 && s3>=s2)
        {
            returnvalue = s3;
        }
        return returnvalue;
    }

    // Liefert einen String, welcher den Kubus beschreibt
    public String toString()
    {
        return "--------------------------\n\rCUBE: "+number+
        "\n\rSides: "+s1+" "+s2+" "+s3+"\n\rVolume: "+getVolume()+
        "\n\rSurface: "+getSurface()+"\n\r";
    }
    public boolean equals(Object other)
    {
        if (this == other) // 1. Test auf Identität
        {
            return true;
        }
        if (other == null) // 2. Test auf null
        {
            return false;
        }
        if (other.getClass() != this.getClass()) // 3. Test auf Vergleichbarkeit
        {
            return false;
        }
        if (this.getVolume()!=((Cube)other).getVolume()) // 4. Vergleich relev. Felder
        {
            return false;
        }
        else
        {
            return true;
        }
    }

    public int hashCode()
    {
        return getVolume();
    }

    public int compareTo(Cube c)
    {
        /* Compares this object with the specified object for order.
         * Returns a negative integer, zero, or a positive integer as this
         * object is less than, equal to, or greater than the specified
         * object.
         */

        int returnvalue = 0;

        if(this == c)
        {
            returnvalue = 0;
        }
        else if(this.getVolume() > c.getVolume())
        {
            returnvalue = 1;
        }
        else if(this.getVolume() < c.getVolume())
        {
            returnvalue = -1;
        }
        return returnvalue;
    }
}
```

2. Aufgabe: Klasse SolidFileIO

```
import java.io.*;
import java.util.*;

public class SolidFileIO
{
    /* Fortlaufende Nummer, welche dem nächsten
     * eingelesenen Körper zugewiesen wird.
     */
    private static int number;

    public SolidFileIO()
    {

    }

    public static void main(String[] args)
    {
        Cube[] cubes = readSolids("solidworks.txt");
        for(Cube c:cubes)
        {
            System.out.print(c.toString());
        }
    }

    /* Liest aus der TextDatei "soliddata.txt"
     * zeilenweise die Daten ein und liefert
     * schlussendlich ein Array mit allen Kuben zurück
     */
    public static Cube[] readSolids(String fileName)
    {
        StringTokenizer sT;
        Set<Cube> list = new TreeSet<Cube>();

        try
        {
            File f = new File(fileName);
            FileReader aFileReader = new FileReader(f);
            BufferedReader aBufferedReader = new BufferedReader(aFileReader);
            String line = aBufferedReader.readLine();

            while (line != null) // while not EOF
            {
                sT = new StringTokenizer(line, ": ");
                String s = sT.nextToken();
                // System.out.println(s.hashCode()); // debugging purpose
                int s1 = Integer.parseInt(sT.nextToken());
                int s2 = 0;
                int s3 = 0;

                switch(s.hashCode())
                {
                    case 67:
                        s2 = Integer.parseInt(sT.nextToken());
                        s3 = Integer.parseInt(sT.nextToken());
                        list.add(new Cube(number, s1, s2, s3));
                        number++;
                    break;

                    case 89:
                        list.add(new Cylinder(number, s1, s2));
                        s2 = Integer.parseInt(sT.nextToken());
                        number++;
                    break;

                    case 83:
                        list.add(new Sphere(number, s1));
                        number++;
                    break;
                    default: //Unsupported Type
                }

                line = aBufferedReader.readLine();
            }
            aFileReader.close();
        }
        catch(Exception ex)
        {
            //to be implemented
        }

        Cube[] c = new Cube[1];
        return list.toArray(c);
    }
}
```

3. Aufgabe: Klasse SolidWorks

```
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SolidWorks extends JFrame
{
    private JButton btnReadFile;
    private JButton btnSortVolume;
    private JButton btnSortSurface;
    private JButton btnSortDimension;
    private JTextArea txtView;
    private JScrollPane jScrollPane1;
    private Cube[] cubes;

    public SolidWorks()
    {
        super("SolidWorks");
        InitializeComponents();

        setResizable(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        pack();
        setVisible(true);
    }

    private void InitializeComponents()
    {
        //
        // panelMain
        //
        JPanel panelMain = new JPanel();
        getContentPane().add(panelMain);

        //
        // btnReadFile
        //
        btnReadFile = new JButton(btnReadFile_clicked);

        //
        // jScrollPane1
        //
        jScrollPane1 = new JScrollPane();

        //
        // txtView
        //
        txtView = new JTextArea();
        //txtView.setColumns(20);   // nicht zwingend nötig
        //txtView.setRows(10);      // nicht zwingend nötig
        txtView.setName("txtView");
        txtView.setEditable(false);
        jScrollPane1.setViewportView(txtView);
        jScrollPane1.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

        //
        // btnSortVolume
        //
        btnSortVolume = new JButton(btnSortVolume_clicked);

        //
        // btnSortSurface
        //
        btnSortSurface = new JButton(btnSortSurface_clicked);

        //
        // btnSortDimension
        //
        btnSortDimension = new JButton(btnSortDimension_clicked);

        //
        // GroupLayout
        //
        GroupLayout jPanel1Layout = new GroupLayout(panelMain);
        panelMain.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(160, 160, 160)
                        .addComponent(btnReadFile))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(btnSortVolume)
                        .addGap(18, 18, 18)
                        .addComponent(btnSortSurface)
                        .addGap(18, 18, 18)
                        .addComponent(btnSortDimension))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jScrollPane1, GroupLayout.DEFAULT_SIZE, 378,
Short.MAX_VALUE)))
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(btnReadFile)
                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1, GroupLayout.DEFAULT_SIZE, 169,
Short.MAX_VALUE)
                .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                    .addComponent(btnSortVolume)
                    .addComponent(btnSortDimension)
                    .addComponent(btnSortSurface))
                .addContainerGap())
        );
    }

    Action btnReadFile_clicked = new AbstractAction("Read File") {
        public void actionPerformed(ActionEvent evt) {
            getCubes();
            showCubes(null);
        }
    };

    Action btnSortVolume_clicked = new AbstractAction("Sort by Volume") {
        public void actionPerformed(ActionEvent evt) {
            showCubes(null);
        }
    };

    Action btnSortSurface_clicked = new AbstractAction("Sort by Surface") {
        public void actionPerformed(ActionEvent evt) {
            showCubes(new SurfaceComparator());
        }
    };

    Action btnSortDimension_clicked = new AbstractAction("Sort by Dimension") {
        public void actionPerformed(ActionEvent evt) {
            showCubes(new MaxDimensionComparator());
        }
    };

    private void getCubes()
    {
        cubes = SolidFileIO.readSolids("solidworks.txt");
    }

    private void showCubes(Comparator c)
    {
        if(c==null)
        {
            Arrays.sort(cubes);
        }
        else
        {
            Arrays.sort(cubes, c);
        }

        String temp = "";
        for(Cube cube:cubes)
        {
            temp = temp+cube.toString();
        }
        txtView.setText(temp);
    }
}
```

4.1. Aufgabe: Klasse SurfaceComparator

```java
import java.util.Comparator;
public class SurfaceComparator implements Comparator<Cube>
{
    public int compare(Cube c1, Cube c2)
    {
        return (c1.getSurface() - c2.getSurface());
    }

    public boolean equals(Object obj)
    {
        if (obj == null){
            return false;
        }
        return (this.getClass() == obj.getClass());
    }
}
```

4.2. Aufgabe: Klasse MaxDimensionComparator

```java
import java.util.Comparator;
public class MaxDimensionComparator implements Comparator<Cube>
{
    public int compare(Cube c1, Cube c2)
    {
        return (c1.getMaxDimension() - c2.getMaxDimension());
    }

    public boolean equals(Object obj)
    {
        if (obj == null){
            return false;
        }
        return (this.getClass() == obj.getClass());
    }
}
```

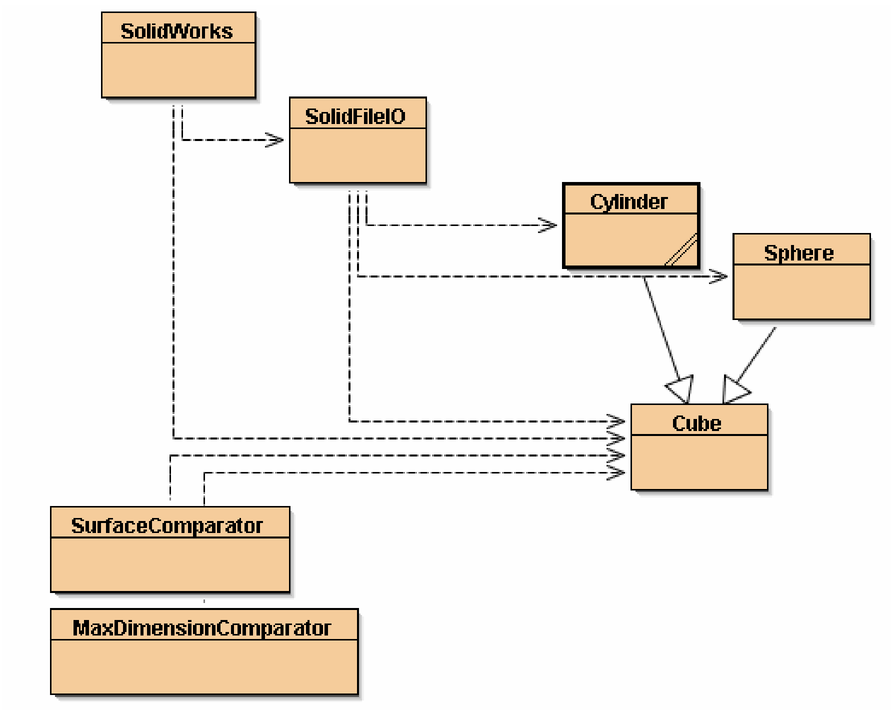
5.1. Aufgabe (Optional) : Klasse Sphere

```java
public class Sphere extends Cube
{
    public static void main(String[] args)
    {
        Sphere s = new Sphere(1,10);
        System.out.print(s.toString());
    }

    public Sphere(int no, int d) //d=Durchmesser
    {
        super(no, d, 0, 0);
    }

    // Liefert einen String, welcher den Zylinder beschreibt
    public String toString()
    {
        return "--------------------------\n\rSPHERE: "+number+
        "\n\rDiameter: "+s1+"\n\rVolume: "+getVolume()+
        "\n\rSurface: "+getSurface()+"\n\r";
    }

    // Berechnet das Volumen der Kugel
    public int getVolume()
    {
        return (int)(4/3)*(int)(Math.PI)*(int)(Math.pow(s1/2,3));
    }
    // Berechnet die Oberfläche der Kugel
    public int getSurface()
    {
        return 4*(int)(Math.PI)*(int)(Math.pow(s1/2,2));
    }
}
```

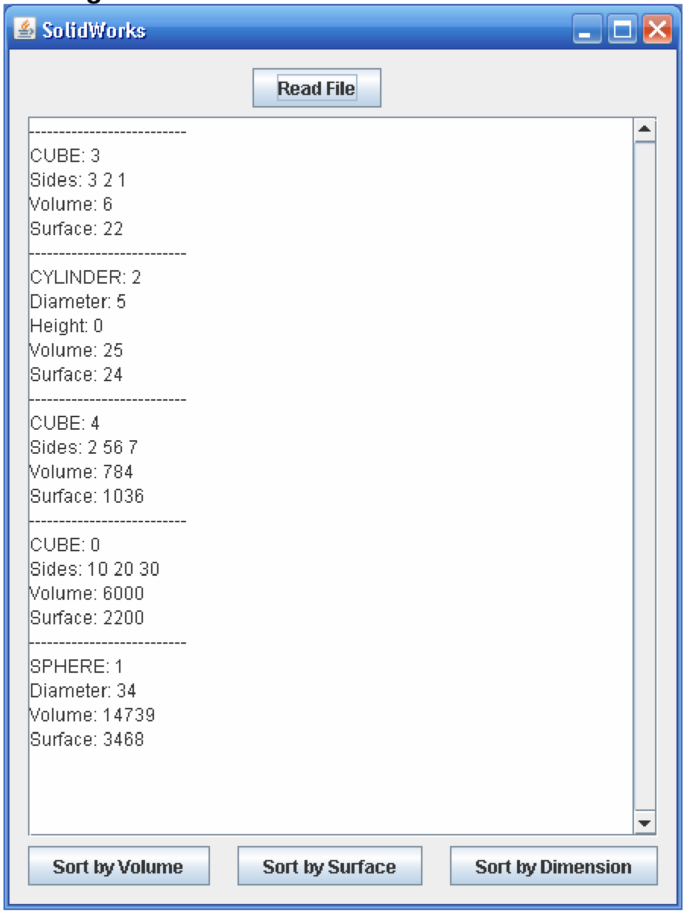
5.2. Aufgabe (Optional) : Klasse Cylinder

```
public class Cylinder extends Cube
{
    public static void main(String[] args)
    {
        Cylinder y = new Cylinder(1,10,20);
        System.out.print(y.toString());
    }

    public Cylinder(int no, int d, int h) //d=Durchmesser, h=Höhe
    {
        super(no, d, h, 0);
    }

    // Liefert einen String, welcher den Zylinder beschreibt
    public String toString()
    {
        return "--------------------------\n\rCYLINDER: "+number+
        "\n\rDiameter: "+s1+
        "\n\rHeight: "+s2+
        "\n\rVolume: "+getVolume()+
        "\n\rSurface: "+getSurface()+"\n\r";
    }

    // Berechnet das Volumen des Zylinders
    public int getVolume()
    {
        return (int)(2*Math.PI*Math.pow(s1/2,2));
    }

    // Berechnet die Oberfläche des Zylinders
    public int getSurface()
    {
        return (int)(2*Math.PI)*(int)(Math.pow(s1/2,2))*s2 +
(int)(2*Math.PI)*(int)(Math.pow(s1/2,2));
    }
}
```

Anhang 1: Klassendiagramm



Anhang 2: GUI



```
---------------------------
CUBE: 3
Sides: 3 2 1
Volume: 6
Surface: 22
---------------------------
CYLINDER: 2
Diameter: 5
Height: 0
Volume: 25
Surface: 24
---------------------------
CUBE: 4
Sides: 2 56 7
Volume: 784
Surface: 1036
---------------------------
CUBE: 0
Sides: 10 20 30
Volume: 6000
Surface: 2200
---------------------------
SPHERE: 1
Diameter: 34
Volume: 14739
Surface: 3468
```