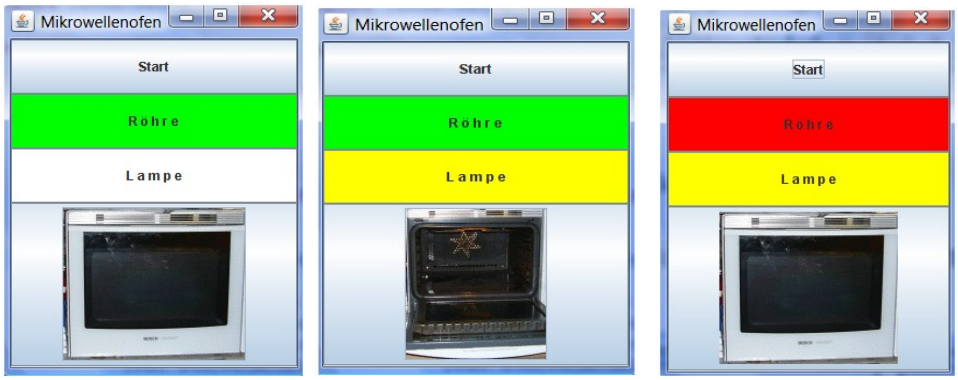
Gegeben ist die Anwendung zur Steuerung eines Mikrowellenofens nach folgendem Beispiel:



Die Mikrowelle hat verschiedene Zustände:

- ist die Tür geschlossen, so kann der Startknopf betätigt werden, die Lampe geht an und die Mikrowelle heizt für 6s auf. Danach gehen Röhre und Lampe wieder aus.
- Wird die Tür geöffnet, so kann die Welle nicht gestartet werden bzw. wird ein laufender Heizvorgang unterbrochen. Die Lampe geht an.

Die Anwendung ist als BLOB (binary large object) programmiert, in einer etwas unstrukturierten Form aufgeschrieben und besteht aus den beiden folgenden Dateien:

```
public class OvenGui {

    public static void main(String[] args) {
        JFrame mainWin = new OvenBlob();

        mainWin.setVisible(true);
        mainWin.setTitle("Mikrowellenofen");
        mainWin.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
    }
}

public class OvenBlob extends JFrame {

    private final JButton startButton = new JButton("Start");
    private final JButton door = new JButton();
    private boolean isDoorOpen  = false;
    private JButton tube = new JButton();
    private final JButton lamp = new JButton();
    private Timer timer;

    public OvenBlob() {
        initComponents();
        pack();
    }

    private void initComponents() {
        door.setIcon(new ImageIcon("closed.gif"));
        isDoorOpen = false;
        tube.setBackground(Color.GREEN);
        tube.setText("R ö h r e");
        lamp.setText("L a m p e");
        add(createOvenPanel());
    }

    private Component createOvenPanel() {
        JPanel subPanel = new JPanel(new GridLayout(3, 1));
        JPanel mainPanel = new JPanel(new GridLayout(2, 1));

        subPanel.add(startButton);
        subPanel.add(tube);
        subPanel.add(lamp);
        mainPanel.add(subPanel);
        mainPanel.add(door);
        timer = new Timer(6000, new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                tube.setBackground(Color.GREEN);
                lamp.setBackground(Color.WHITE);
```

```
            }});
        startButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                if(!isDoorOpen) {
                    lamp.setBackground(Color.RED);
                    tube.setBackground(Color.YELLOW);
                    timer.start();
                }
            }});
        door.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                if(!isDoorOpen) {
                    door.setIcon(new ImageIcon("open.gif"));
                    lamp.setBackground(Color.RED);
                    tube.setBackground(Color.GREEN);
                    timer.stop();
                    isDoorOpen = true;
                }
                else {
                    door.setIcon(new ImageIcon("closed.gif"));
                    lamp.setBackground(Color.WHITE);
                    isDoorOpen = false;
                }
            }});
        return mainPanel;
    }
}
```

Aufgaben

1. Problemanalyse
   - Machen Sie sich anhand geeigneter Quellen mit dem MVC-Entwurfsmuster vertraut (z.B.: unter http://de.wikipedia.org/wiki/Model_View_Controller
   - Ändern Sie die Anwendung so ab, dass die Klasse OvenBlob entsprechend dem MVC-Konzept komplett ersetzt wird durch die drei Klassen OvenView, OvenController und OvenModel. Alle Funktion/Methoden sollen entsprechend der Regeln, die auch für die Kapselung wichtig sind, den neuen Klassen zugeordnet werden.
   - Suchen und beheben Sie evtl. noch vorhandene Fehler. Setzen sie ggf. Methoden des Unit-Testings für einen fehlerfreien Ablauf ein.
   - Erstellen Sie ein entsprechendes Klassendiagramm für Ihre Anwendung.
2. Expertenrunde:
   - Fassen Sie Ihre Beobachtungen zusammen und halten Sie fest, wie das MVC-Muster implementiert wird.

Zusatzaufgabe:

Implementieren Sie in Ihre Lösung das Observer-Pattern zur Benachrichtung mehrerer Views bei Veränderungen im Model.