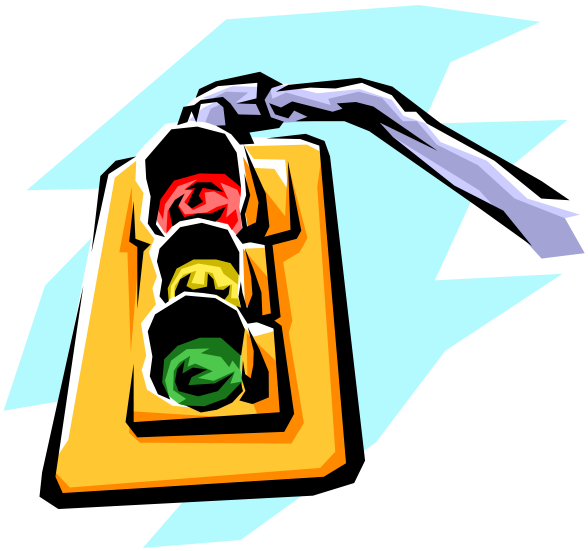
Softwarepraktikum

Teil: Eingebettete Systeme

Sommersemester 2003

Implementierung III:

GUI und Verhalten (Teil 2)



Aufgabe 5

Implementierung III: GUI und Verhalten (Teil 2)

Umfang: **1 Woche**

Punkte: **50 P.**

Als Nächstes soll die Implementierung des Verhaltens abgeschlossen werden und eine grafische Benutzersschnittstelle (Graphical User Interface, kurz GUI) entwickelt werden. Diese soll Fehlerzustände und statistische Daten der aktuellen Verkehrssituation darstellen, sowie Eingriffe in die Steuerung erlauben.

1 Konzepte

1.1 Model-View-Controller

Da erfasste Daten auf verschiedene Arten dargestellt und geändert werden können, ist es sinnvoll, eine Benutzerschnittstelle in drei Teile aufzuteilen. Ein Teil beinhaltet dabei die eigentlichen Daten und die Kernfunktionalität, ein weiterer kümmert sich um die Darstellung der Daten und ein dritter Teil verwaltet die Benutzereingaben. Diese Dreiteilung wird auch als „**Model-View-Controller**“ bezeichnet. Dabei verwalten die **Model**-Klassen die eigentlichen Daten. Diese Daten können über einen **Controller** geändert werden. Solch eine Änderung hat in der Regel eine Aktualisierung der **Views**

zur Folge. Folgende Abbildung (Abbildung 23) verdeutlicht das prinzipielle Vorgehen:

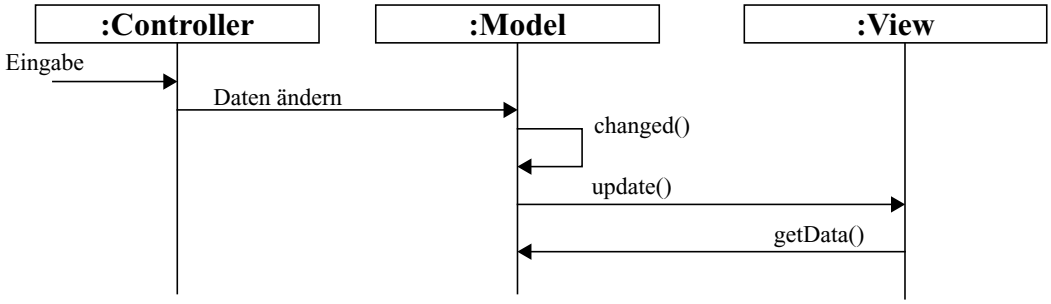


Abbildung 23 Model-View-Controller: Verteilung der Verantwortlichkeiten

Wird in einer Eingabemaske der grafischen Benutzerschnittstelle eine Änderung vorgenommen, so muss das zu Grunde liegende Modell geändert werden. Daraufhin weden diejenigen Views, die Daten des Modells anzeigen, aktualisiert. Damit der Mechanismus funktioniert, wird das Observer-Pattern (vgl. vorige Aufgabe) eingesetzt. Insbesondere die Views müssen sich beim Modell als Listener registrieren (im Sequenzdiagramm in Abbildung 23 ist die typische Funktionalität des Observables Modell zu erkennen).

Bei Java stellen die Java Foundation Classes (JFC/Swing) bereits ein umfangreiches Framework zur Implementierung grafischer Benutzerschnittstellen zur Verfügung. Auch hier kommt das MVC-Prinzip zum Einsatz, wobei aber häufig Controller und View zusammen in einer Klasse realisiert werden (siehe dazu [SwingMVC]).

1.2 Beispiel

Folgendes Beispiel zeigt, wie eine Benutzeroberfläche in Java realisiert werden kann:

Die Daten werden in Objekten der Klasse `MyModel` gehalten. Bei jeder Änderung der Daten, werden alle angeschlossenen Views aktualisiert.

```
import java.util.*;

public class MyModel extends Observable {
    private int numClicks = 0;
```

```
    public int getClicks() {
        return numClicks;
    }

    public void clicked() {
        numClicks++;
        // Modell hat sich geaendert
        setChanged();
        // Observer benachrichtigen
        notifyObservers();
    }
}
```

Die Klasse `MyView` stellt einen Button und Label dar. Das Label zeigt an, wie oft auf den Button gedrückt wurde. Dabei wird die Anzahl der Clicks im Modell abgelegt. Der Controller, der auf die Button-Clicks reagiert, ist hierbei in den View integriert (siehe Kommentare).

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

public class MyView extends JPanel implements Observer {
    private static String labelPrefix = "Number of clicks: ";
    private JLabel label;
    private MyModel model;

    public MyView(MyModel newModel) {
        super();
        model = newModel;
        // View als Observer des Modells registrieren
        model.addObserver(this);

        // Text-label
        label = new JLabel(labelPrefix + "0    ");
        // Button
        JButton button = new JButton("Click me!");
        // Controller fuer den Button anlegen
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Modell aktualisieren (dadurch wird auch der View
                // neu gezeichnet).
                model.clicked();
            }
        });
        label.setLabelFor(button);

        // Ein Rahmen um den Inhalt
        setBorder(BorderFactory.createEmptyBorder(10,10,0,10));
        // Layout (Gitter mit einer Spalte)
        setLayout(new GridLayout(0, 1));

        add(button);
        add(label);
    }

    public void update (Observable o, Object arg) {
        int numClicks = model.getClicks();
        label.setText(labelPrefix + numClicks);
    }
}
```

Zuletzt benötigt man eine Komponente, die das Modell und den oder die Views verbindet und einen Rahmen für das gesamte GUI darstellt:

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

public class GUI {
    public static void main(String args[]) {
        // Frame anlegen als Top-Level-Container
        JFrame frame = new JFrame("Hallo Welt");

        // Modell anlegen
        MyModel model = new MyModel();
        // View-Komponente anlegen...
        MyView view = new MyView(model);
        // ...und dem Fenster hinzufuegen
        frame.getContentPane().add(view, BorderLayout.CENTER);

        // Programm beenden wenn das Fenster geschlossen wird
        // (Controller)
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        // und alles darstellen
        frame.pack();
        frame.setVisible(true);
    }
}
```

Weitere Informationen zur GUI-Programmierung finden sich unter [SwingTutorial].

2 GUI

Folgende Daten sollen auf der GUI dargestellt werden:

- Verkehrsdurchsatz aller Kreuzungen zusammen (d.h. eingefahrene Fahrzeuge, Fahrzeuge die das Kreuzungssystem verlassen haben und die Differenz),
- Verkehrsdurchsatz entlang der grünen Welle (also nur Fahrzeuge betrachten, die bei LSA 7 oder 10 einfahren und bei LSA 7 oder 12 ausfahren),
- Fehlerzustände und
- eine Möglichkeit, um die Steuerung zu beenden und das GUI zu verlassen.

Beim Entwurf des GUI sollen die folgenden Schritte berücksichtigt werden:

- Aufbau und Funktionsumfang des GUI festlegen, evtl. Skizze der Oberfläche.

- Implementierung der Statistik-Funktionen.
- Implementierung und Test des GUI als Java-Programm.

3 Verhalten (2)

Parallel zum Entwurf der grafischen Benutzerschnittstelle soll nun auch die Implementierung der Zustandsdiagramme fertiggestellt werden. Setzen Sie die noch fehlenden Zustandsdiagramme in Java-Kode um. Werden zur Kommunikation zwischen den Zustandsdiagrammen weitere Events benötigt, so implementieren Sie diese (durch Spezialisierung der `NamedEventObject`-Klasse) und koppeln Sie Ihre Zustandsdiagramme unter Verwendung des Observer-Patterns.

Protokollieren Sie weiter wichtige Testläufe. Testen Sie Ihre Implementierung und vergleichen Sie die Ergebnisse mit den vorher modellierten Sequenz-Diagrammen.

4 Aufgaben (50 P.)

Abzugeben sind

- die Java-Sourcen für alle implementierten Zustandsdiagramme und für die GUI (inkl. vernünftigen javadoc-Kommentaren)
- die generierten javadoc-Dokumente (vollständig und ausführlich).
- (mind. 10) protokollierte Testläufe (z.B. Mitprotokollieren der Standardausgabe und Instrumentierung des Java-Programms mit `System.out.println`). Dabei soll nachgewiesen werden, dass die Szenarien aus Aufgabenteil 1 korrekt ablaufen können.

5 Literatur

[SwingTutorial] *Creating a GUI with JFC/Swing.* The Java Tutorial.
—Link ist auf der Praktikums-Site vorhanden

[SwingMVC] *Introducing Swing Architecture.* The Swing Connection.
—Link ist auf der Praktikums-Site vorhanden