Design Pattern: Model-View-Controller

1 Designprinzip

- Model: verwaltet Daten, benachrichtigt View (Observer) bei Änderung
- View: Beobachter von Model (Anzeige der Daten), verwendet Controller als Strategie (Benutzeraktionen)
- Controller: verarbeitet Benutzeraktionen und leitet sie an Model weiter, steuert View (Steuerelemente deaktivieren)

2 Java-Code

2.1 Modelschnittstelle

```
public interface ModelSchnittstelle {
        // View (Beobachter) muss sich registrieren
        void registrieren(ViewSchnittstelle view);

        // Methoden zum Lesezugriff auf Daten fuer View
        int getDaten();

        // Methoden zum Schreibzugriff fuer den Controller
        void aendern();
}
```

2.2 Viewschnittstelle

```
public interface ViewSchnittstelle {
        // Diese Methode ruft das Model (Subject) bei Aenderung der Daten auf
        void update();

        // Diese Methoden kann der Controller aufrufen
        // z.B. GUI erstellen,
        // Steuerelemente deaktivieren
        void fensterErstellen();
}
```

2.3 Controllerschnittstelle

```
public interface ControllerSchnittstelle {
        // Diese Methoden kann die View aufrufen
        // Benutzeraktionen werden von View an Controller uebermittelt
        void aendern();
}
```

2.4 Hauptprogramm

```
ModelSchnittstelle model = new KonkretesModel();
ControllerSchnittstelle controller = new KonkreterController(model);
```

2.5 Konkretes Model

```
public class KonkretesModel implements ModelSchnittstelle {
        // Model kennt nur Referenzen von allen Views (Observer)
        // Model kennt Controller nicht
        private ArrayList<ViewSchnittstelle> beobachterliste
                                = new ArrayList<ViewSchnittstelle>();

        // Vom Model selbst verwaltete Daten
        private int daten;
        private Random random = new Random();

        // Views (Observer) muessen sich registrieren
        @Override
        public void registrieren(ViewSchnittstelle view) {
                beobachterliste.add(view);
        }

        // Views (Observer) werden ueber Aenderung der Daten benachrichtigt
        private void broadcast(){
                for(int i=0; i<beobachterliste.size(); i++){
                        beobachterliste.get(i).update();
                }
        }

        // Lesezugriff von aussen auf Daten (fuer Views, Observer)
        @Override
        public int getDaten() {
                return daten;
        }

        // Schreibzugriff von aussen auf Daten (fuer Controller)
        @Override
        public void aendern() {
                daten = random.nextInt();
                // bei Aenderung der Daten alle Views benachrichtigen
                broadcast();
        }
}
```

2.6 Konkrete View

```
public class View implements ViewSchnittstelle{
        // View holt sich Daten vom Model
        private ModelSchnittstelle model;

        // View uebergibt Benutzeraktionen an Controller
        private ControllerSchnittstelle controller;

        // GUI-Elemente
        private JButton button;


        public View(ControllerSchnittstelle controller,
                        ModelSchnittstelle model) {
                // View kennt Controller zur Uebergabe von Aktionen
                this.controller = controller;

                // View kennt Model (Subject) zum Empfangen von Aenderungsmeldungen
                // und zum Holen der Daten
                this.model = model;
                model.registrieren(this);

        }

        // wird vom Model (Subject) aufgerufen
        @Override
        public void update() {
                // View holt Daten selbst (PULL)
                int daten = model.getDaten();
                button.setText("Zahl: " + daten);
        }

        // GUI
        @Override
        public void fensterErstellen() {
                JFrame frame = new JFrame();
                frame.setSize(200,200);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                button = new JButton("Drueck");
                frame.add(button);
                button.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                // Benutzeraktion wird an Controller uebermittelt
                                controller.aendern();
                        }
                });
                frame.setVisible(true);
        }
}
```

2.7 Konkreter Controller

```
public class KonkreterController implements ControllerSchnittstelle {
        // Controller ist Vermittler zwischen View und Model
        private ModelSchnittstelle model;
        private View view;

        // Controller erstellt View
        public KonkreterController(ModelSchnittstelle model) {
                this.model = model;
                view = new View(this,model);
                view.fensterErstellen();
        }

        // Controller verarbeitet Nachrichten von View
        // und gibt sie an Model weiter
        @Override
        public void aendern() {
                model.aendern();
        }
}
```

Literatur

[Head First Design Patterns] `http://www.oreilly.de/catalog/9780596007126/`

[Wikipedia] `http://de.wikipedia.org/wiki/Model_View_Controller`