

Model-View-Controller



Model – View – Controller (MVC)

Architekturmuster für die Gestaltung von interaktiven Oberflächen

3 Komponenten

- *Model*: Datenmodell
- *View*: Ansicht
- *Controller*: Interaktion

Bei Java AWT und Swing sind View und Controller integriert

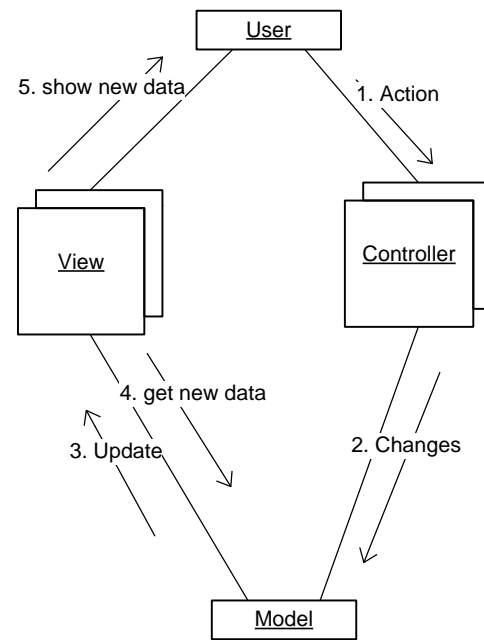
Motivation:

- Datenmodell unabhängig von Views und Controller
- Mehrere Views und Controller für ein Datenmodell



MVC Ablauf

- Aktion durch Benutzer
- Controller reagiert und Daten werden geändert
- Model verständigt alle Views, dass sich Daten geändert haben
- View holen sich die neuen Daten vom Model
- Views stellen die neuen Daten dar



Ereigniskonzept in Java (aus: JavaBeans Specification)

Quelle

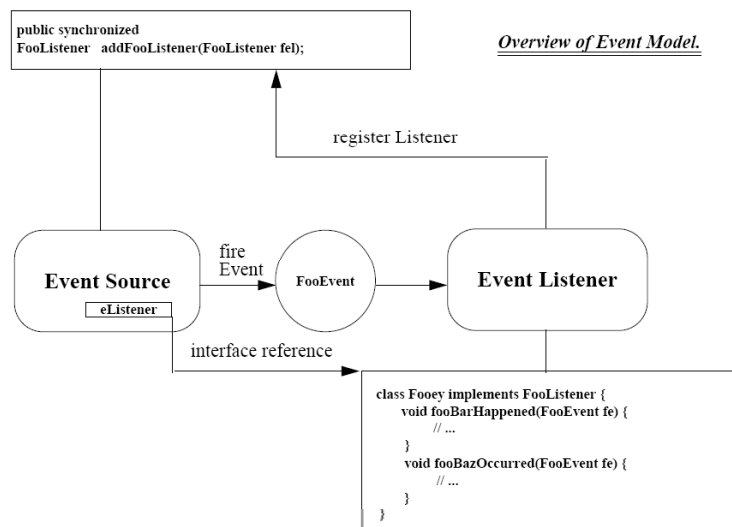
- Auslöser eines Ereignisses
- benachrichtigt Interessierte

Interessierter (Listener)

- reagiert auf Ereignis
- wird bei Quelle registriert und deregistriert

Ereignisobjekt (EventObject)

- Informationen über Ereignis



Design Pattern für Ereignisse

Listener

- Interface welche Ereignismethoden implementiert
- *muss* `java.util.EventListener` erweitern

```
public interface TimerListener extends java.util.EventListener {  
    public void timerAction(TimerEvent e);  
}
```

Quelle

- implementiert Methoden für das Anfügen und Entfernen von Listener
`public void addListenerType(ListenerType listener)`
`public void removeListenerType(ListenerType listener);`

```
public class TimerBean {  
    public void addTimerListener(TimerListener l) { ... }  
    public void removeTimerListener(TimerListener l) { ... }  
  
    private void fireTimerEvent(TimerEvent e) { ... }  
    ...  
}
```

Design Pattern für Ereignisse (Fortsetzung)

Ereignisobjekt

- *erweitert* `java.util.EventObject`
- *kommuniziert* Informationen über Ereignis

```
public class TimerEvent extends java.util.EventObject {  
    private long time;  
  
    public TimerEvent(Object source) {  
        super(source);  
        time = System.currentTimeMillis();  
    }  
  
    public long getTime() {  
        return time;  
    }  
}
```

MVC in Java

Java MVC arbeitet stark mit Ereignissen

Controller:

- sendet AWT Events bei Benutzeraktionen
- Eventhandler implementieren Datenänderungen

Model:

- meldet Datenänderungen mit Ereignissen
- Model stellt Quelle der Ereignisse dar (hat Methoden zum An-/Abmelden der Listener)
- meist eigenes Listener-Interface und EventObject

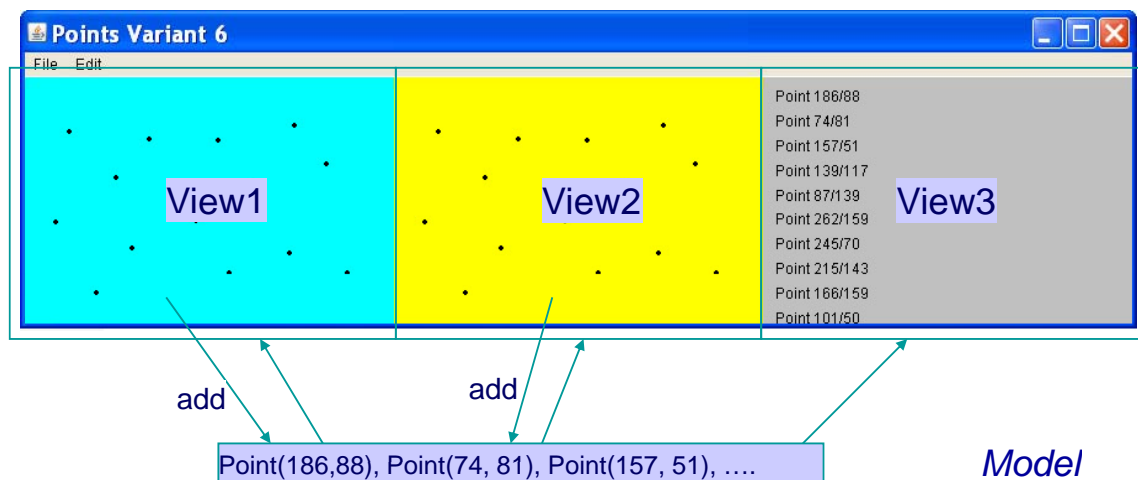
Views:

- Implementieren die Listener-Interfaces
- melden sich beim Model als Listener an
- reagieren auf die gemeldeten Datenänderungen mit neuer Darstellung



Beispiel Points: Architektur

Mehrere Views und Editors für ein Modell



Beispiel Points: PointListener und PointEvent

PointListener: Listener-Interface für Änderungen bei Points

```
public interface PointListener extends java.util.EventListener {  
    public void pointChanged(PointEvent event);  
}
```

PointEvent: Event-Object mit Informationen über Ereignis

```
import java.awt.Point;  
import java.util.EventObject;  
  
public class PointEvent extends EventObject {  
    private Point point; private String eventName;  
  
    public PointEvent(Object source, String eventName, Point point) {  
        super(source); this.point = point; this.eventName = eventName;  
    }  
    public Point getPoint() { return point; }  
  
    public String getEventName() { return eventName; }  
}
```



Beispiel Points: PointModel

Datenobjekt PointModel:

```
public class PointModel {  
    private List<Point> points = new ArrayList<Point>();  
    private List<PointListener> listeners = new ArrayList<PointListener>();  
}
```

Methoden für Datenänderungen

```
public void clear() {  
    points.clear();  
    firePointEvent("removed all", null);  
}  
public void add(Point p) {  
    points.add(p);  
    firePointEvent("added", p);  
}  
...  
}
```

Datenzugriff

```
public Point[] getPoints() {  
    return points.toArray(new Point[0]);  
}  
...  
}
```



Beispiel Points: PointModel (2)

An-/Abmelden der Listener und Feuern der Ereignisse

```
public void addPointsListener(PointListener l) {
    listeners.add(l);
}

public void removePointsListener(PointListener l) {
    listeners.remove(l);
}

private void firePointEvent(String eventName, Point p) {
    PointEvent evt = new PointEvent(this, eventName, p);
    for (PointListener pointL: listeners) {
        pointL.pointChanged(evt);
    }
}
}
```



Beispiel Points: PointCanvas

PointCanvas und reagiert auf Mouse-Clicks und führt repaint durch

```
public class PointCanvas extends Canvas {
    private final PointModel pointsModel;
    public PointCanvas(PointModel points) {
        this.pointsModel = points;
        pointsModel.addPointsListener(pointsChangedHandler);
        this.addMouseListener(mouseClickHandler);
    }
    public void paint(Graphics g) {
        super.paint(g);
        for (Point p: pointsModel.getPoints()) {
            g.fillOval(p.x-2, p.y-2, 4, 4);
        }
    }
    private MouseListener mouseClickHandler =
        new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                pointsModel.add(new Point(e.getX(), e.getY()));
            }
        };
    private PointListener pointsChangedHandler = new PointListener() {
        @Override public void pointChanged(PointEvent event) {
            repaint();
        }
    };
}
```



Beispiel Points: PointTextPanel

PointTextPanel gibt Points textuell aus; PointsListener führt repaint durch

```
public class PointTextPanel extends Panel {  
  
    PointModel pointsModel;  
  
    public PointTextPanel(PointModel points) {  
        this.pointsModel = points;  
        pointsModel.addPointsListener(pointsChangedHandler);  
    }  
  
    public void paint(Graphics g) {  
        super.paint(g);  
        int line = 20;  
        for (Point p: pointsModel.getPoints()) {  
            g.drawString("Point "+p.x+"/"+p.y, 10, line);  
            line = line + 20;  
        }  
    }  
  
    private PointListener pointsChangedHandler = new PointsListener() {  
        public void pointChanged(PointEvent event) {  
            repaint();  
        }  
    };  
}
```



Beispiel Points: Frame mit 3 Views

```
public class PointFrame extends Frame {  
    private PointModel pointModel;  
    private PointCanvas panel1, panel2;  
    private PointTextPanel textPanel;  
  
    public PointFrame(PointModel model) {  
        ...  
        this.pointModel = model;  
        MenuItem clear = new MenuItem("Clear");  
        editMenu.add(clear);  
        clear.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                pointModel.clear();  
            }  
        });  
        ...  
        canvas1 = new PointCanvas(this.pointModel);  
        canvas1.setBackground(Color.CYAN);  
        add(canvas1);  
  
        canvas2 = new PointCanvas(this.pointModel);  
        canvas2.setBackground(Color.YELLOW);  
        add(canvas2);  
  
        textPanel = new PointTextPanel(this.pointModel);  
        textPanel.setBackground(Color.LIGHT_GRAY);  
        add(textPanel);  
    }  
}
```



Model

MenuItem "Clear"

PointCanvas canvas1

PointCanvas canvas2

PointTextPanel



Beispiel Points: Ablauf

