

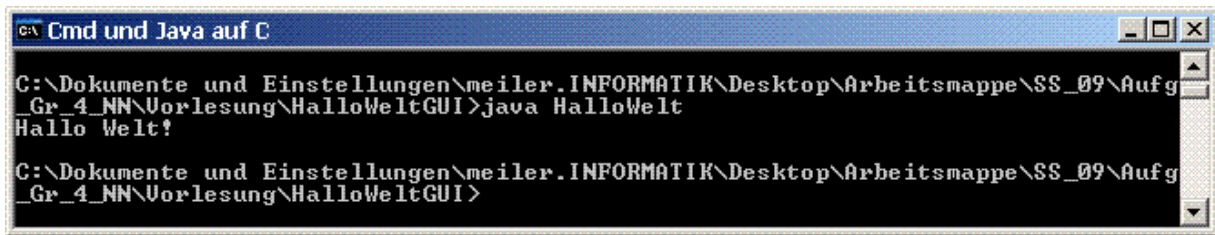
## Einführung zur Aufgabengruppe 4

Alle Literaturhinweise beziehen sich auf die Vorlesung WS 10  
[Modellierung und Programmierung 1](#)

<b>1</b>	<b>Grafische Benutzeroberflächen (s. Kapitel GUI)</b> .....	<b>1-2</b>
1.1	<i>Oberflächenaufbau und Ereignisverarbeitung</i> .....	1-2
1.2	<i>Grafik</i> .....	1-3
<b>2</b>	<b>MVC-Architektur (s. Kapitel MVC)</b> .....	<b>2-4</b>
2.1	<i>MVC-Architektur und Überwachungsmechanismus</i> .....	2-4
2.2	<i>Einige Anwendungen aus Vorlesung und Praktikum</i> .....	2-6
<b>3</b>	<b>Projekte mit mehreren Benutzeroberflächen</b> .....	<b>3-8</b>
3.1	<i>Klassendiagramm mit eingebundenen MVC-Tool</i> .....	3-8
3.2	<i>Paket <code>Tools.MVC</code></i> .....	3-9
3.3	<i>Beispiel „Ampelsteuerung“ mit zwei View/Controller</i> .....	3-9
<b>4</b>	<b>Zusammenfassung</b> .....	<b>4-11</b>

# 1 Grafische Benutzeroberflächen (s. Kapitel [GUI](#))

## Beispiel „Hallo Welt!“, Konsolenausgabe



```
Cmd und Java auf C
C:\Dokumente und Einstellungen\meiler.INFORMATIK\Desktop\Arbeitsmappe\SS_09\Aufg_Gr_4_NN\Vorlesung\HalloWeltGUI>java HalloWelt
Hallo Welt!
C:\Dokumente und Einstellungen\meiler.INFORMATIK\Desktop\Arbeitsmappe\SS_09\Aufg_Gr_4_NN\Vorlesung\HalloWeltGUI>
```

[HalloWelt.java](#)

## 1.1 Oberflächenaufbau und Ereignisverarbeitung

### *Oberflächenaufbau*

Der Aufbau einer Oberfläche (**GUI** *Graphical User Interface*) erfolgt in *Containern* nach einem hierarchischen Baukastensystem. Aus einer vorgegebenen Menge sogenannter *Komponenten* werden die Oberflächen zusammengesetzt. Von diesen Komponenten können einige wieder als Container verwendet werden, d.h. diese können selbst Bausteine aufnehmen.

Java bietet zwei Pakete **java.awt** und **javax.swing**, mit deren Hilfe man grafische Benutzeroberflächen entwickeln kann.

**AWT** (*Abstract Windowing Toolkit*) gehört seit JDK Version 1.0 zum Lieferumfang und ist in den aktuellen Java Versionen enthalten.

**Swing** ist eine Weiterentwicklung des AWT, ist einfacher zu programmieren und erzeugt sehr komfortable Oberflächen, deren Aussehen plattformunabhängig und während der Laufzeit veränderbar sind.

### Beispiel „Hallo Welt!“ ohne Benutzereingaben



[HalloWeltGUI1.java](#)

### *Ereignisverarbeitung*

Die Klassen der Ereignisse (*Events*) und ihren Empfängern (*Listener*) befinden sich im Paket **java.awt.event**.

### Beispiel „Hallo Welt!“ mit Benutzereingaben

Button „Set“ ändert zufällig die Hintergrundfarbe, Button „Quit“ beendet das Programm



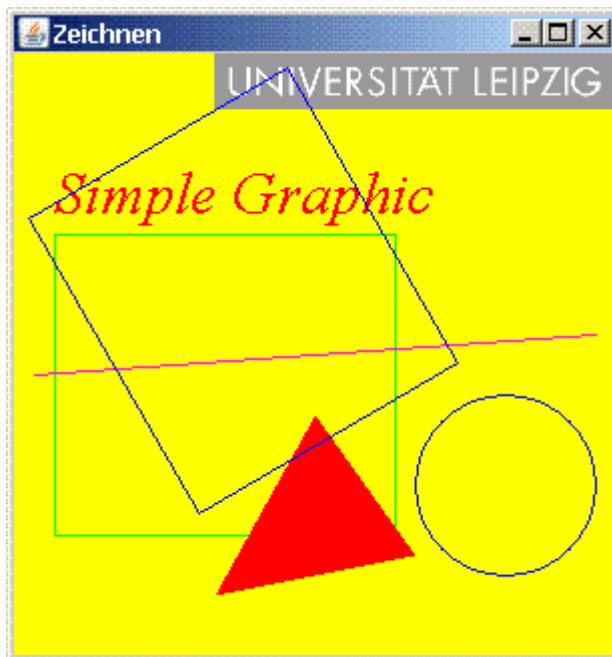
[HalloWeltGUI2.java](#)

## 1.2 Grafik

Ein **Repaint-Manager** ist für die Aktualisierung der Benutzeroberfläche beim erstmaligen Aufbau und bei Änderungen des Erscheinungsbildes zuständig. In diesen Fällen wird eine Methode **repaint** aufgerufen, die die Aktualisierung des Fensters durch den Aufruf einer Methode **paint** für jede Komponente des Fensters veranlasst. Die Methode **paint** bekommt eine Instanz der Klasse **Graphics** übergeben. **Graphics** ist Javas Implementierung eines *Devicekontexts* (auch *Grafikkontext* genannt) und stellt die Abstraktion eines *universellen Ausgabegeräts* für Grafik und Schrift dar. Die Klasse befindet sich im Paket **java.awt** und bietet Methoden zur Erzeugung von Linien-, Füll- und Textelementen, verwaltet die Zeichenfarbe, Fonts zur Ausgabe von Schrift u. a.

Eigene Grafiken kann man in einem entsprechenden Container durch die Methode **paint** erzeugen und bei Bedarf aktualisieren.

### Beispiel „Simple Graphic“

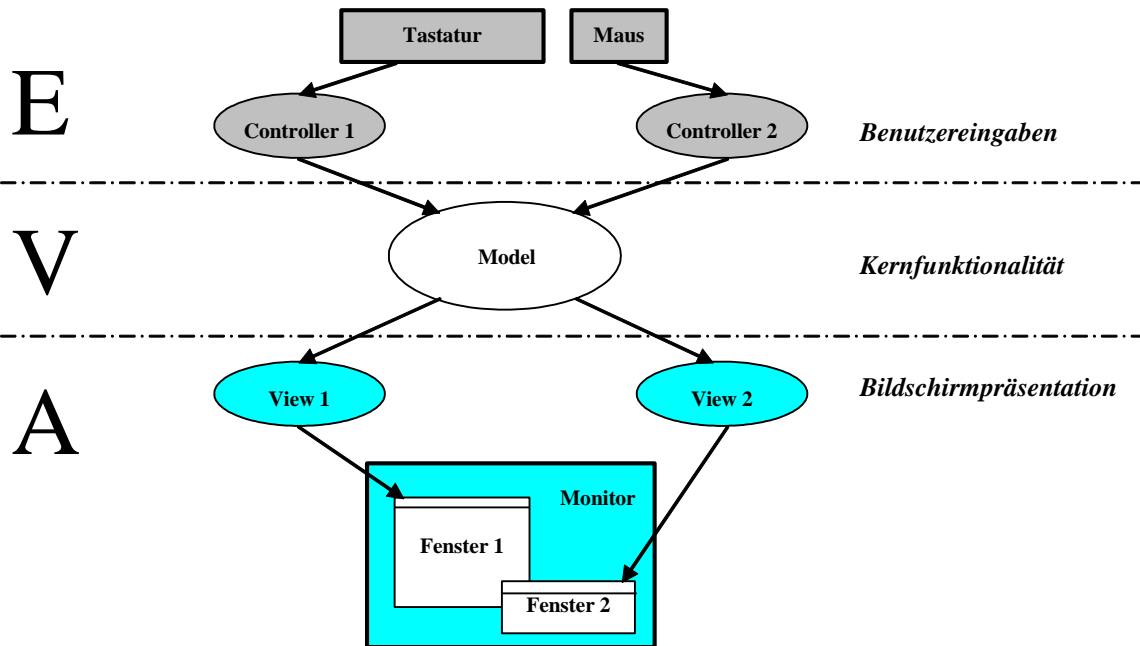


[SimplePanel.java](#)  
[SimpleGrafik.java](#)

## 2 MVC-Architektur (s. Kapitel [MVC](#))

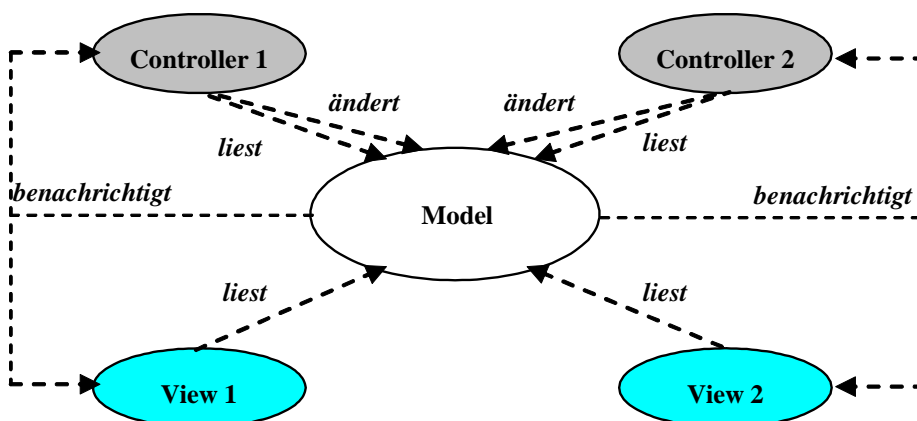
### 2.1 MVC-Architektur und Überwachungsmechanismus

Ein Entwurfsmuster (**design pattern**) zur Konstruktion von Benutzerschnittstellen ist die bereits mit Smalltalk eingeführte **MVC-Architektur** (*M*.. *Model*, *V*.. *View*, *C*.. *Controller*). Ziel dieser Architektur ist die Trennung der *Verarbeitung* eines Problems (**Model**) von dessen *Präsentation* (**View**) und von der *Manipulation* (**Controller**) der Anwendungsdaten durch Benutzereingaben.



Durch diese Architektur ist es möglich, zu einem **Model** *mehrere View's* und *mehrere Controller* zur Verfügung zu stellen. In der Regel hat jeder **View** einen spezifischen **Controller**.

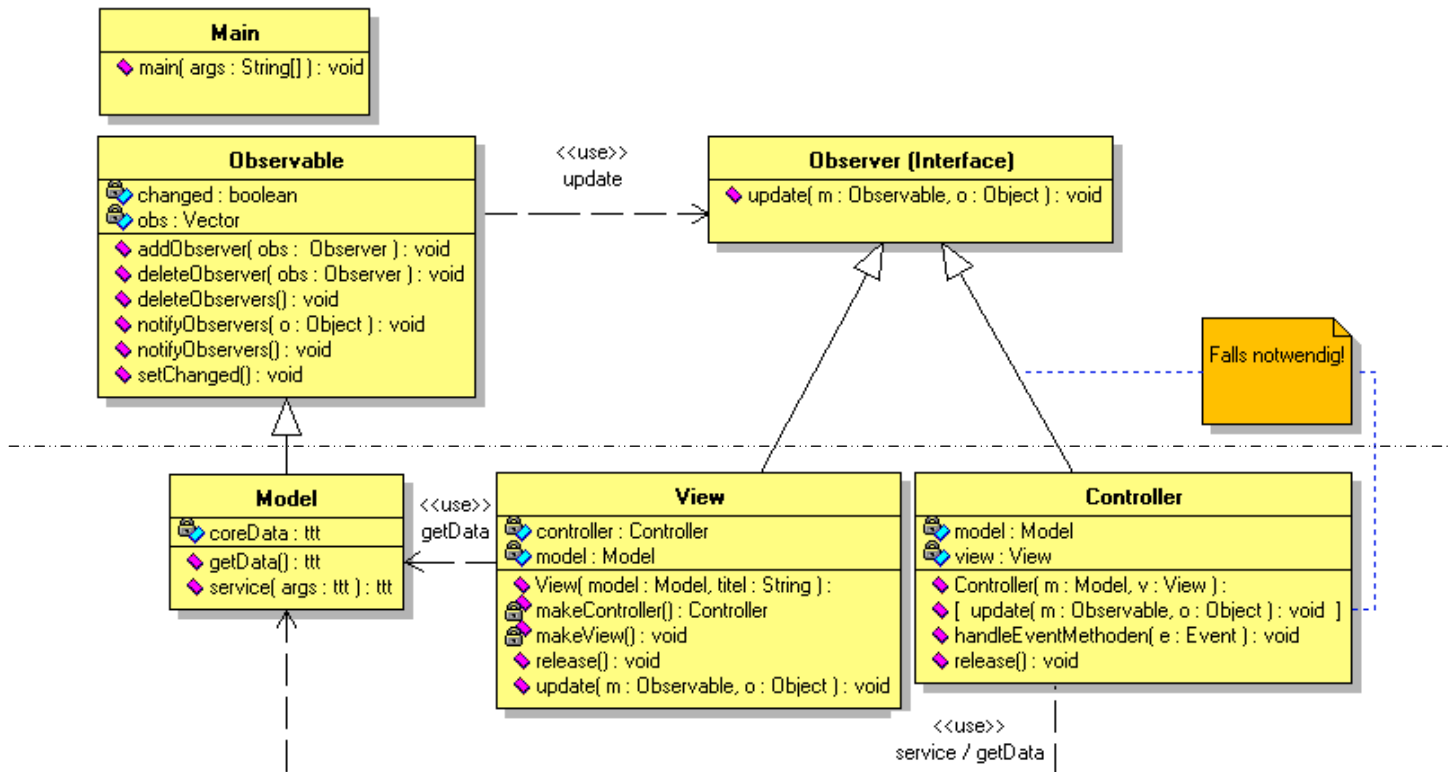
Durch die MVC-Architektur wird das mathematische Modell von der eigentlichen Programmsteuerung entkoppelt. Bei der Programmsteuerung wiederum werden Eingaben und Ausgaben getrennt behandelt.



Das Verhalten eines **View** und unter Umständen auch des dazugehörigen **Controller** ist von Änderungen im **Model** abhängig. Das bedeutet, sie müssen das **Model** überwachen bzw. das **Model** wird von ihnen überwacht. Java stellt nun zwei Klassen zur Verfügung, welche den Aufbau eines solchen *Überwachungsmechanismus* ermöglicht:

Zu *überwachte* Klassen werden von der Klasse **Observable** abgeleitet. *Überwachende* Klasse implementieren das Interface **Observer**.

Der *Überwachungsmechanismus* wird in der folgenden **MVC-Grundstruktur** beschrieben:



**Modellierung und Programmierung mit MVC-Architektur**

**Grundstruktur**

[Klassendiagramm](#), [Dokumentation](#), [MVC.zip](#)

## 2.2 Einige Anwendungen aus Vorlesung und Praktikum

### Beispiel „Zähler modulo 10“

```

K:\Monika_neu\PrakJava\SS_04\Aufg_Gr_3\Vorlesung\GUIampeIMCU\MVC>java MyMain
0
Weiter <j/n>: j
1
Weiter <j/n>: j
2
Weiter <j/n>: j
3
Weiter <j/n>: j
4
Weiter <j/n>: j
5
Weiter <j/n>: j
6
Weiter <j/n>: j
7
Weiter <j/n>: j
8
Weiter <j/n>: j
9
Weiter <j/n>: j
0
Weiter <j/n>: j
1
Weiter <j/n>: j
2
Weiter <j/n>: n
K:\Monika_neu\PrakJava\SS_04\Aufg_Gr_3\Vorlesung\GUIampeIMCU\MVC>
    
```

*Modellierung und Programmierung mit MVC-Architektur*

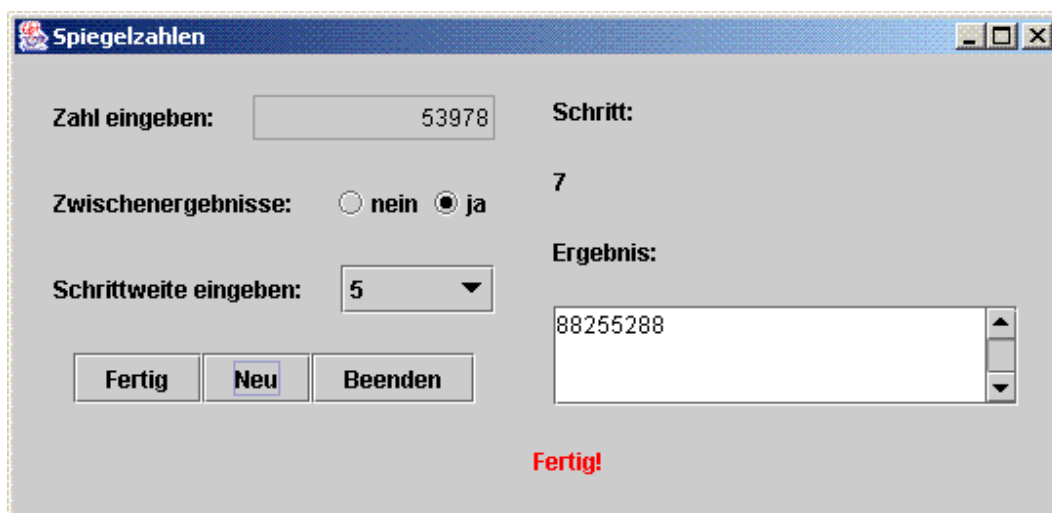
[Klassendiagramm](#), [Dokumentation](#), [ZaehlerMVC.zip](#)

*Modellierung und Programmierung mit MVC-Architektur und Oberfläche*



[Klassendiagramm](#), [Dokumentation](#), [ZaehlerMVC\\_GUI.zip](#)

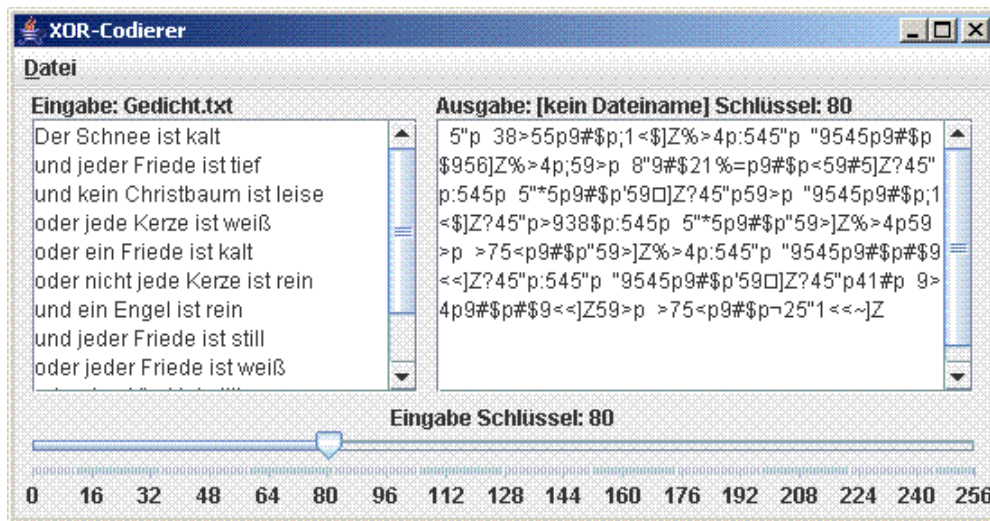
### Beispiel „Spiegelzahlen“ aus Aufgabengruppe 1



*Modellierung und Programmierung mit MVC-Architektur*

[Klassendiagramm](#), [Dokumentation](#), [SpiegelZahlMVC.zip](#)

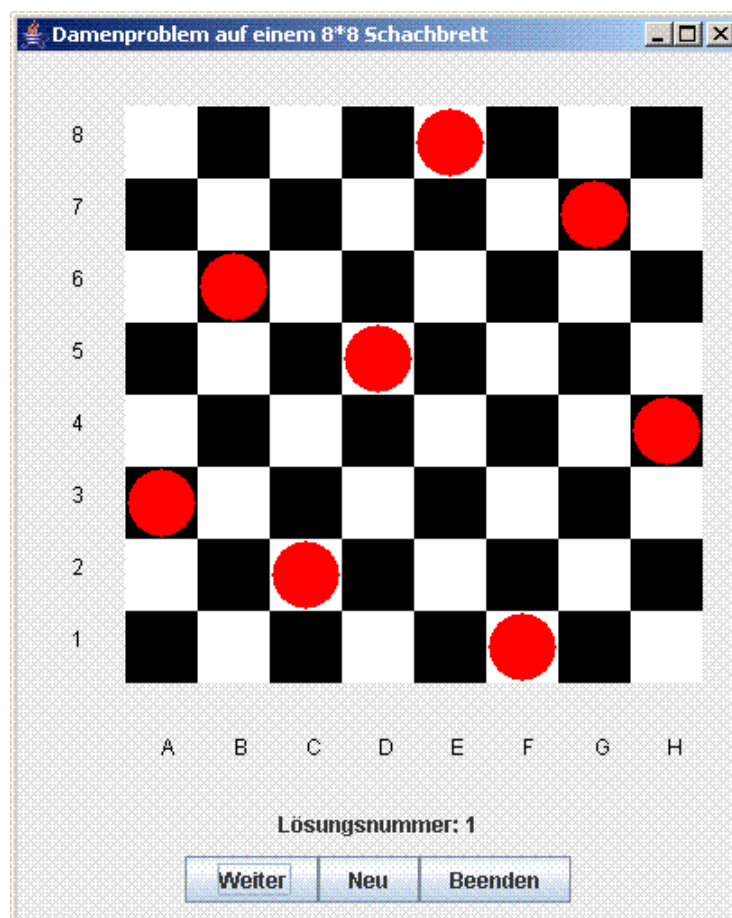
**Beispiel „XORCodierer“ aus Aufgabengruppe 2**



*Modellierung und Programmierung mit MVC-Architektur*

[Klassendiagramm](#), [Dokumentation](#), [XOR MVC.zip](#)

**Beispiel „Dameproblem“ aus Aufgabengruppe 3**

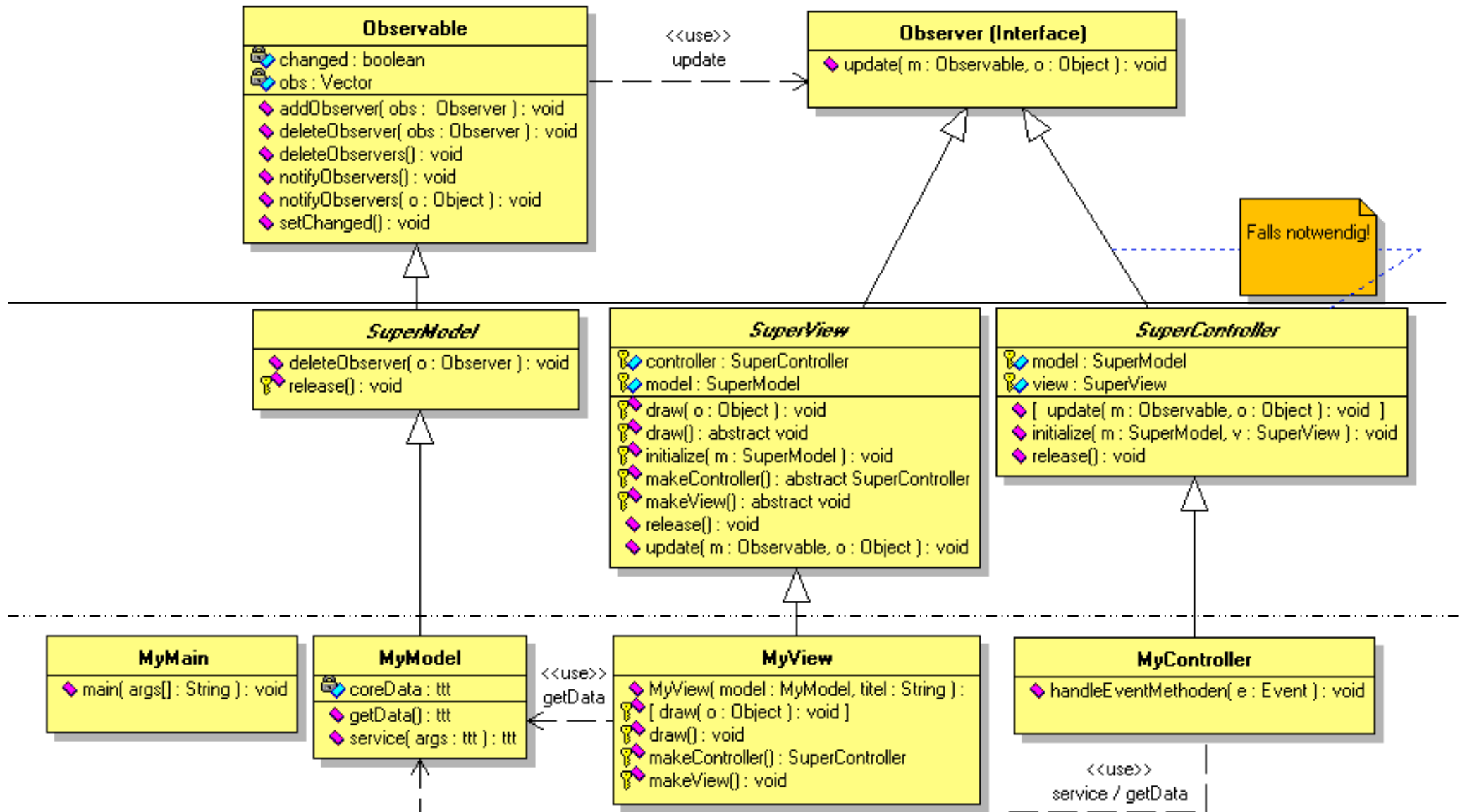


*Modellierung und Programmierung mit MVC-Architektur*

[Klassendiagramm](#), [Dokumentation](#), [DameMVC.zip](#)

### 3 Projekte mit mehreren Benutzeroberflächen

#### 3.1 Klassendiagramm mit eingebundenen MVC-Tool



### 3.2 Paket `Tools.MVC`

Alle diese Beispiele beinhalten in ihren Klassen *ständig wiederkehrende*, aber natürlich auch *programmspezifische Attribute und Methoden*. Es bietet sich hier eine *Generalisierung* an, insbesondere in Hinblick auf die Programmierung von mehreren Benutzeroberflächen zu einem Model. Methoden und Attribute, die alle Programme gemeinsam haben, werden in den abstrakten Klassen des Pakets [Tools.MVC](#) zusammengefasst. Dieses Tools lässt sich aber auch bei nur einer Oberfläche einbinden.

**Paket `Tools.MVC`**

[SuperModel.java](#)  
[SuperView.java](#)  
[SuperController.java](#)

**Anwendung des Pakets**

[MyModel.java](#)  
[MyView.java](#)  
[MyController.java](#)  
[MyMain.java](#)

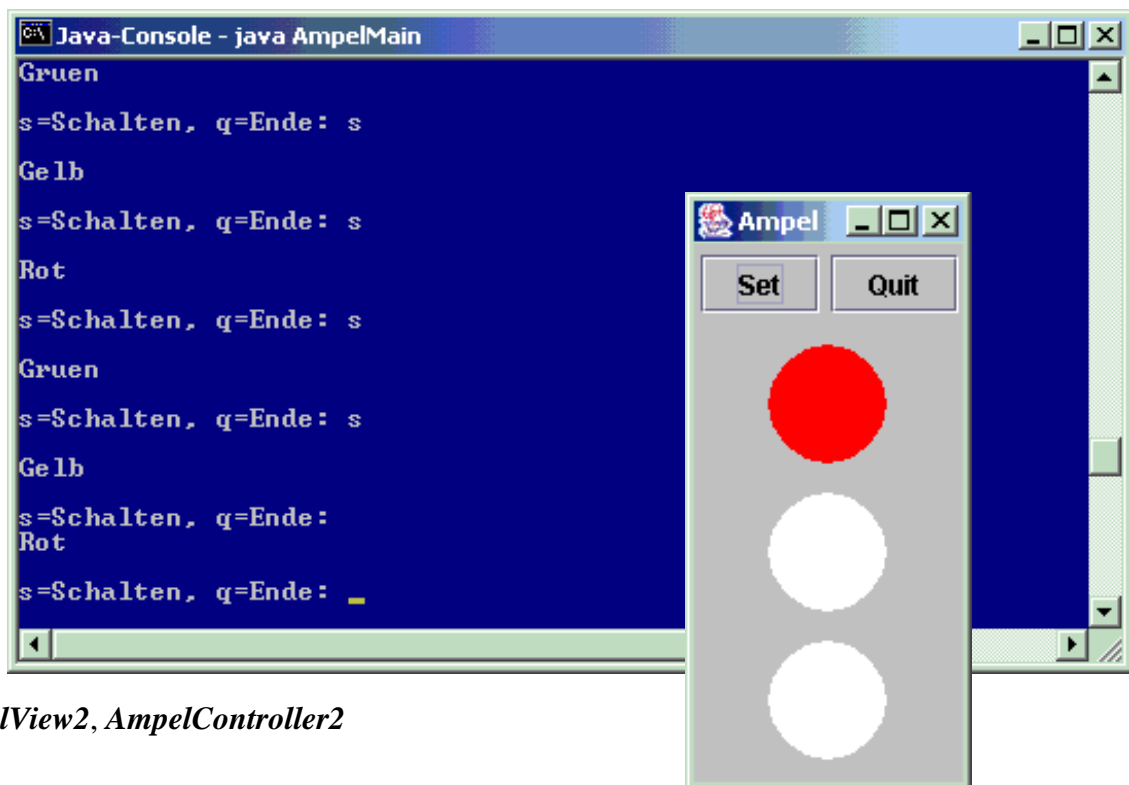
*Modellierung und Programmierung mit MVC-Architektur*

mit Pakets [Tools.MVC](#)

[Klassendiagramm](#), [Dokumentation](#), [MVCTool.zip](#)

### 3.3 Beispiel „Ampelsteuerung“ mit zwei View/Controller

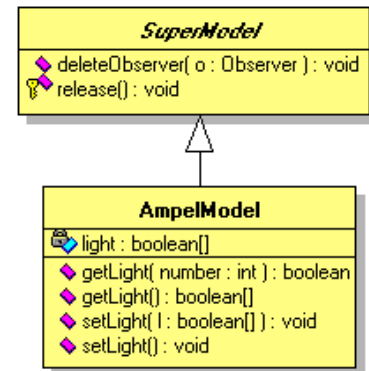
*AmpelView1, AmpelController1*



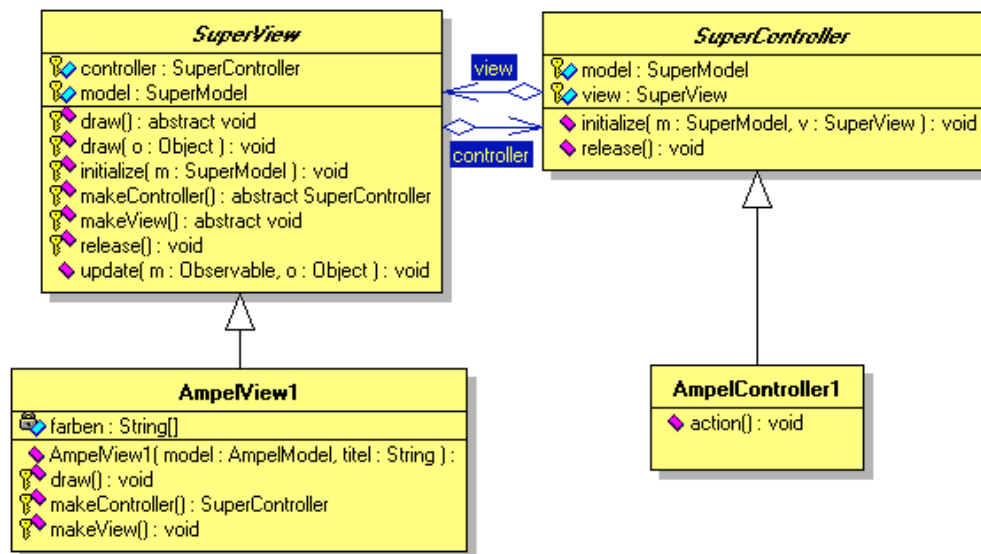
*AmpelView2, AmpelController2*

**Klassendiagramm mit MVC-Tool**

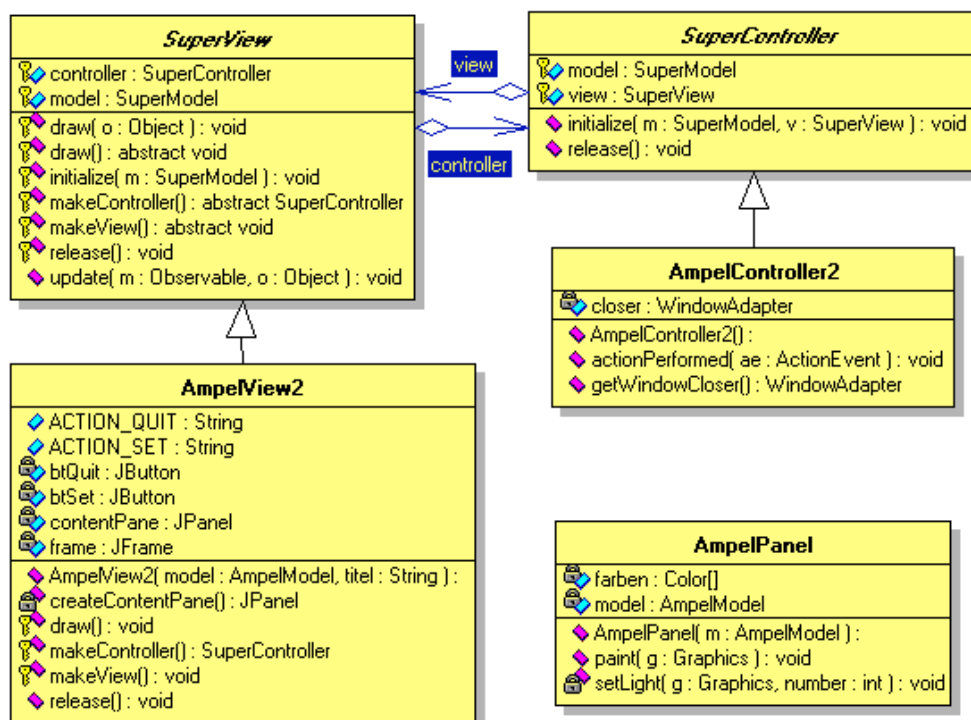
**Modell**



**Benutzeroberfläche Version Tastatur/Konsole**



**Benutzeroberfläche Version grafische Benutzeroberfläche**



Im Hauptprogramm **AmpelMain** werden beide **AmpelView/AmpelController**-Versionen für das **AmpelModel** installiert.

**AmpelMain.java**

MM 2007

```
// AmpelMain.java
// MVC-Main
/**
 * Simulation einer Ampelschaltung,
 * Version mit zwei Views und zwei Controller.
 */
public class AmpelMain
{
/**
 * Starten eines Ampelprogramms mit MVC-Architektur,
 * Initialisiere ein Model mit zwei View.
 */
    public static void main( String args[] )
    {
// Model
        AmpelModel trafficLight = new AmpelModel();

// Views
        AmpelView2 viewTrafficLight2 =
            new AmpelView2( trafficLight, "Ampel" );

        AmpelView1 viewTrafficLight1 =
            new AmpelView1( trafficLight, "Ampel" );
    }
}
```

**Modellierung und Programmierung mit MVC-Architektur**

[Dokumentation, AmpelMVCTool.zip](#)

**4 Zusammenfassung**

Zusammenfassend kann man den Aufbau eines Programms durch die folgende Struktur beschreiben:

