Medientechnik

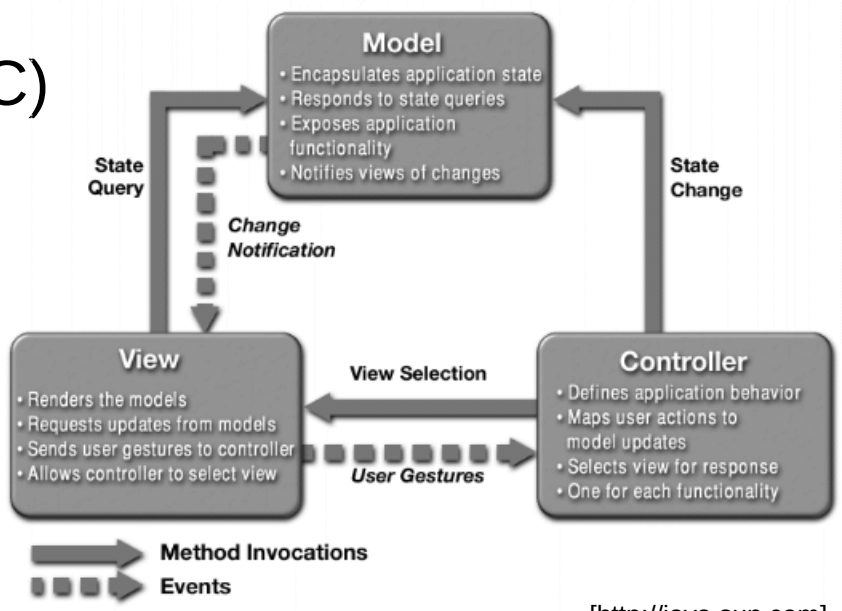
Übung 3 – MVC & Observer

Planung

Nr	Zeitraum	Thema
1	20.04. – 25.04.	Bildretusche mit Gimp
2	27.04. – 01.05.	GUI Programmierung
3	04.05. – 08.05.	Model-View Controller
4	18.05. – 22.05.	Bildfilter
5	26.05. – 29.05.	Video & Film Theorie
6	29.05. – 03.06.	Audio-Aufnahme und -Bearbeitung

Heute

- Model-View-Controller (MVC)
  - Model programmieren
    - Programmlogik
    - Programmdaten
  - Controller programmieren
    - GUI ←→ Model



[http://java.sun.com]

- Observer-Pattern
  - Observable (Model) verwaltet Daten
  - Observer (View) zeigt die Daten an und aktualisiert sich, sobald im Observable `setChanged();` und `notifyObservers();` aufgerufen wird

- Code sinnvoll kommentieren (Javadoc) ☺

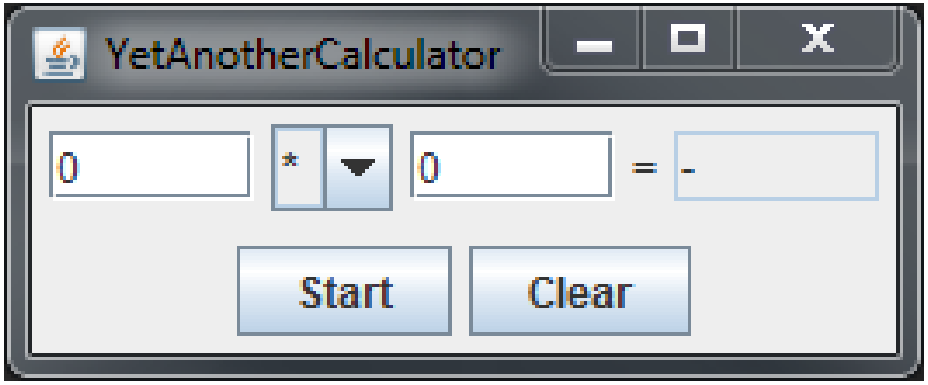
Los geht’s!

- Eclipse starten und Workspace festlegen
- View-Projekt der letzten Übung öffnen

  Source-Dateien gibt es notfalls auf der Homepage zum Download!

Kurzer Rückblick

- GUI für sehr einfachen Taschenrechner
- Unterschiedliche Elemente mit Hilfe von Layout-Managern angeordnet
- (default-)Werte für einzelne Elemente gesetzt



MVC – main-Methode

Yaca.java

```
public class Yaca {


    public static void main(String[] args){
        new Controller();
    }
}
```

MVC - Controller

Controller.java

```
public class Controller {

public View yacaView;

public Controller() {
    yacaView = new View();
    yacaView.setVisible(true);
}
```

MVC - Controller

Controller.java

```
public class Controller {

private View yacaView;
private Model yacaModel;

public Controller() {
    yacaModel = new Model();
    yacaView = new View();

    yacaModel.addObserver(yacaView);
    yacaView.setVisible(true);
}
```

MVC - Model

Model.java

```
import java.util.Observable;

public class Model extends Observable{

 private float result;
 private float a;
 private float b;

 public Model() {
    result = 0;
    a = 0;
    b = 0;

 }
}
```

MVC - Model

Model.java

```
public class Model extends Observable{

 private float result;
 private float a;
 private float b;
[…]
 public float getResult() {
     return result;
 }

 public float getA() {
     return a;
 }

 public float getB() {
     return b;
 }
}
```

MVC - View

View.java

```
[…]
import java.util.Observer;
import java.util.Observable;

public class View extends JFrame implements Observer {

   public View() {
      […]
   }
   public void update(Observable o, Object params) {

   }
}
```

MVC - View

View.java

```
[…]
import java.util.Observer;
import java.util.Observable;

public class View extends JFrame {

  public View() {
     […]
  }
  public void update(Observable o, Object obj) {
          Model m = (Model) o;
          result.setText(""+ m.getResult());
          firstInput.setText("" + m.getA());
          secondInput.setText(""+ m.getB());
  }
}
```

MVC - View

View.java

```
[…]
public class View extends JFrame implements Observer {

    JButton start = new JButton("Start");
    JButton clear = new JButton("Clear");

    JTextField firstInput = new JTextField(5);
    JTextField secondInput = new JTextField(5);
    JTextField result     = new JTextField(5);
    String[] methods = {"+", "-", "*", "/"};
    JComboBox  methodBox = new JComboBox(methods);

    […]
}
```

Alle diese Zeilen vorher im Konstruktor löschen und zu Instanzvariablen Machen!

ACTION LISTENER

MVC - View

View.java

```
[…]
import java.util.Observer;
import java.util.Observable;

public class View extends JFrame implements Observer {
      […]
      public View(ActionListener yacaController){
            […]
            contentButtons.add(start);
            start.setActionCommand("Start");
            start.addActionListener(yacaController);
            contentButtons.add(clear);
            clear.setActionCommand("Clear");
            clear.addActionListener(yacaController);
            […]
      }
      […]
}
```

MVC - View

View.java

```
[…]
public class View extends JFrame implements Observer {
    […]
    public float getFirstInput() {
        float a = 0;
        try {
            a = Float.parseFloat(firstInput.getText());
        }
        catch (NumberFormatException e) {
            System.out.println("First value invalid!");
        }
        return a;
    }

    public float getSecondInput() {
        float b = 0;
        try {
            b = Float.parseFloat(secondInput.getText());
        }
        catch (NumberFormatException e){
            System.out.println("Second value invalid!");}
        return b;
    }
}
```

MVC - View

View.java

```
[…]
public class View extends JFrame implements Observer {
    […]

    public String getMethod() {
        return (String)methodBox.getSelectedItem();
    }

}
```

MVC - Controller

Controller.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controller implements ActionListener {

  Model yacaModel;
  View yacaView;

  public Controller() {
     yacaView = new View(this);
     yacaModel = new Model();

     yacaModel.addObserver(yacaView);
     yacaView.setVisible(true);
  }

  public void actionPerformed(ActionEvent event) {}
}
```

MVC - Controller

Controller.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controller implements ActionListener {
   […]
  public void actionPerformed(ActionEvent event) {
     String cmd = event.getActionCommand();
     if (cmd.equals("Start")){
         yacaModel.calc(
         yacaView.getFirstInput(),
         yacaView.getSecondInput(),
         yacaView.getMethod()
         );
     }
     if (cmd.equals("Clear")){
         yacaModel.clear();
     }
}
```

MVC - Model

Model.java

```
public class Model extends Observable{
[…]
      public void calc(float a, float b, String method){
         this.a = a;
         this.b = b;

         if (method.equals("+")) {
             result = a + b;
         }
         else if (method.equals("-")) {
             result = a - b;
         }
         setChanged();
         notifyObservers();
      }
}
```

MVC - Model

Model.java

```
public class Model extends Observable{
[…]
    public void clear(){
        result = 0;
        a = 0;
        b = 0;
        setChanged();
        notifyObservers();
    }
}
```

MVC - Controller

Controller.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controller implements ActionListener {

  Model yacaModel;
  View yacaView;

  public Controller() {
     yacaView = new View(this);
     yacaModel = new Model();
     yacaModel.addObserver(yacaView);
     yacaModel.clear();
     yacaView.setVisible(true);
  }

  public void actionPerformed(ActionEvent event) {[…]}
}
```

Übungsblatt 2

- Zur bisherigen GUI ein Model und einen Controller ergänzen
- GUI-Aktualisierung durch Observer-Pattern
- Genauere Informationen auf dem Übungsblatt → Fragen am besten im Forum stellen
- Noch *keine* funktionierenden Bildfilter nötig!

Wrap-up Quiz

1. In welcher Klasse wird der Observer beim Modell registriert?
2. Welche Methode der Schnittstelle ActionListener muss man implementieren?
3. Wie wird ein ActionListener einem Element hinzugefügt?
4. Was ist ein ActionCommand und wofür wird er benötigt?
5. Ist Observable eine Klasse oder eine Schnittstelle?
6. Welche beiden Methoden müssen aufgerufen werden, wenn sich etwas am Modell geändert hat?
7. Wann kann es zu einer NumberFormatException kommen?
8. Was bewirkt die folgende Zeile?
   Model m = (Model) someObject;
9. Warum wurden in unserem Beispiel die Textfelder zu Instanzvariablen gemacht?

Vielen Dank!

WELCHE FRAGEN GIBT ES? ☺