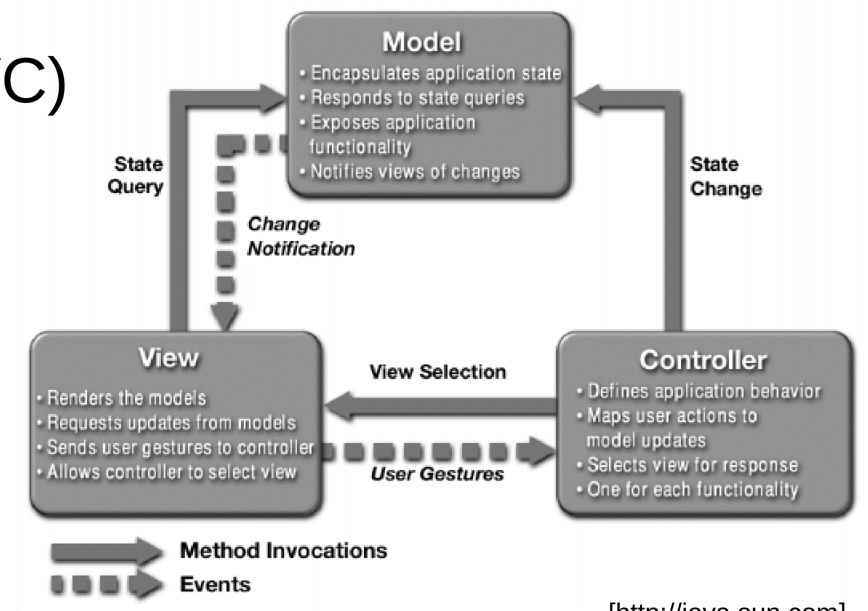
Medientechnik

Übung – MVC

Heute

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



Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

[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

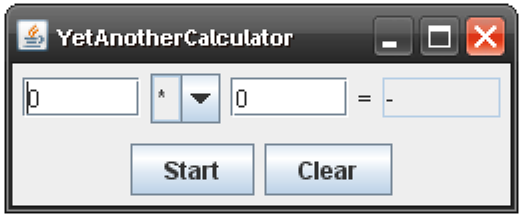
- Code sinnvoll kommentieren (Javadoc) ☺

Eclipse

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



MVC – main-Methode

```
public class Yaca {

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

Yaca.java

MVC – Controller

```
public class Controller {

public View yacaView;

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

Controller.java

MVC – Model

```
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;

 }

}
```

Model.java

MVC – Model

```
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 getA() {
     return a;
 }

}
```

Model.java

MVC – View

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

public class View extends JFrame implements Observer {
    public View(){
    […]
    }

    public void update(Observable o, Object obj) {

    }
}
```

View.java

MVC – View

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

public class View extends JFrame implements Observer {
    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());
    }
}
```

View.java

MVC – Controller

```
public class Controller {

public View yacaView;
public Model yacaModel;

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

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

Controller.java

MVC – View

```
[…]

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);

    […]
}
```

Diese Zeilen im Konstruktor löschen!

View.java

MVC – Controller

```
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) {
}
```

Controller.java

MVC – View

```
[…]

public class View extends JFrame implements Observer {

    public View(Controller yacaController){
    […]
        contentButtons.add(start);
        start.setActionCommand("Start");
        start.addActionListener(yacaController);
        contentButtons.add(clear);
        clear.setActionCommand("Clear");
        clear.addActionListener(yacaController);
    […]
    }

    […]
}
```

View.java

MVC – Controller

```
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();
     }
 }
}
```

Controller.java

MVC – Model

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

 }
}
```

Model.java

MVC – Model

```
public class Model extends Observable{

     […]


public void clear()
 {
     result = 0;
     a = 0;
     b = 0;
     setChanged();
     notifyObservers();
 }

}
```

Model.java

MVC – View

```
[…]

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(“Ungueltiger erster Wert!”);}
            return a;
      }

      public float getSecondInput() {
            float b = 0;
            try {
             b = Float.parseFloat(secondInput.getText());
            }
            catch (NumberFormatException e) {System.out.println(“Ungueltiger zweiter Wert!”);}
            return b;
      }
}
```

View.java

MVC – View

```
[…]

public class View extends JFrame implements Observer {
    […]
    public float getSecondInput() {
        floatt b = 0;
        try {
         b = Float.parseFloat(secondInput.getText());
        }
        catch (NumberFormatException e)
    {System.out.println(“Ungueltiger zweiter Wert!”);}
        return b;
    }

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

View.java

MVC – Controller

```
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) {
}
```

Controller.java

Kurze Info zum Übungsblatt

- 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!