Vorlesung

TECHNISCHE HOCHSCHULE LÜBECK

Programmieren I und II

Unit 11

Graphical User Interfaces

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
1

1

Disclaimer

TECHNISCHE HOCHSCHULE LÜBECK

Zur rechtlichen Lage an Hochschulen:

Dieses Handout und seine Inhalte sind durch den Autor selbst erstellt. Aus Gründen der Praktikabilität für Studierende lehnen sich die Inhalte stellenweise im Rahmen des Zitatrechts an Lehrwerken an.

Diese Lehrwerke sind explizit angegeben.

Abbildungen sind selber erstellt, als Zitate kenntlich gemacht oder unterliegen einer Lizenz die nicht die explizite Nennung vorsieht. Sollten Abbildungen in Einzelfällen aus Gründen der Praktikabilität nicht explizit als Zitate kenntlichgemacht sein, so ergibt sich die Herkunft immer aus ihrem Kontext: „Zum Nachlesen ...“.

Creative Commons:

Und damit andere mit diesen Inhalten vernünftig arbeiten können, wird dieses Handout unter einer Creative Commons Attribution-ShareAlike Lizenz (CC BY-SA 4.0) bereitgestellt.

https://creativecommons.org/licenses/by-sa/4.0

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
2

2

TECHNISCHE HOCHSCHULE LÜBECK

Prof. Dr. rer. nat. Nane Kratzke

Praktische Informatik und betriebliche Informationssysteme

- **Raum: 17-0.10**
- **Tel.:** 0451 300 5549
- **Email:** nane.kratzke@th-luebeck.de

@NaneKratzke

Updates der Handouts auch über Twitter #prog_inf und #prog_itd

3

Units

TECHNISCHE HOCHSCHULE LÜBECK

Unit 1 Einleitung und Grundbegriffe	**Unit 2** Grundelemente imperativer Programme	**Unit 3** Selbstdefinierbare Datentypen und Collections	**Unit 4** Einfache I/O Programmierung
Unit 5 Rekursive Programmierung und rekursive Datenstrukturen	**Unit 6** Einführung in die objektorientierte Programmierung und UML	**Unit 7** Konzepte objektorientierter Programmiersprachen	**Unit 8** Testen (objektorientierter) Programme
Unit 9 Generische Datentypen	**Unit 10** Objektorientierter Entwurf und objektorientierte Designprinzipien	**Unit 11** Graphical User Interfaces	**Unit 12** Multithread Programmierung

4

Abgedeckte Ziele dieser UNIT

TECHNISCHE HOCHSCHULE LÜBECK

Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmus-begriffs, Implementieren einfacher Algorithmen	Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)	Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Mulithread Programmierung sowie funktionale Programmierung

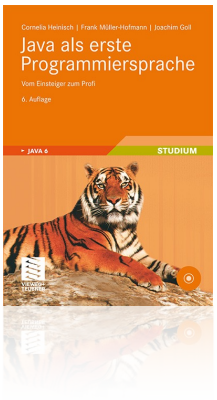
Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
5

5

Zum Nachlesen ...

TECHNISCHE HOCHSCHULE LÜBECK



Kapitel 21

Oberflächenprogrammierung mit SWING

Teil VI

Kapitel 35 (Swing Grundlagen)

Kapitel 36 (Container und Menus)

Kapitel 37 (Komponenten I)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
6

6

Themen dieser Unit

TECHNISCHE HOCHSCHULE LÜBECK

GUI

- Java Swing
- MVC
- View Konzepte
- Controller Konzepte

Taschenrechner

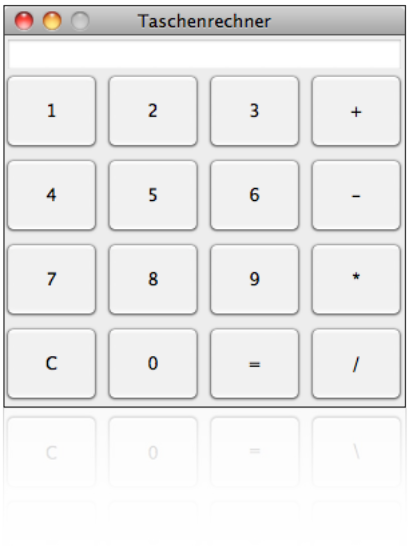
- Model
- View
- Controller

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
7

7

Entwicklung eines Taschenrechners

TECHNISCHE HOCHSCHULE LÜBECK



- Grafische Bedienoberflächen unterstützen Bediener in der Bedienung einer Anwendung
- Viele Programmiersprachen bieten hierzu spezielle Bibliotheken an, die grafische Bedienelemente definieren
- JAVA nutzt hierzu u.a. die sogenannte SWING Bibliothek
- Programmieroberfläche am Beispiel einer einfachen Taschenrechner Applikation

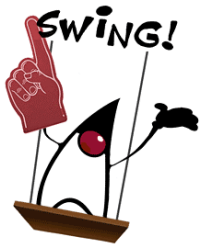
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
8

8

Die SWING Klassenbibliothek

TECHNISCHE HOCHSCHULE LÜBECK

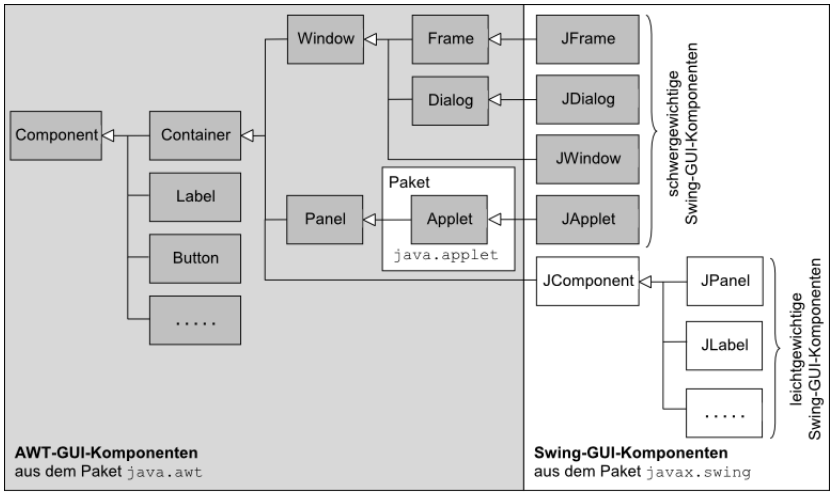
- SWING ist eine Klassenbibliothek für die Programmierung von **grafischen Bedienoberflächen**
- SWING beinhaltet schwer- und leichtgewichtige GUI-Komponenten (**Graphical User Interface – GUI**)
- Klassen zur Darstellung von „Fenstern“ (`JFrame, JDialog, JWindow, JApplet`) sind die **schwergewichtige** SWING-GUI-Komponenten und sind deshalb in Aussehen und Verhalten abhängig vom Betriebssystem
- „**Leichtgewichtige**“ SWING-GUI-Komponenten werden mit Hilfe von Java 2D-Klassenbibliotheken durch die JVM selbst auf dem Bildschirm gezeichnet und sind damit in Aussehen und Verhalten unabhängig vom Betriebssystem



9

Die Grafik Bibliothek SWING von JAVA

TECHNISCHE HOCHSCHULE LÜBECK

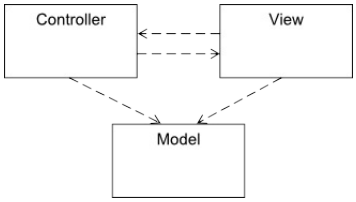


10

Model – View – Controller (I)
MVC

TECHNISCHE HOCHSCHULE LÜBECK

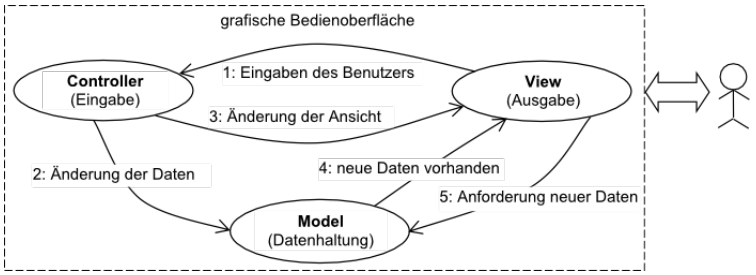
- Ein MVC ist ein so genanntes **Architekturmuster**
- Ein Architekturmuster beschreibt im Generellen eine **bewährte Zerlegung** eines Systems in Teilsysteme und ihr Zusammenwirken
- Das MVC Pattern beschreibt einer bewährte Aufteilung eines Systems mit einem **Human-Machine-Interface (HMI)**



11

Zusammenarbeit von Model, View und Controller

TECHNISCHE HOCHSCHULE LÜBECK



Quelle: Java als erste Programmiersprache

12

Verantwortlichkeiten im MVC Pattern

TECHNISCHE HOCHSCHULE LÜBECK

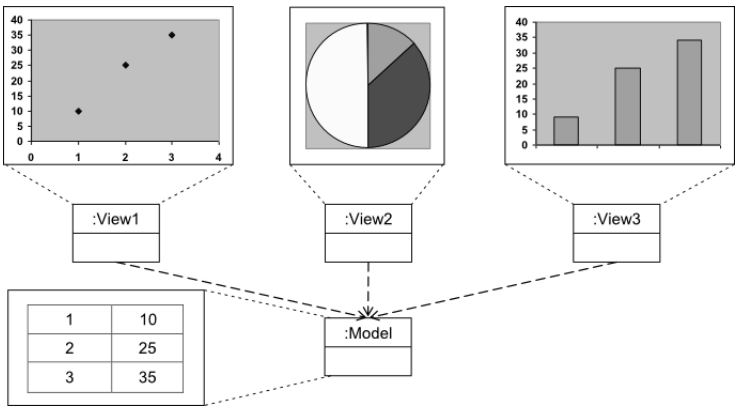
Model	View	Controller
• Realisiert die konzeptionelle Lösung (Programmlogik) • Hält die für die Anzeige relevanten Daten • Das Model bestimmt welche Daten wie verändert werden • Welche Daten zur Ansicht bereitgestellt werden	• Darstellung der Daten des Model • Bereitstellung von Eingabemöglichkeiten für das Model • Verschiedene Views können Daten des Model unterschiedlich darstellen • Werden Daten im Model geändert, so ändern sich auch die Darstellungen der Views	• Steuert das Model und den View • Controller ruft Methoden des Model auf, um Zustand gem. Benutzereingaben zu ändern • Ggf. wird auch die Darstellung des Views aktualisiert

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
13

13

Model – View – Controller (II)
MVC

TECHNISCHE HOCHSCHULE LÜBECK



Quelle: Java als erste Programmiersprache

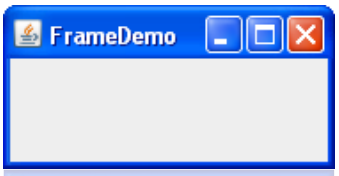
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
14

14

View Konzepte
Hier: Windows und Dialoge

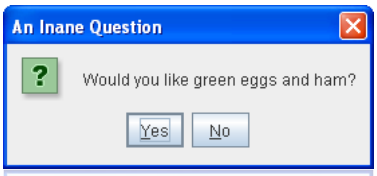
TECHNISCHE HOCHSCHULE LÜBECK

JFrame



A Frame is a top-level application window with a title and a border. A Frame contains typically several GUI components arranged by a layout manager.

JDialog



A Dialog window is an independent subwindow meant to carry temporary notice apart from the main Swing Application Window. Most Dialogs present an error message or warning to a user, but Dialogs can present images, directory trees, or just about anything compatible with the main Swing Application that manages them. Dialogs typically block the main window of an application until the dialog is closed/finished.

15

View Konzepte
Hier: GUI Swing Komponenten (Auswahl)

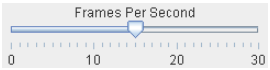
TECHNISCHE HOCHSCHULE LÜBECK

JButton

JCheckBox

JComboBox

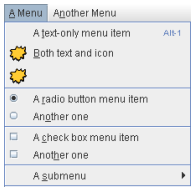
JSlider



JRadioButton

JList

JMenu



JTextField

JPasswordField

JTable

Host	User	Password	Last Modified
Biocca Games	Freddy	f#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazbl34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkzfADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

16

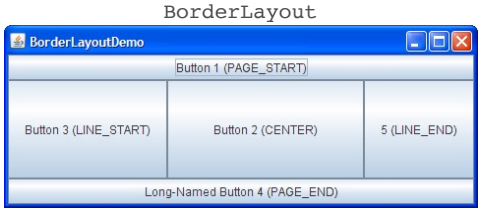
View Konzepte
Hier: Layout Manager (Auswahl)

TECHNISCHE HOCHSCHULE LÜBECK

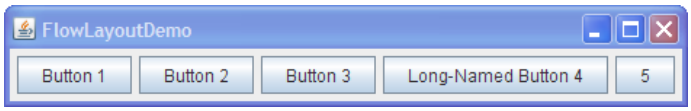
A layout manager is an object that determines the size and position of the components within a container.

Each container can have its own layout manager.

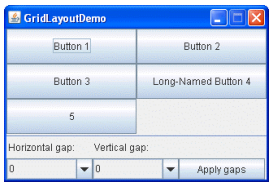
BorderLayout



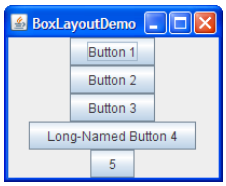
FlowLayout



GridLayout



BoxLayout



17

View Konzepte
Hier: Beispiel `BorderLayout`

TECHNISCHE HOCHSCHULE LÜBECK

```
import java.awt.*;
import javax.swing.*;
public class LayoutManager {
  public static void main(String[] args) {
    JButton tf1 = new JButton("Hello");
    JButton tf2 = new JButton("My");
    JButton tf3 = new JButton("Name");
    JButton tf4 = new JButton("is");
    JButton tf5 = new JButton("Rabbit.");

    JFrame f = new JFrame("Main Window");
    f.setLayout(new BorderLayout());
    f.add(tf1, BorderLayout.PAGE_START);
    f.add(tf2, BorderLayout.LINE_START);
    f.add(tf3, BorderLayout.CENTER);
    f.add(tf4, BorderLayout.LINE_END);
    f.add(tf5, BorderLayout.PAGE_END);

    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
  }
}
```

Anlegen der GUI Komponeten

Erzeugen des Window und Zuweisen eines Layout Managers

Zuweisen der Komponenten an Bereiche des Layout Managers

Darstellen des Windows

18

View Konzepte
Hilfreiche Links auf die JAVA Dokumentation

TECHNISCHE HOCHSCHULE LÜBECK

JAVA Swing UI Manuals and Tutorials

`http://docs.oracle.com/javase/tutorial/uiswing/TOC.html`

Swing GUI Components

`http://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html`

Layout Managers

`http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html`

19

Controller Konzepte
Hier: Ereignisbehandlung

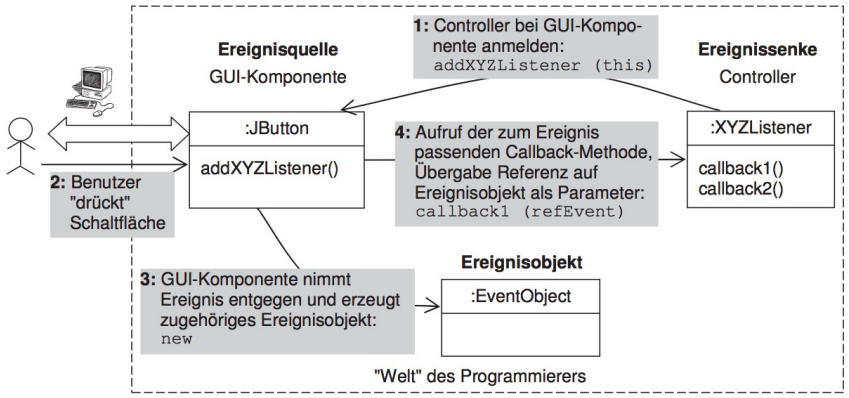
TECHNISCHE HOCHSCHULE LÜBECK

- **Ereignisbehandlung**
  - Für das Verständnis der Ereignisbehandlung ist es hilfreich, die Begriffe **Ereignisquelle** und **Ereignissenke** einzuführen.
  - Eine Ereignissenke (Controller) meldet sich bei einer Ereignisquelle (GUI-Komponente) für spezielle Ereignisse (Events) mittels eines Listeners an.
  - Tritt ein Ereignis auf, so wird dies von der Ereignisquelle an alle für dieses Ereignis angemeldeten Ereignissenken weitergeleitet.
  - Die Ereignissenken sind dann für die eigentliche Ereignisverarbeitung zuständig.

20

Controller Konzepte
Hier: Callback-Schnittstelle

TECHNISCHE HOCHSCHULE LÜBECK



Einer GUI-Komponente sind **Callback**-Schnittstellen (mit Namen `XYZListener`) zugeordnet, die der Programmierer in einem Controller implementieren muss, falls er Ereignisse für diese GUI-Komponente abfangen und verarbeiten möchte.

Bildquelle: Java als erste Programmiersprache

21

Controller Konzepte
Hier: Ereignisse

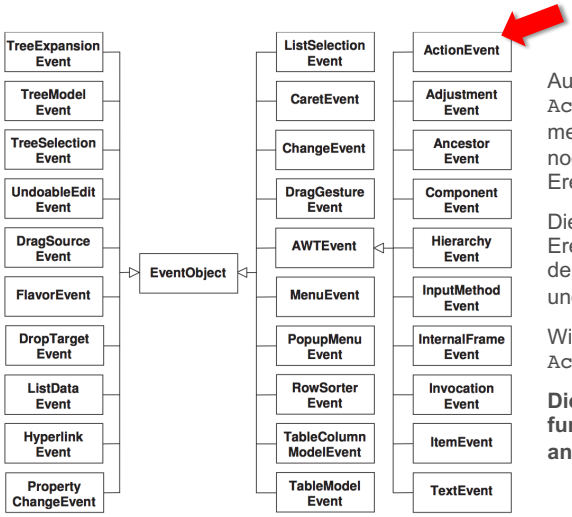
TECHNISCHE HOCHSCHULE LÜBECK

- Für jedes Ereignis wird in JAVA bei dessen Eintreten ein Objekt vom Typ `EventObject` angelegt.
- Dieses Ereignisobjekt spielt eine wichtige Rolle für den Informationsaustausch zwischen Ereignisquelle und Ereignissenke.
- Ein Ereignisobjekt speichert Informationen zum aufgetretenen Ereignis sowie eine Referenz auf die Ereignisquelle.
- Damit die Ereignissenke die Ereignisquelle ermitteln kann, implementiert die Klasse `EventObject` die Methode `getSource()`.

22

Controller Konzepte
Hier: Klassenhierarchie der Ereignisse

TECHNISCHE HOCHSCHULE LÜBECK



Auch wenn das Ereignisobjekt `ActionEvent` eines der meist benutzten ist, gibt es noch eine Vielzahl anderer Ereignisse.

Die dargestellten Ereignisklassen finden sich in den Unterpaketen `java.awt` und `javax.swing`.

Wir werden uns nur mit dem `ActionEvent` befassen.

Die vermittelten Prinzipien funktionieren jedoch für die anderen Ereignisse analog.

23

Controller Konzepte
Hier: Listener Schnittstelle

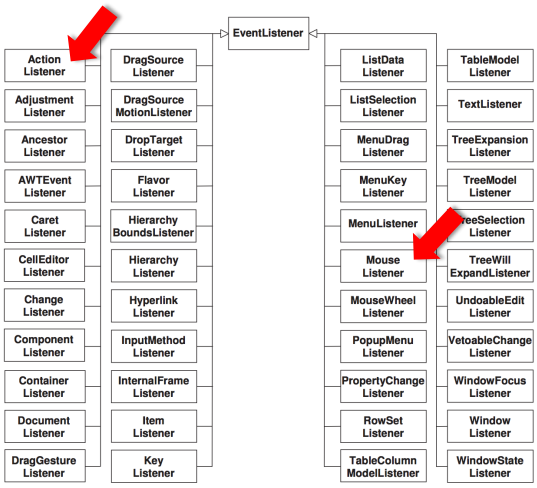
TECHNISCHE HOCHSCHULE LÜBECK

- Damit ein Controller (eine Ereignissenke) bestimmte Ereignisse abfangen kann, muss dieser eine zum Ereignistyp passende `Listener` Schnittstelle implementieren.
- Beispielsweise korrespondiert die `MouseListener` Schnittstelle zum Ereignis vom Typ `MouseEvent`.
- In der Schnittstelle `MouseListener` sind unter anderem die Callback-Methoden `mousePressed()` und `mouseReleased()` deklariert, die es ermöglichen, die zugehörigen Ereignisse einer Maus innerhalb einer GUI-Komponente abzufangen.

24

Controller Konzepte
Hier: Listener Schnittstellen

TECHNISCHE HOCHSCHULE LÜBECK



Die zum meist benutzten Ereignisobjekt `ActionEvent` passende Listener Schnittstelle ist die `ActionListener` Schnittstelle.

Nachfolgendes Beispiel demonstriert die Funktionsweise an einem `MouseListener`.

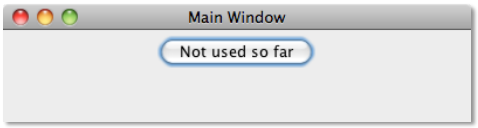
Die vermittelten Prinzipien funktionieren jedoch für alle anderen Listener analog.

25

Controller Konzepte
Hier: Beispiel eines `MouseEvent` Listeners

TECHNISCHE HOCHSCHULE LÜBECK

```
// Necessary imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```




```
public class MouseListenerTest {
  public static void main(String[] args) {
    JFrame f = new JFra
    JButton b = new JBu
    b.addMouseListener(
    f.setLayout(new Flo
    f.add(b);
    f.setSize(400, 100)
    f.setDefaultCloseOp
    f.setVisible(true);
  }
}
```

```
class MyController implements MouseListener {
  private int counter = 0;

  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}

  public void mouseReleased(MouseEvent e) {
    ((JButton)e.getSource()).setText(++counter + " times released.");
  }
}
```


26

Controller Konzepte
Hier: Adapter Klassen

TECHNISCHE HOCHSCHULE LÜBECK

Bei der Implementierung einer Listener-Schnittstelle muss ein Controller alle in der Schnittstelle definierten Callback-Methoden – mindestens durch einen leeren Rumpf – implementieren, auch wenn nur auf ein einzelnes Ereignis reagiert werden soll. Um den Implementierungsaufwand für den Programmierer möglichst gering zu halten, existieren die so genannten Adapter-Klassen.

Eine **Adapter-Klasse implementiert** alle von einer Listener-Schnittstelle vorgegebenen **Callback-Methoden mit einem leeren Rumpf**.

Ein Controller kann nun – alternativ zur Implementierung einer Listener-Schnittstelle – von der zugehörigen Adapter-Klasse ableiten. Es werden im Controller dann nur diejenigen **Callback-Methoden überschrieben**, für die auch tatsächlich eine Implementierung durch den Controller bereitgestellt wird.

Für Listener-Schnittstellen, die mehr als eine Callback-Methode enthalten, wird durch die Java-Klassenbibliothek eine zugehörige Adapter-Klasse bereitgestellt.

27

Controller Konzepte
Hier: Überblick aller Swing Adapter Klassen

TECHNISCHE HOCHSCHULE LÜBECK

Listener-Schnittstelle	Adapter-Klasse
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
DragSourceListener	DragSourceAdapter
DragSourceMotionListener	DragSourceAdapter
DropTargetListener	DropTargetAdapter
FocusListener	FocusAdapter
HierarchyBoundsListener	HierarchyBoundsAdapter
InternalFrameListener	InternalFrameAdapter
KeyListener	KeyAdapter
MouseInputListener	MouseInputAdapter
MouseListener	MouseAdapter MouseInputAdapter
MouseMotionListener	MouseAdapter MouseInputAdapter MouseMotionAdapter
MouseWheelListener	MouseAdapter MouseInputAdapter
WindowFocusListener	WindowAdapter
WindowListener	WindowAdapter
WindowStateListener	WindowAdapter

Quelle: Java als erste Programmiersprache

28

Controller Konzepte
Hier: Überblick aller Swing Adapter Klassen

TECHNISCHE HOCHSCHULE LÜBECK

```
class MyController implements MouseListener {
  private int counter = 0;

  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}

  public void mouseReleased(MouseEvent e) {
    ((JButton)e.getSource()).setText(++counter + " times released.");
  }
}
```

```
class MyAdapter extends MouseAdapter {
  private int counter = 0;

  public void mouseReleased(MouseEvent e) {
    ((JButton)e.getSource()).setText(++counter + " times released.");
  }
}
```

29

Controller Konzepte
Beispiel: MouseAdapter als leere Interface Implementierung

TECHNISCHE HOCHSCHULE LÜBECK

```
class MouseAdapter implements MouseListener {
  public void mouseClicked(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
}
```

Adapter implementieren also alle Callback Methoden eines Listener Interfaces mit der leeren Implementierung.

Von Adaptern abgeleitete Klassen müssen also nicht alle Callbacks Implementieren, sondern nur die notwendigen für die eigene Programmlogik überschreiben.

Meist bedeutet dies eine weniger umfangreiche Implementierung.

30

Implementierungsvarianten von Controllern

TECHNISCHE HOCHSCHULE LÜBECK

Neben der Möglichkeit, einen Controller entweder durch die **Implementierung einer Listener-Schnittstelle** oder durch das **Ableiten von einer Adapter-Klasse** zu schreiben, gibt es generell die Möglichkeit, einen Controller entweder in der Form einer **externen Klasse**, einer **Elementklasse,** einer **anonymen Klasse** oder als **Lambda-Funktion** zu schreiben.

- Controller als externe Klasse
- Controller als Elementklasse
- Controller als anonyme Klasse
- Controller als Lambda-Funktion

31

Controller Implementierung
Variante: Externe Klasse

TECHNISCHE HOCHSCHULE LÜBECK

Die Implementierung eines Controllers durch eine externe Klasse haben Sie bereits in den bisherigen Beispielprogrammen kennengelernt. Durch die Implementierung eines Controllers als eigene externe Klasse wird eine Separierung der Darstellung einer grafischen Bedienoberfläche von der Reaktion auf Benutzereingaben **deutlich zum Ausdruck** gebracht. Durch Speicherung einer selbst implementierten externen Controller-Klasse in einer eigenen Datei kann diese Separierung **noch expliziter** zum Ausdruck gebracht werden.

```
public class MouseListenerTest {
  public static void main(String[] args) {
    JFrame f = new JFrame("Main Window");
    JButton b = new JButton("Not used");
    b.addMouseListener(new MyController());
}
```

```
class MyController extends MouseAdapter {
  private int counter = 0;

  public void mouseReleased(MouseEvent e) {
    ((JButton)e.getSource()).setText(++counter + " times released.");
  }
}
```


32

Controller Implementierung
Variante: Elementklasse

TECHNISCHE HOCHSCHULE LÜBECK

```
public class MouseListenerTest {
  public static void main(String[] args) {
    JFrame f = new JFrame("Main Window");
    JButton b = new JButton("Not used");
    b.addMouseListener(new MyController());
    f.setLayout(new FlowLayout());
    f.add(b);
    f.setSize(400, 100);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
  }
  // Controller als Elementklasse
  class MyController extends MouseAdapter {
    private int counter = 0;
    public void mouseReleased(MouseEvent e) {
      b.setText(++counter + "times released.");
    }
  }
}
```

Die Implementierung eines Controllers durch eine **Elementklasse** bietet den Vorteil, dass eine Elementklasse Zugriff auf die Elemente der äußeren Klasse hat. **Der wesentliche Unterschied ist**, dass der Controller direkt auf die View Elemente der umschließenden Klasse zugreifen kann.

In unserem Falle der Button **b**.

Man erkauft sich diese Vereinfachung allerdings durch eine stärkere Verschränkung des Views und des Controllers. Tendenziell hat diese Art der Implementierung die Tendenz **weniger wartbar** zu sein.

33

Controller Implementierung
Variante: Anonyme Controller Klasse

TECHNISCHE HOCHSCHULE LÜBECK

```
public class MouseListenerTest {
  public static void main(String[] args) {
    JFrame f = new JFrame("Main Window");
    JButton b = new JButton("Not used");
    // Controller als Anonyme Klasse
    b.addMouseListener(new MouseAdapter() {
      private int counter = 0;
      public void mouseReleased(MouseEvent e) {
        b.setText(++counter + "times released.");
      }
    });
    f.setLayout(new FlowLayout());
    f.add(b);
    f.setSize(400, 100);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
  }
}
```

Die Implementierung eines Controllers durch eine anonyme Klasse ist in vielen Fällen die **kompakteste Implementierungsvariante**. Der wesentliche Unterschied ist, dass die anonyme Klassendefinition direkt beim Anmelden des Controllers bei den GUI Komponenten eingefügt wird. Da ein Controller in der Form einer anonymen Klasse nur als Ereignissenke für genau eine GUI-Komponente dienen kann, muss **für jede Schaltfläche ein eigener Controller** geschrieben werden.

Man erkauft sich diese weitere Vereinfachung allerdings durch eine stärkere Verschränkung des Views und des Controllers. Tendenziell hat daher auch diese Art der Implementierung die Tendenz **weniger wartbar** zu sein. Viele **GUI Builder** erzeugen jedoch diese Art von Code **(sie sollten aus diesen anonymen Controllerklassen immer direkt eine klar definierte Controllerklasse aufrufen!)**.

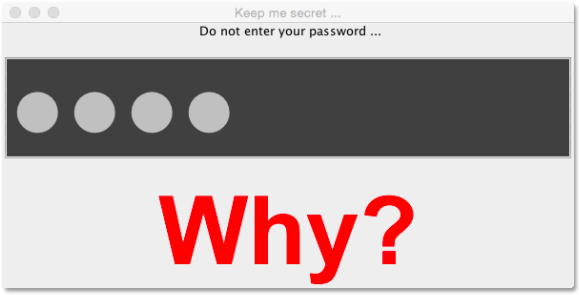
34

Controller Implementierung
Variante: Lambda-Funktion

TECHNISCHE HOCHSCHULE LÜBECK

Die folgende Variante funktioniert nur für Interfaces mit einer Methode (sogenannte funktionale Interfaces) wie bspw. für `ActionListener`.

Für das `MouseListener` Interface (bzw. `MouseAdapter`) funktioniert dies also nicht, da diese Interfaces mehr als eine Methode definieren.

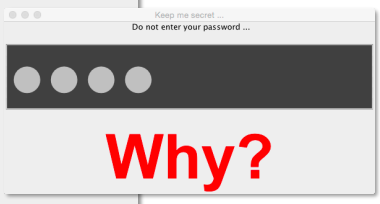


35

Controller Implementierung
Variante: Lambda-Funktion

TECHNISCHE HOCHSCHULE LÜBECK

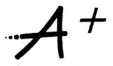
```
public class SwingView extends JFrame {

  private JPasswordField text = new JPasswordField("Edit me");
  private JLabel label = new JLabel("Do not enter your password ...");
  private JLabel ausgabe = new JLabel("...");

  public SwingView(String title) {
    super(title);
    this.setLayout(new BorderLayout(20, 20));
    text.addActionListener(e -> {
      String geheim = ((JTextField)e.getSource()).getText();
      ausgabe.setText(geheim);
    });
    this.add(label, BorderLayout.NORTH);
    this.add(text, BorderLayout.CENTER);
    this.add(ausgabe, BorderLayout.SOUTH);

    this.setVisible(true);
  }
}
```



Man erkauft sich diese weitere Vereinfachung allerdings durch eine stärkere Verschränkung des Views und des Controllers. Tendenziell hat daher auch diese Art der Implementierung die Tendenz **weniger wartbar** zu sein. **Sie sollten auch aus Lambda-Funktionen immer direkt eine klar definierte Controllerklasse aufrufen!**.

36

Zusammenfassung



TECHNISCHE HOCHSCHULE LÜBECK

- **Graphical User Interfaces**
  - JAVA Swing Bibliothek
  - Model View Controller (MVC) Konzept
- **View Konzepte**
  - Schwergewichtige Fenster (`JFrame` und `JDialog`)
  - GUI Komponenten (`JButton`, `JCheckBox`, ...)
  - Layout Manager (`BorderLayout`, `GridLayout`, ...)
- **Controller Konzepte**
  - Events verbinden Quellen (GUI Komponenten) und Senken (Controller)
  - `Listener` und `Adapter` (Callback Methoden)
  - Varianten der Controller Implementierung (Externe Klasse, Elementklasse, Anonyme Klasse)

37

Themen dieser Unit

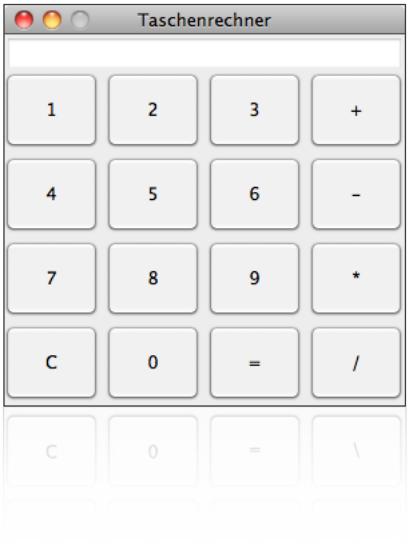
TECHNISCHE HOCHSCHULE LÜBECK

GUI	Taschenrechner
• Java Swing • MVC • View Konzepte • Controller Konzepte	• Model • View • Controller

38

Entwicklung eines Taschenrechners

TECHNISCHE HOCHSCHULE LÜBECK

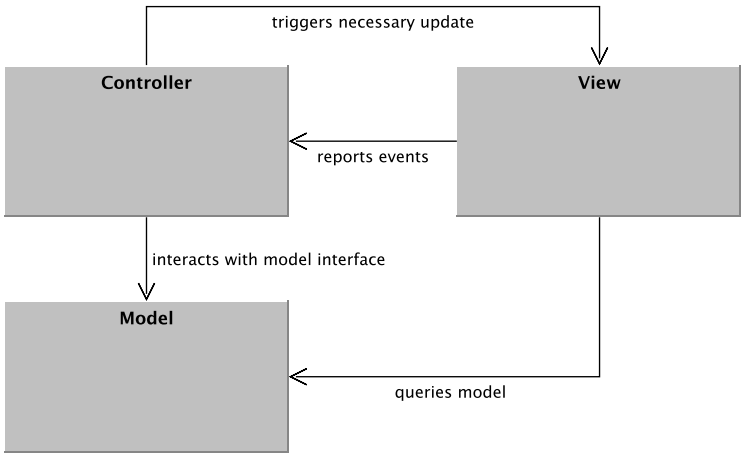


- Grafische Bedienoberflächen unterstützen Bediener in der Bedienung einer Anwendung
- Viele Programmiersprachen bieten hierzu spezielle Bibliotheken an, die grafische Bedienelemente definieren
- JAVA nutzt hierzu u.a. die sogenannte SWING Bibliothek
- Programmieroberfläche am Beispiel einer einfachen Taschenrechner Applikation

39

MVC

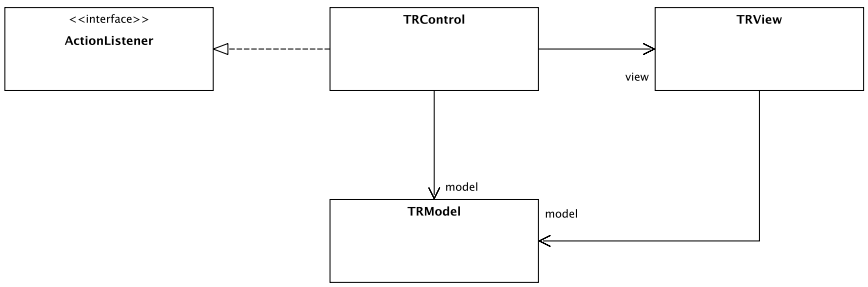
TECHNISCHE HOCHSCHULE LÜBECK



40

MVC Ausprägung unseres Taschenrechners

TECHNISCHE HOCHSCHULE LÜBECK



41

Model unseres Taschenrechners

TECHNISCHE HOCHSCHULE LÜBECK

- Das Model umfasst
  - die logische Datenhaltung,
  - die Taschenrechnerlogik (Businesslogik) und
  - die Interaktionsmöglichkeiten mit dem Taschenrechnerkonzept (Schnittstelle).
- Das Model soll vollkommen unabhängig von der Darstellung (View) oder der Programmsteuerung (Controller) sein.

42

Logisches Modell unseres Taschenrechners Hier: Datenhaltung

TECHNISCHE HOCHSCHULE LÜBECK

- Unser Rechner soll intern vier **Register** haben, die von außen nicht manipulierbar sein sollen:
  - **Result** zum Speichern von Rechenergebnissen
  - **Operand** zum Speichern einer Eingabe (Operanden).
  - **Operator** zum Speichern eines eingegeben Operators. Ein Operator kann +, -, * und / annehmen.
  - **Error** zum Speichern des letzten aufgetretenen Fehlers
- Um das Taschenrechnerkonzept möglichst in vielfältigen Rechnerarchitekturen einsetzen zu können, sollen die Register ganzzahlige Werte und Fließkomma Werte als Stringrepräsentationen speichern, da dieses Datenformat auf nahezu allen Systemen ausgewertet werden kann.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
43

43

Logisches Modell unseres Taschenrechners Hier: Interaktion mit TR Konzept (I)

TECHNISCHE HOCHSCHULE LÜBECK

Damit das TR Konzept zum Berechnen genutzt werden kann, muss es eine Interaktionsschnittstelle anbieten.

berechne()
- Berechnet das Ergebnis aus Result Operator und Operand und speichert dieses in result ab. Ggf. wird bei Fehlern das error Register gesetzt.

getOperand()
- Liest den aktuell im Taschenrechner gesetzten Operanden aus.

setOperand()
- Setzt einen neuen Operanden im Operand Register des TR.

getResult()
- Gibt das Ergebnis aus dem Result Register zurück.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
44

44

Logisches Modell unseres Taschenrechners
Hier: Interaktion mit TR Konzept (II)

TECHNISCHE HOCHSCHULE LÜBECK

Damit das TR Konzept zum Berechnen genutzt werden kann, muss es eine Interaktionsschnittstelle anbieten.

getOperator()
- Liest den aktuell im TR gesetzten Operator aus.

setOperator()
- Setzt einen neuen Operator im operator Register des TR. Stößt ggf. Berechnungen bzw. umspeicheroperationen in den Registern durch, falls erforderlich.

getError()
- Liest den aktuell im TR gesetzten Error aus dem Error Register aus.

clear()
- Setzt alle Register im Taschenrechner auf den Initialzustand zurück.

45

berechne()

TECHNISCHE HOCHSCHULE LÜBECK

```
public void berechne() {
  try {
    // Resultat, Operator oder Operand liegen nicht vor => tue nichts
    if (this.result.equals("") || this.operator.equals("") || this.operand.equals(""))
      return;

    // Ab hier normale Verarbeitung
    float a = Float.valueOf(this.result);
    float b = Float.valueOf(this.operand);

    if (this.operator.equals("+")) this.result = String.valueOf(a + b);
    if (this.operator.equals("-")) this.result = String.valueOf(a - b);
    if (this.operator.equals("/")) {
    // Nicht durch Null teilen
    if (b == 0.0) throw new Exception("Division by Zero");
      this.result = String.valueOf(a / b);
    }
    if (this.operator.equals("*")) this.result = String.valueOf(a * b);

    this.operator = "";
    this.operand = "";
    this.error = "";
  } catch (Exception ex) {
    this.clear();
    this.error = ex.getMessage();
  }
}
```


46

setOperator()

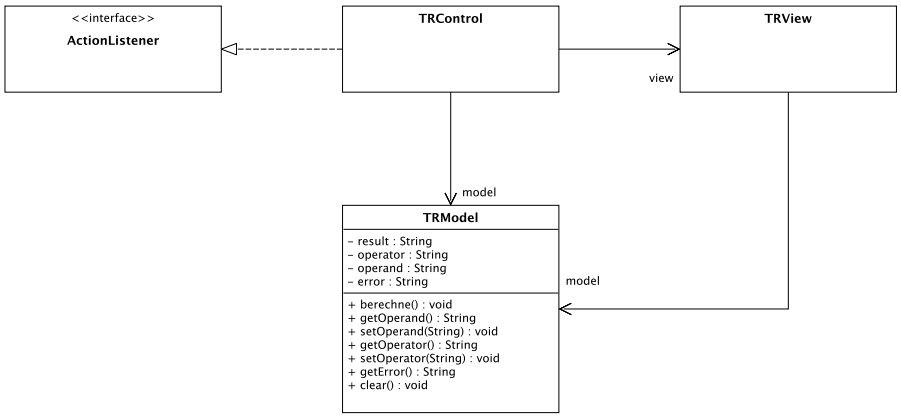
TECHNISCHE HOCHSCHULE LÜBECK

```
public void setOperator(String op) {
  // Resultat, Operator und Operand existieren aus vorherigen Eingaben => erstmal
  // rechnen
  if (!(this.result.equals("") && this.operator.equals("") &&
        this.operand.equals("")))
  {
    this.berechne();
    if (!this.getError().equals("")) { return; }
    // Wenn Fehler aufgetreten, Methode verlassen
  }

  // Es wurde bereits ein Operand eingegeben => diesen zum Resultat machen
  if (!this.operand.equals("")) {
    this.result = this.operand;
    this.operand = "";
  }

  // Es liegt kein Resultat vor => Resultat auf Null setzen
  if (this.result.equals("")) { this.result = "0"; }

  this.operator = op;
  this.error = "";
}
```


47

Verfeinerte MVC Ausprägung unseres Taschenrechners (Model Details)

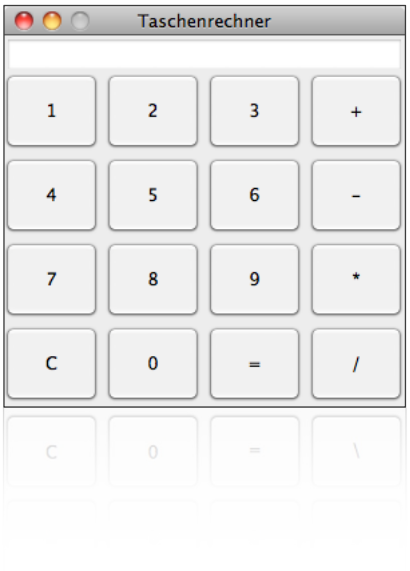
TECHNISCHE HOCHSCHULE LÜBECK



48

Darstellung unseres Taschenrechners
Das View Konzept

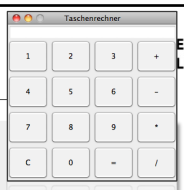
TECHNISCHE HOCHSCHULE LÜBECK



- Unser Taschenrechner soll eine Displayzeile und die üblichen Bedienelemente eines Taschenrechners im üblichen Layout (Zahlen 0 bis 9, und die Operatoren *, /, +, - und =) haben.
- Der View legt diese Elemente nur an, überwacht werden sie von externen Controller.
- Der View kann für ein Darstellungsupdate angestoßen werden, z.B. nach einer durchgeführten Berechnung, einer eingegeben Zahl, etc.

49

Aufbau des Views (I)
Datenfelder: Anlegen der View Elemente

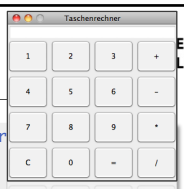



```
public class TRView extends JFrame {
   // Datenfeld des Taschenrechner-Views zur Darstellung des Displays
   private JTextField display = new JTextField();
   {  this.display.setEditable(false); this.display.setSize(200, 60); }

   /* Datenfeld in Form einer Liste, in der alle Tasten des Taschenrechners abgelegt werden
    * Die Tasten werden ueber den entsprechenden Index angesprochen. */
   public List<JButton> buttons = new LinkedList<JButton>();
   {  buttons.add(new JButton ("0")); // Index 0
      buttons.add(new JButton ("1")); // Index 1
      buttons.add(new JButton ("2")); // Index 2
      [...]
      buttons.add(new JButton ("+")); // Index 10
      buttons.add(new JButton ("-")); // Index 11
      [...]
   }

   /* Datenfelder des Views die auf das Modelobjekt und Controllerobjekt des TR verweisen */
   protected TRModel model = new TRModel();
   protected TRControl controller = new TRControl(this, model);
}
```


50

Aufbau des Views (II)

Konstruktor: Platzieren der Bedienelemente im Fenster

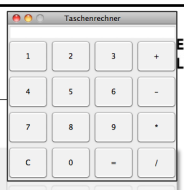


```
/* Konstruktor zum Anlegen eines Viewobjekts eines Taschenrechners. Der Konstr
 * platziert alle Bedienelemente und "verlinkt" diese mit einem Controller */
public TRView() {
  super ("Taschenrechner"); this.setDefaultCloseOperation(EXIT_ON_CLOSE);
  Panel tastenpanel = new Panel(); GridLayout gbLayout = new GridLayout (4, 4);
  gbLayout.setHgap(5); gbLayout.setVgap(5); tastenpanel.setLayout(gbLayout);

  // Zeile 1 des Bedienpanels des TR wird angelegt
  tastenpanel.add (this.buttons.get(1)); tastenpanel.add (this.buttons.get(2));
  tastenpanel.add (this.buttons.get(3)); tastenpanel.add (this.buttons.get(10));
  [...]
  // Zeile 4 des Bedienpanels des TR wird angelegt
  tastenpanel.add (this.buttons.get(15)); tastenpanel.add (this.buttons.get(0));
  tastenpanel.add (this.buttons.get(14)); tastenpanel.add (this.buttons.get(13));
  [...]
  // Display des TR in die erste Zeile setzen. Das Bedienpanel direkt darunter
  this.add(display, BorderLayout.NORTH); this.add(tastenpanel, BorderLayout.CENTER);

  // Alle Tasten des Rechners mit dem Controllerobjekt verknuepfen
  for (JButton b : buttons) { b.addActionListener(controller); }
}
```


51

Aufbau des Views (III)

Methoden: Update eines Views



```
/**
 * Diese Methode wird vom Controller aufgerufen, wenn der View
 * aufdatiert werden soll.
 */
public void update() {
  String result = model.getResult();
  String operator = model.getOperator();
  String operand = model.getOperand();
  String error = model.getError().equals("") ? "" : (model.getError() + "!!!");
  display.setText(result + operator + operand + error);
}
```

Ein Update des Views erfolgt immer aus den darzustellenden Werten des Models und wird üblicherweise vom Controller angestoßen.

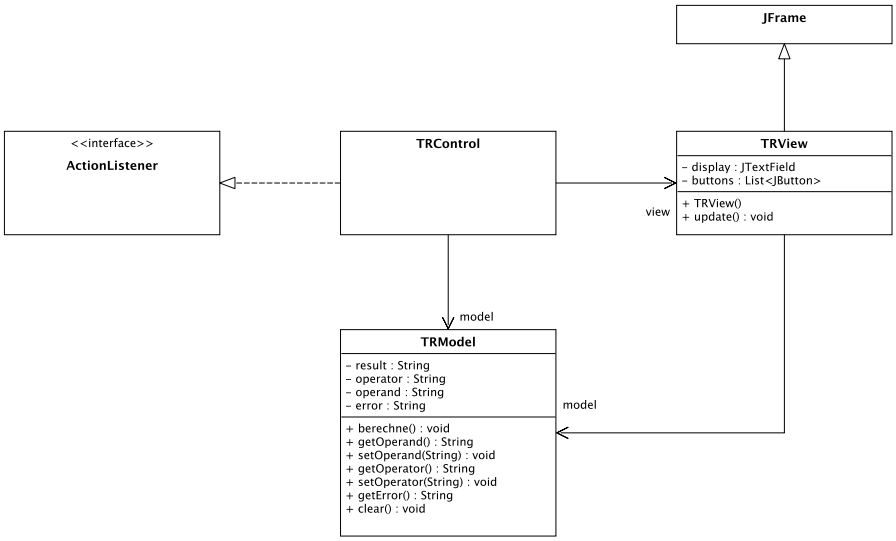
In unserem Fall, lässt sich dies so lösen:

Es wird das Resultat, Operator, Operand Register des Models abgefragt und diese Werte in das Display des Taschenrechners geschrieben.

52

Weiter verfeinerte MVC Ausprägung unseres Taschenrechners (TRView Details)

TECHNISCHE HOCHSCHULE LÜBECK



53

Controller unseres Taschenrechners

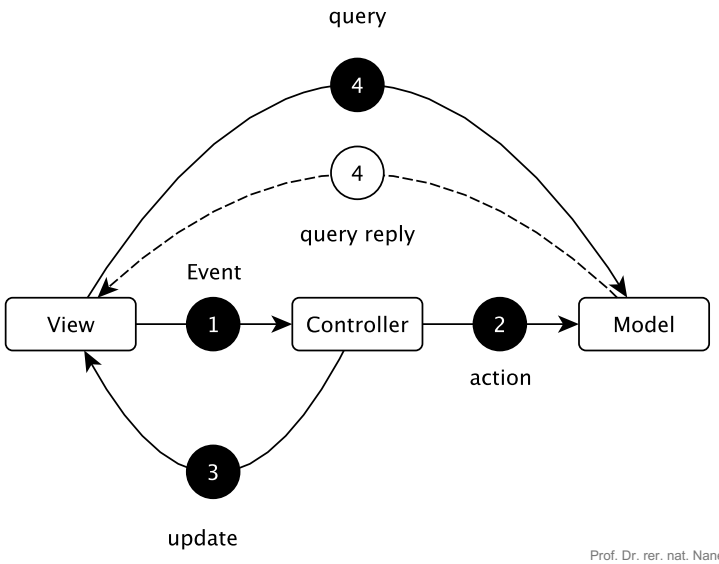
TECHNISCHE HOCHSCHULE LÜBECK

- Ist als „Dirigent“ des MVC-Ensembles verantwortlich für
  - **Wahrnehmen von Bedienerinteraktionen**
  - (z.B. Eingabe einer Zahl, eines Operators)
  - **Übersetzen von Bedienerinteraktionen** in Zustandseingaben an das Model
  - (z.B. Eingabe eines Operators => die gerade eingegebene Zahl ist abgeschlossen)
  - **Veranlassen des Aufdatierens eines Views**, falls dies durch Zustandsänderungen im Model erforderlich wird
  - (z.B. Berechnung wurde durchgeführt => im Display sollte das Ergebnis erscheinen)

54

Ablauf einer Benutzerinteraktion

TECHNISCHE HOCHSCHULE LÜBECK



55

Der Controller verknüpft View und Model

TECHNISCHE HOCHSCHULE LÜBECK

```
public class TRControl implements ActionListener {

   /**
    * Datenfelder des Controller Objekts eines Taschenrechners
    */
   private TRView view;
   private TRModel model;

   /**
    * Konstruktor zum Anlegen eines Taschenrechner Controller Objekts
    * @param v Viewobjekt eines Taschenrechners, dass dieser Controller betreut
    * @param m Modelobjekt eines Taschenrechners, dass dieser Controller betreut
    */
   public TRControl (TRView v, TRModel m) {
      this.view = v;
      this.model = m;
   }
   [...]
}
```


56

Listener Callbacks
Die zentrale Anlaufstelle eines Controllers

TECHNISCHE HOCHSCHULE LÜBECK

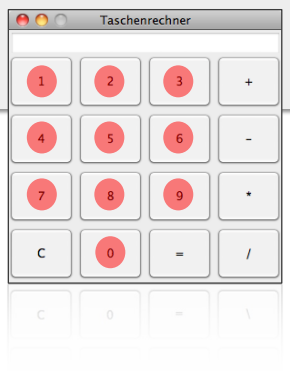
```
/* ActionListener - Diese Methode wird immer aufgerufen, wenn ein Button auf dem TR
 * betaetigt wurde.
 */
public void actionPerformed(ActionEvent ev) {
   if (ev.getSource() == view.buttons.get(0)) zahlAnhaengen("0");     // 0
   if (ev.getSource() == view.buttons.get(1)) zahlAnhaengen("1");     // 1
   if (ev.getSource() == view.buttons.get(2)) zahlAnhaengen("2");     // 2

   [...]

   if (ev.getSource() == view.buttons.get(10)) setRechenzeichen("+"); // Plus
   if (ev.getSource() == view.buttons.get(11)) setRechenzeichen("-"); // Minus
   if (ev.getSource() == view.buttons.get(12)) setRechenzeichen("*"); // Mal
   if (ev.getSource() == view.buttons.get(13)) setRechenzeichen("/"); // Geteilt

   if (ev.getSource() == view.buttons.get(14)) berechnen();           // =
   if (ev.getSource() == view.buttons.get(15)) loeschen();            // C
}
```


57

Zahleingeben (Zifferntasten)

TECHNISCHE HOCHSCHULE LÜBECK

```
/**
 * Wird aufgerufen, wenn eine Zahl auf dem Taschenrechner betaetigt wurde
 * Diese Zahl wird der aktuell auf dem Display stehenden Zahl angehaengt
 * @param i Die Ziffer die an den aktuell eingegebenen Operanden angehaengt werden soll
 */
private void zahlAnhaengen (String i) {
   model.setOperand(model.getOperand() + i);
   view.update();
}
```



(1) Hängt an den aktuell eingegebenen Operator eine weitere Ziffer an und ändert damit den Modelzustand des Taschenrechners (Operand-Register).

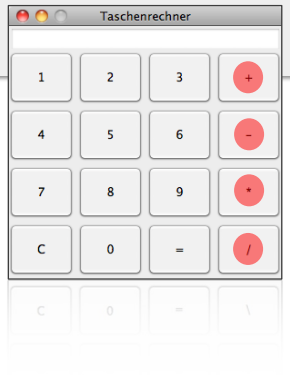
(2) Stößt daher anschließend ein Update des Views an, um den Modelzustand durch den View auszulesen und in der Displayzeile darzustellen.

58

Operatoreingabe (+, -, *, / Tasten)

TECHNISCHE HOCHSCHULE LÜBECK

```
/**
 * Wird aufgerufen, wenn eine Operatoraste, -, /, *) betaetigt wurde
 * @param i Der eingegebene Operator (+, -, /, *)
 */
private void setRechenzeichen (String i) {
   model.setOperator(i);
   view.update();
}
```



(1) Setzt im Model des Taschenrechners den anzuwendenden Operator und ändert damit den Modelzustand (Operator-Register).

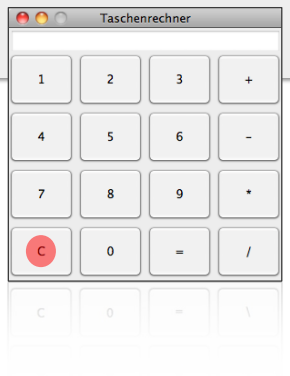
(2) Stößt daher anschließend ein Update des Views an, um den Modelzustand durch den View auszulesen und in der Displayzeile darzustellen.

59

Löschen (C Taste)

TECHNISCHE HOCHSCHULE LÜBECK

```
/**
 * Wird aufgerufen, wenn die C Taste auf einem Taschenrechner
 * betaetigt wurde.
 */
private void loeschen() {
  model.clear();
  view.update();
}
```



(1) Löscht alle Register im Model. Setzt den Taschenrechner also wieder in den Ursprungszustand.

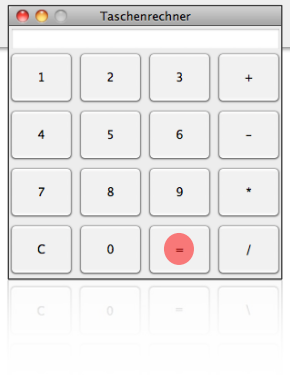
(2) Stößt daher anschließend ein Update des Views an, um den Modelzustand durch den View auszulesen und in der Displayzeile darzustellen.

60

Auswertung (= Taste)

TECHNISCHE HOCHSCHULE LÜBECK

```
/**
 * Wird aufgerufen, wenn die = Taste auf einem Taschenrechner betaetigt wurde.
 */
private void berechnen() {
   model.berechne();
   view.update();
}
```



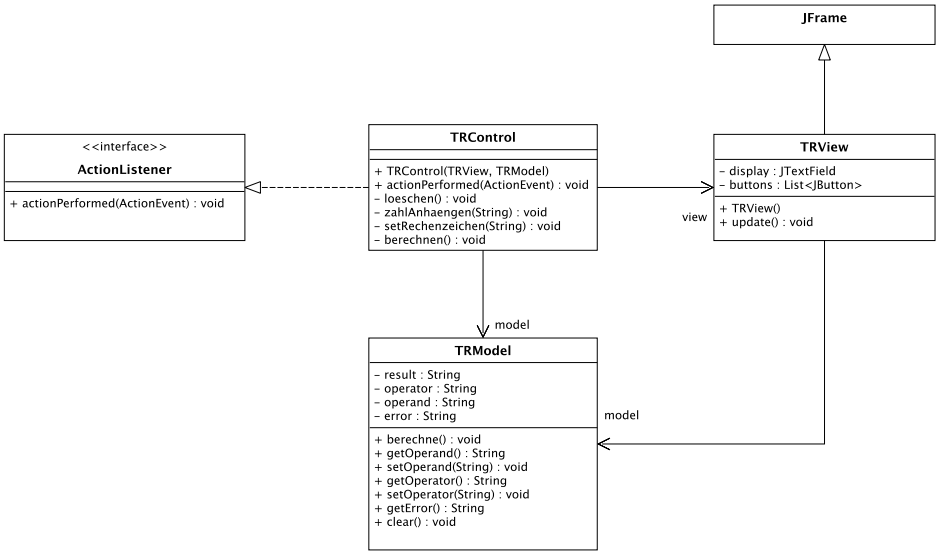
(1) Stößt die Berechnung des Models an. Dieses ändert die modelinternen Registerzustände des Taschenrechners.

(2) Stößt daher anschließend ein Update des Views an, um den Modelzustand durch den View auszulesen und in der Displayzeile darzustellen.

61

Weiter verfeinerte MVC Ausprägung unseres Taschenrechners (TRControl Details)

TECHNISCHE HOCHSCHULE LÜBECK



62

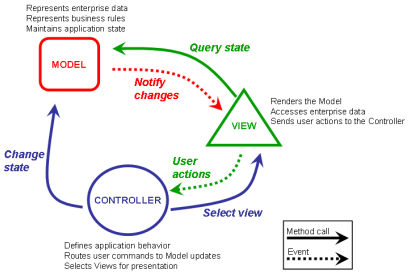
MVC – ein häufig genutztes Pattern

TECHNISCHE HOCHSCHULE LÜBECK

Das MVC Pattern ist nicht nur auf einen Taschenrechner oder die Programmiersprache JAVA begrenzt. Es kann als grundlegendes Muster für alle Applikationen mit Graphical User Interfaces (GUIs) genutzt werden.

Mehr finden Sie z.B. hier:

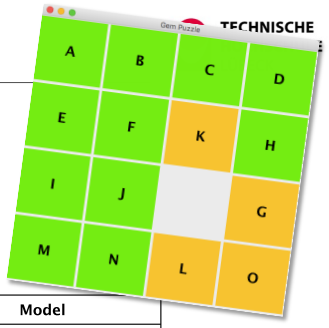
http://de.wikipedia.org/wiki/Model_View_Controller

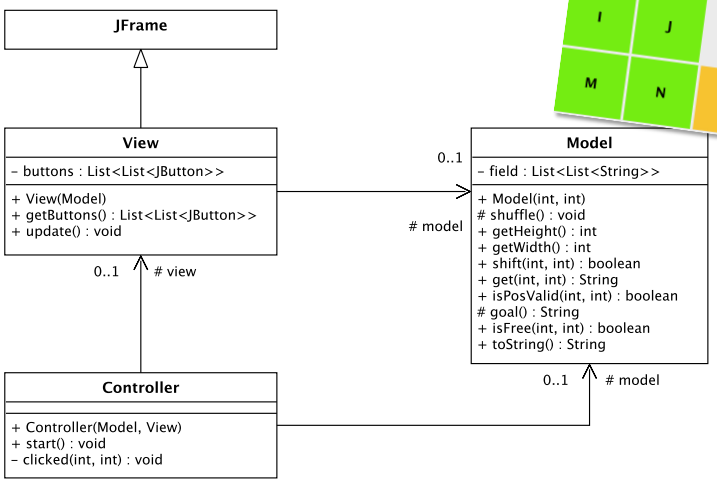


Quelle: Wikipedia

63

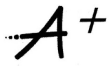
Nun: Ein Spiel ...





64

Zusammenfassung



TECHNISCHE HOCHSCHULE LÜBECK

- **Graphical User Interfaces**
  - Am Beispiel eines Taschenrechners
  - Nutzung des Model View Controller Paradigmas
- **Taschenrechner Modell**
  - Vier Register (Result, Operand, Operator, Status)
  - Interaktionsmethoden (insbesondere `setter` und `berechne` Methode)
- **Taschenrechner View**
  - Events verbinden
  - Quellen (GUI Komponenten) und
  - Senken (Controller)
- **Taschenrechner Controller**
  - Listener Callbacks (hier Methode `actionPerformed`)
  - Bedieneraktion auswerten und in Modelmethoden (Interaktionen) übersetzen
  - View Update anstoßen