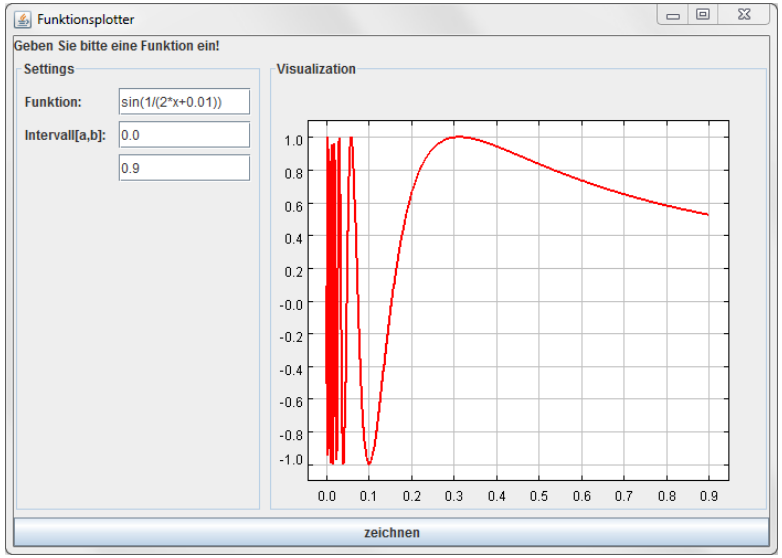
RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

Sommersemester 2023

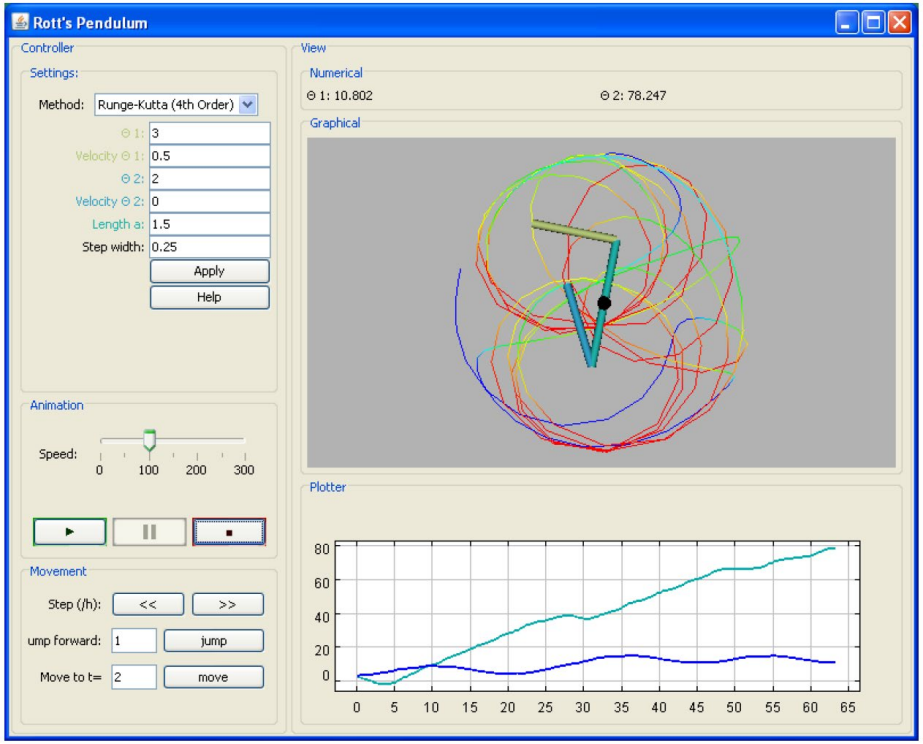
Graphische Benutzeroberflächen



Benutzeroberflächen

Einleitung

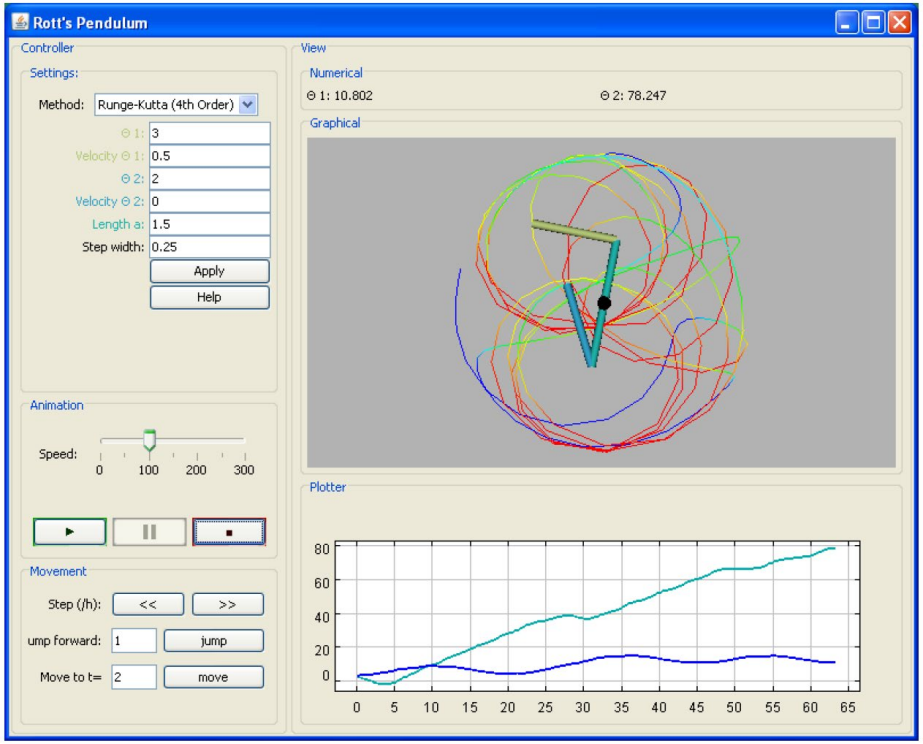
- Benutzeroberflächen dienen der Interaktion zwischen Programm und Anwender
  - Eingabe von Daten
  - Starten von Aktionen
  - Ausgabe von Ergebnissen



Benutzeroberflächen

Beispiel Simulation eines Pendels

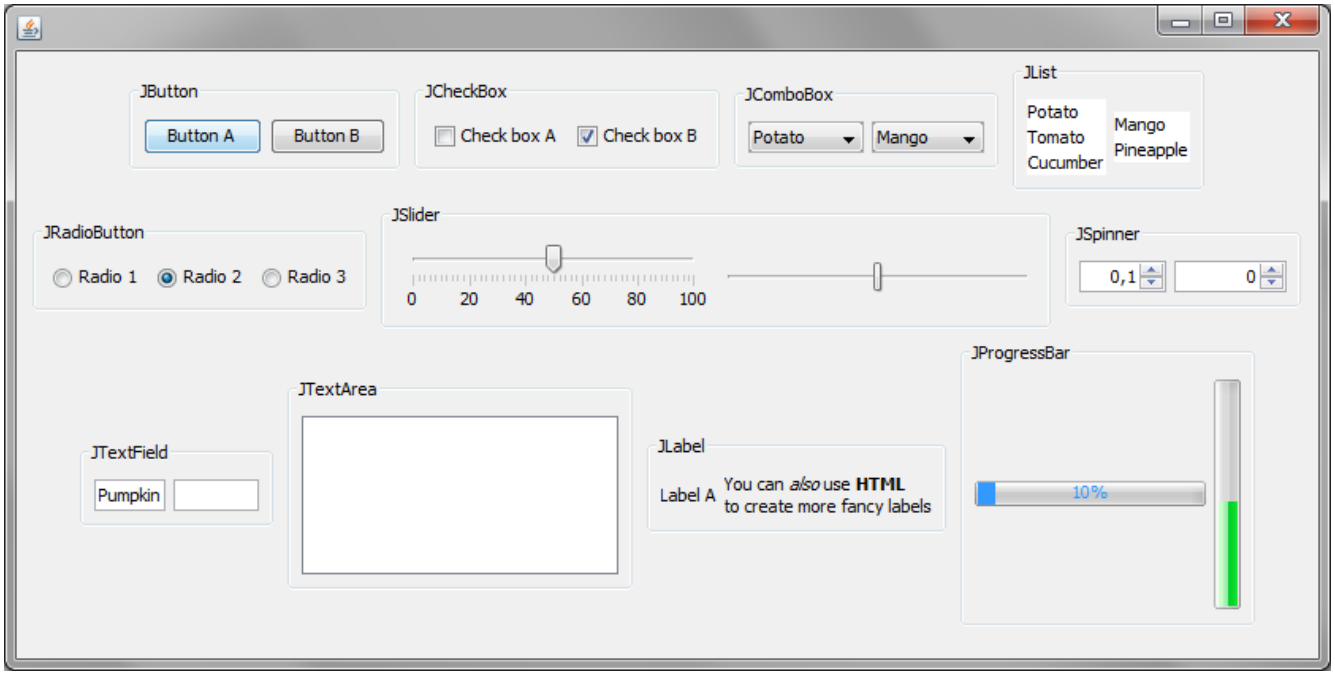
- Anwendungsfenster
- Animation der Bewegung
- Ausgabe eines Diagramms
- Eingabe der Elementdaten
- Auswahl eines Berechnungsverfahrens
- Steuerung der Animation



Benutzeroberflächen

Komponententechnik

- Eine graphische Benutzeroberfläche wird aus grafischen Komponenten aufgebaut.
- Java stellt hierfür die Swing-Bibliothek zur Verfügung.
  - Alternativen in Java sind AWT, SWT, und JavaFX.



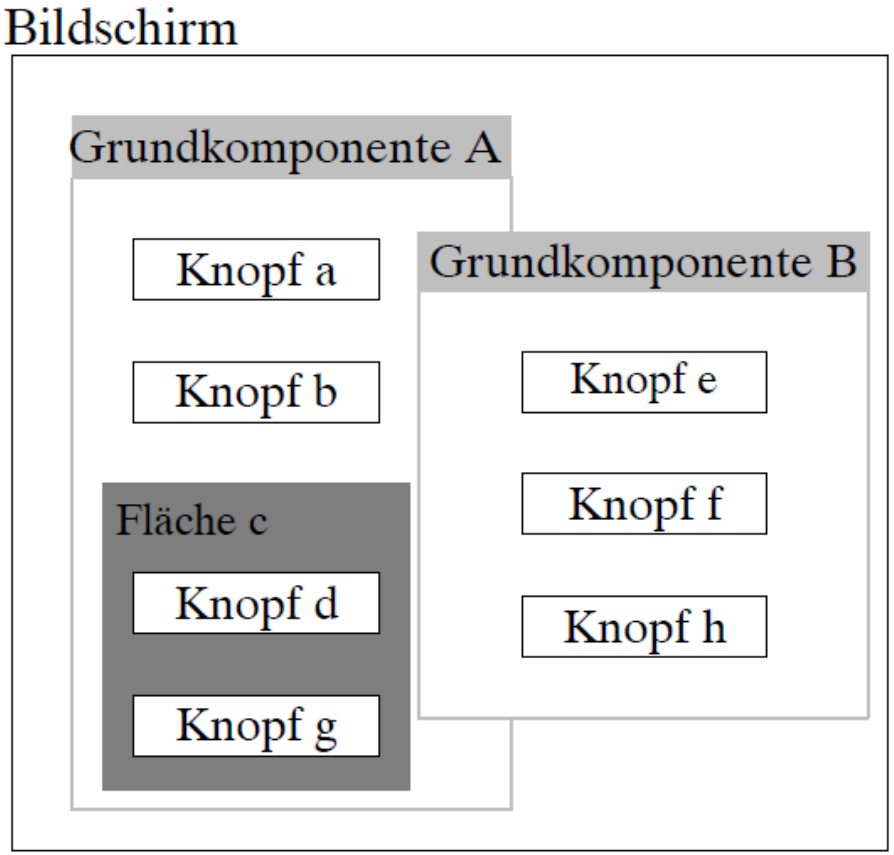
Benutzeroberflächen

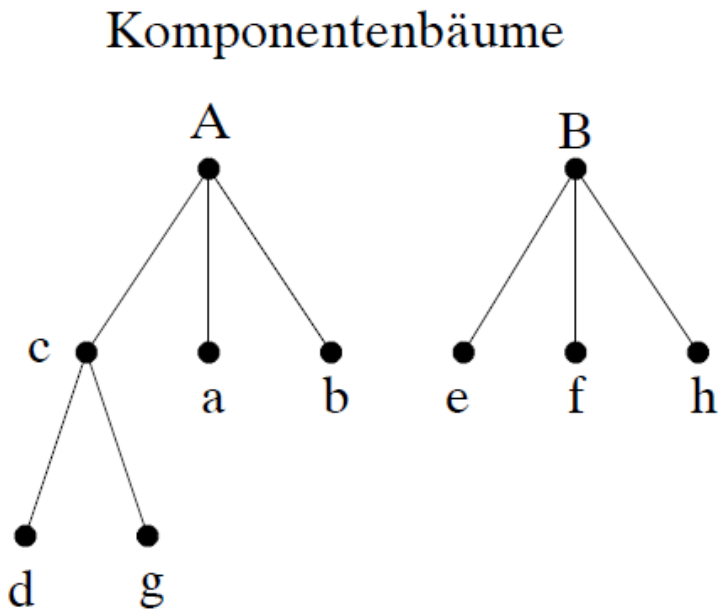
Komponententypen

- Eine **Komponente** ist ein Objekt einer Klasse zur Darstellung oder zur Organisation einer Benutzeroberfläche.
- Ein **Container** ist eine spezielle Komponente, die andere Komponenten aufnehmen und gruppieren kann.
- Ein **Fenster** ist eine spezielle Komponente, die einen Container besitzt und mit dem Window-Manager des Betriebssystems kommuniziert.
- Für jedes Fenster kann ein **Komponentenbaum** aufgebaut werden, der sehr wichtig für die Darstellung ist (siehe Composite-Muster).

Benutzeroberflächen

Komponententypen

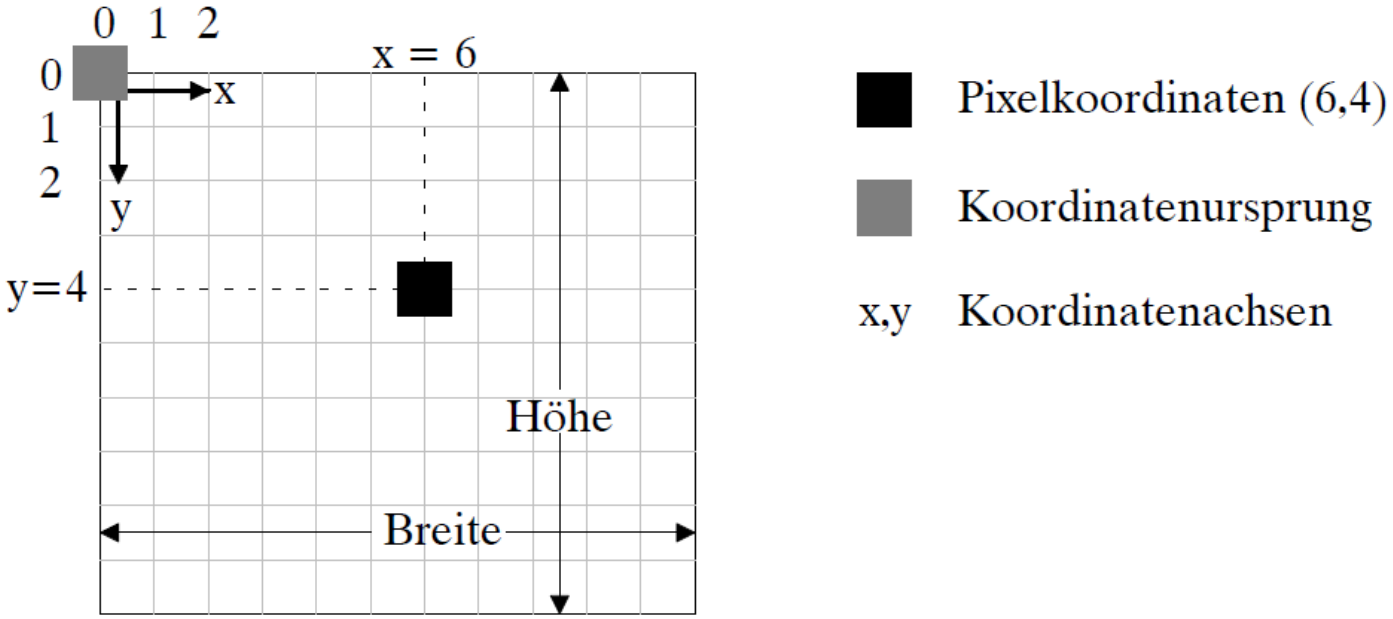




Benutzeroberflächen

Komponentenattribute

- Abmessungen einer Komponente basieren auf einer rechteckigen Fläche, welche in Pixeln angegeben wird.
- Der Ursprung einer Komponente liegt in der linken oberen Ecke.



Benutzeroberflächen

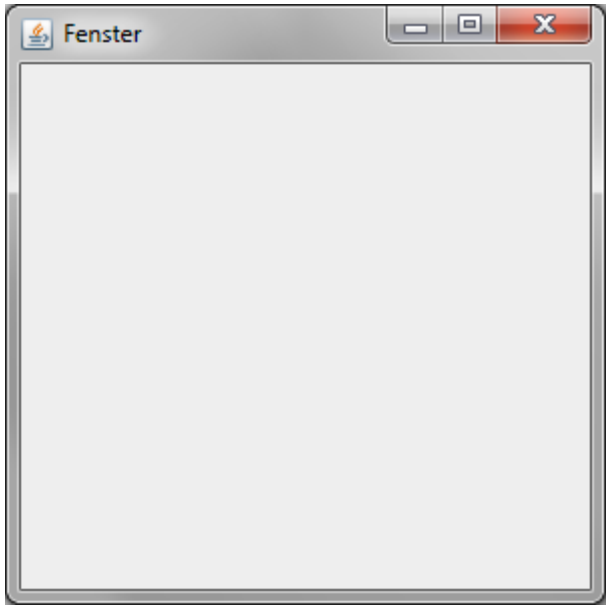
Komponentenzustände

- Jede Komponente befindet sich zu jedem Zeitpunkt in einem eindeutigen Zustand.
- Der **abbildbare** Zustand ist explizit zu setzen und kann auf die enthaltenden Komponenten übertragen werden.
- Eine abbildbare Komponente ist nur **darstellbar**, wenn alle Vorgänger der Komponente abbildbar sind.
- Eine darstellbare Komponente ist **sichtbar**, wenn sie nicht vollständig durch eine andere Komponente verdeckt wird.

Benutzeroberflächen

Fenster

- Ein einfaches Fenster einschließlich Fensterrahmen wird mit Hilfe der Klasse `JFrame` erzeugt:
  - Fenstertitel angeben (Konstruktor)
  - Fenstergröße setzen
  - Fenster freigeben beim Schließen
  - Fenster abbildbar setzen

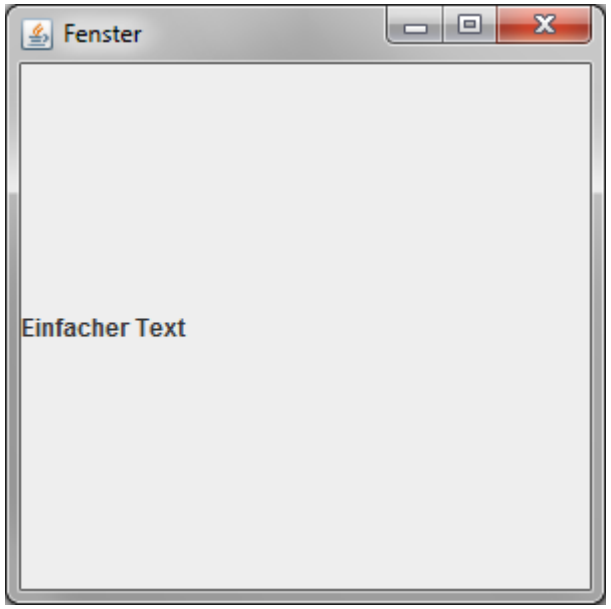



```
public static void createAndShowGUI() {

    JFrame f = new JFrame("Fenster");
    f.setSize(300, 300);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setVisible(true);
}
```

Benutzeroberflächen

Komponente JLabel

- Eine Beschriftung kann über die Klasse `JLabel` erzeugt werden:
  - Beschriftung kann geändert werden
  - Objekt muss einem Container hinzugefügt werden
  - Komponenten in einem Container werden in einem bestimmten Layout angeordnet

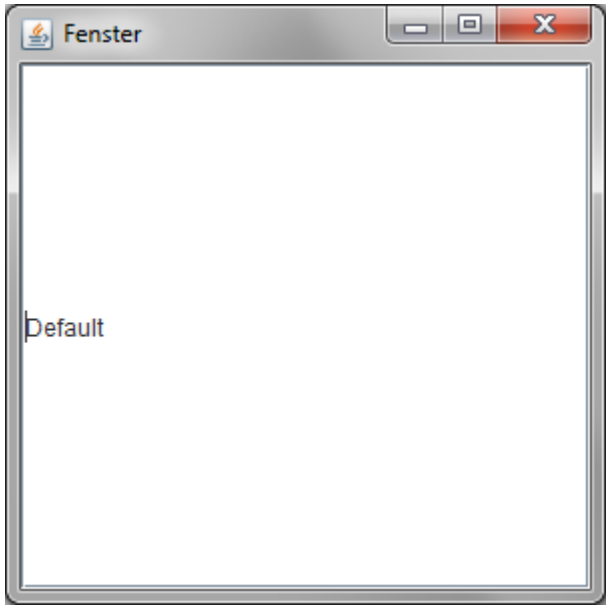



```
JFrame f = new JFrame("Fenster");
f.setSize(300, 300);

JLabel l1 = new JLabel("Einfacher Text");
f.getContentPane().add(l1);

f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
f.setVisible(true);
```

Benutzeroberflächen

Komponente JTextField

- Ein Text kann mit Hilfe eines Objektes der Klasse `JTextField` erfasst werden:
  - Text kann gesetzt und abgefragt werden
  - Größe richtet sich nach dem Layout des Containers




```
JFrame f = new JFrame("Fenster");
f.setSize(300, 300);

JTextField t1 = new JTextField("Default");
f.getContentPane().add(t1);

System.out.println(t1.getText());
//...
```

Benutzeroberflächen

Komponente JButton

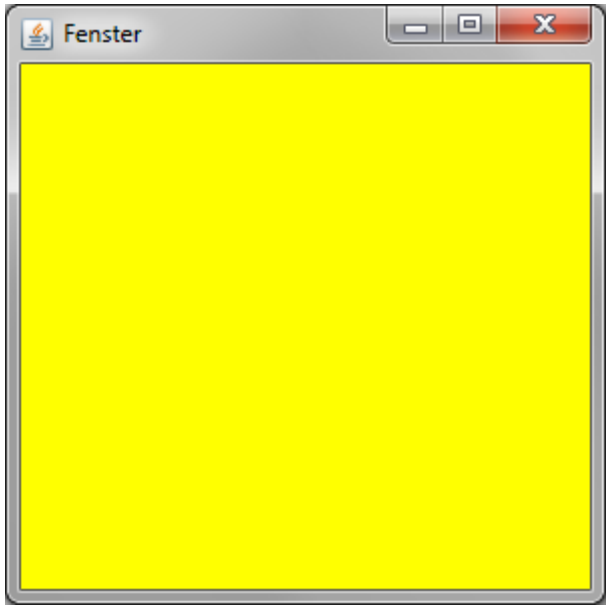
- Eine Schaltfläche wird mit Hilfe der Klasse `JButton` erzeugt:
  - Beschriftung kann gesetzt werden
  - Größe richtet sich nach dem Layout des Containers


```
JFrame f = new JFrame("Fenster");
f.setSize(300, 300);

JButton b1 = new JButton("starten");
f.getContentPane().add(b1);

//...
```

Benutzeroberflächen

Komponente JPanel

- Zur Organisation von Komponenten kann die Klasse `JPanel` verwendet werden:
  - Container mit rechteckiger Fläche
  - Es kein ein bestimmtes Layout festgelegt werden
  - Klasse `Color` stellt Farben zur Verfügung




```
JFrame f = new JFrame("Fenster");
f.setSize(300, 300);

JPanel p1 = new JPanel();
p1.setBackground(Color.YELLOW);
f.getContentPane().add(p1);

//...
```

Applikation starten

GUIs in Java laufen auf einem nebenläufigen Event-Thread auf dem auch alle Events ausgeführt werden. Applikationen sollten daher GUI-Elemente nur auf diesem Thread erstellen:

```
public static void main(String[] args) {
    SwingUtilities.invokeAndWait(new Runnable() {
        public void run() {
                createAndShowGUI();
        }
    });
}
```

Benutzeroberflächen

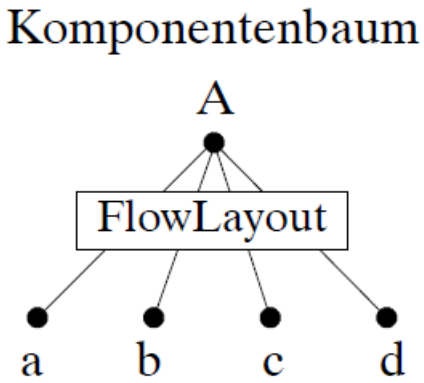
Anordnung von Komponenten

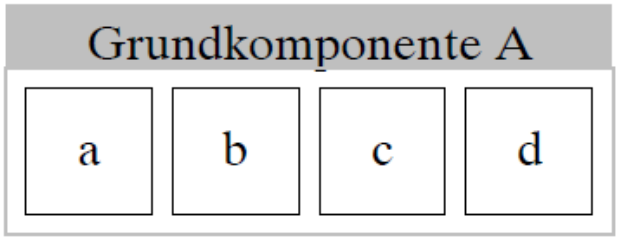
- Die Anordnung von Komponenten in einem Container erfolgt mit Hilfe eines **Layout-Manager**s.
- Es ist keine absolute Positionierung notwendig, die Komponenten versuchen jeweils einem bestimmten Layout zu folgen.
- Die Größe der einzelnen Komponenten wird, falls erlaubt, automatisch angepasst.
- In Java werden verschiedene Layout-Manager standardmäßig zur Verfügung gestellt, z.B.
  - `FlowLayout`
  - `BorderLayout`
  - `GridLayout`

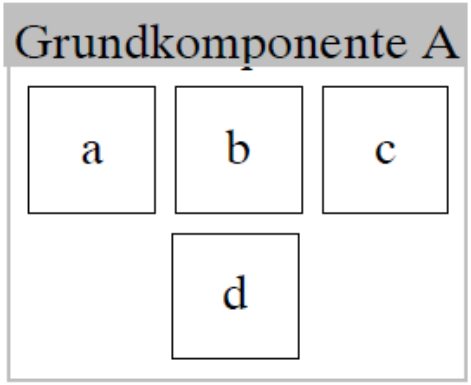
Benutzeroberflächen

FlowLayout

- Die Klasse `FlowLayout` stellt einen Layout-Manager bereit, der die Komponenten in einer Zeile anordnet:
  - Reihenfolge der Komponenten ist wichtig
  - Reicht die Breite nicht aus, wird eine neue Zeile erzeugt



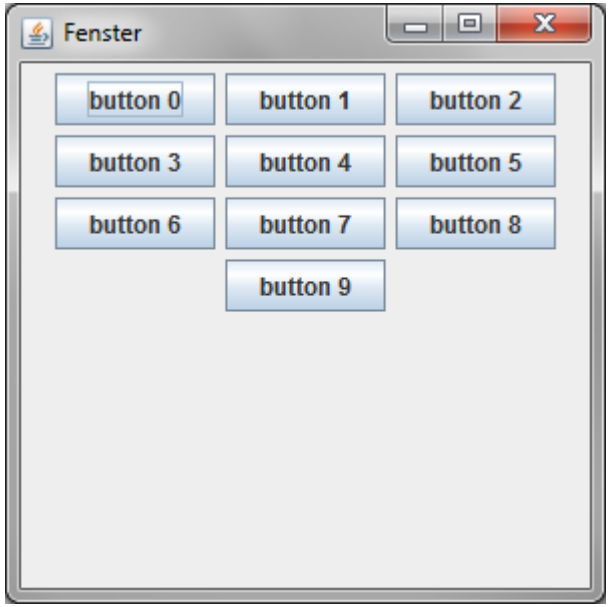




Benutzeroberflächen

Beispiel FlowLayout

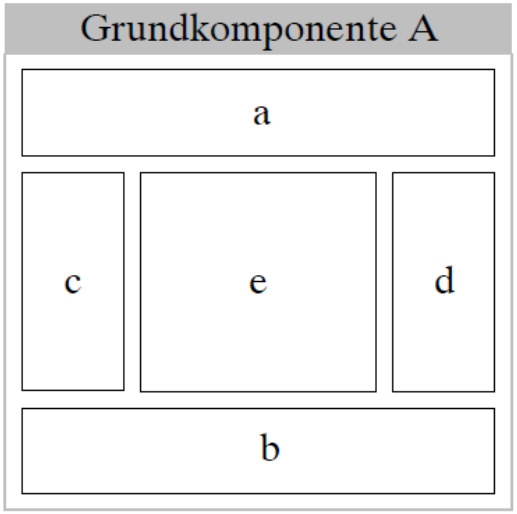
```
public class FlowLayoutExample01 {

 public static void createAndShowGUI(){

    JFrame f = new JFrame("Fenster");
    f.setSize(300, 300);

    f.getContentPane().setLayout(new FlowLayout());

    for (int i=0; i<10; i++) {
      f.getContentPane().add(
         new JButton("button "+i));
    }

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

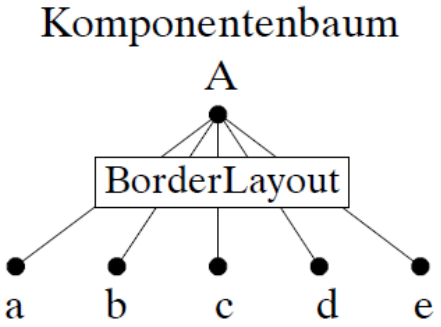


Benutzeroberflächen

BorderLayout

- Die Klasse `BorderLayout` stellt einen Layout-Manager bereit, der fünf Komponenten bzgl. der Himmelsrichtungen und dem Zentrum anordnen kann.
  - Werden weniger als fünf Komponenten angeordnet, wird die freie Fläche automatisch aufgeteilt.





Benutzeroberflächen

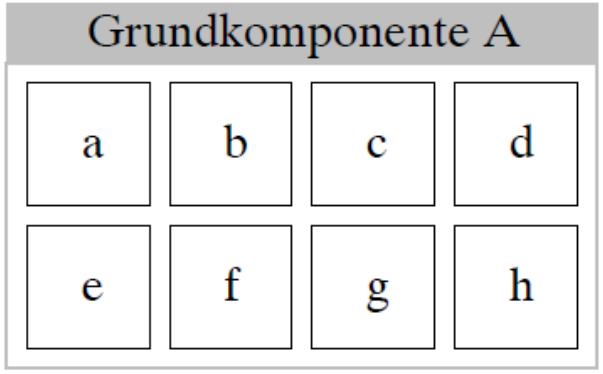
Beispiel BorderLayout

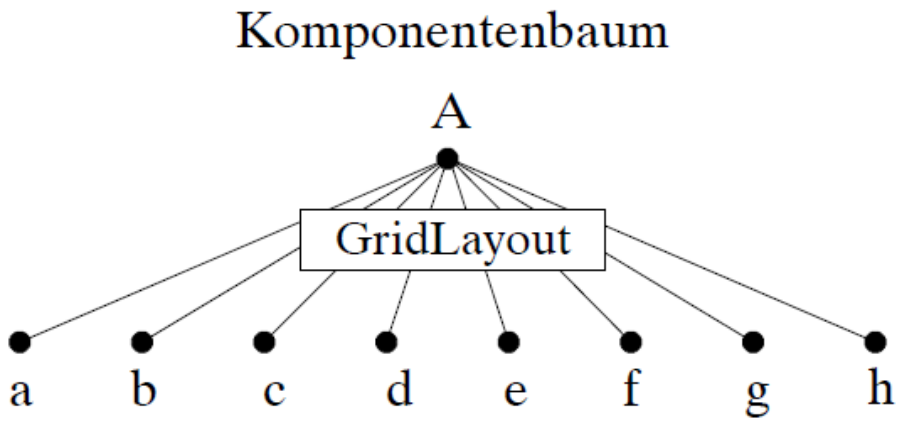
```
JFrame f = new JFrame("Fenster");
f.setSize(300, 300);

f.getContentPane().setLayout(new BorderLayout());

f.getContentPane().add(new JButton("oben"),
     BorderLayout.NORTH);

f.getContentPane().add(new JButton("unten"),
     BorderLayout.SOUTH);

f.getContentPane().add(new JButton("links"),
     BorderLayout.WEST);

f.getContentPane().add(new JButton("rechts"),
     BorderLayout.EAST);

f.getContentPane().add(new JButton("mitte"),
     BorderLayout.CENTER);
```

Benutzeroberflächen

GridLayout

- Die Klasse `GridLayout` stellt einen Layout-Manager bereit, der Komponenten in einem orthogonalen Raster anordnen kann.

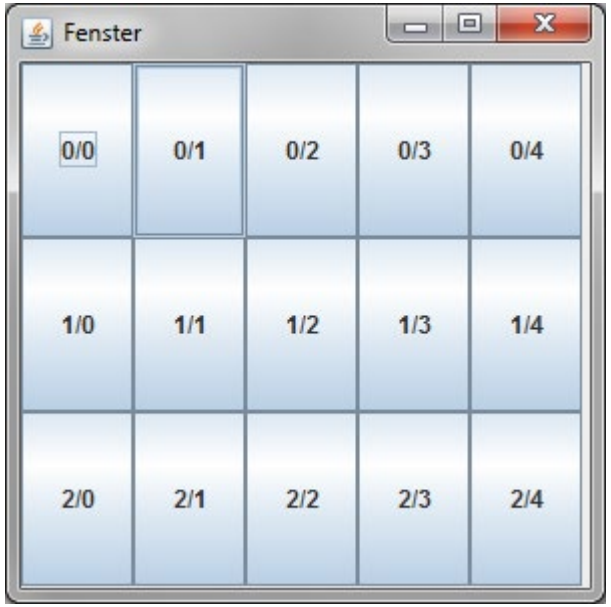




Benutzeroberflächen

Beispiel GridLayout

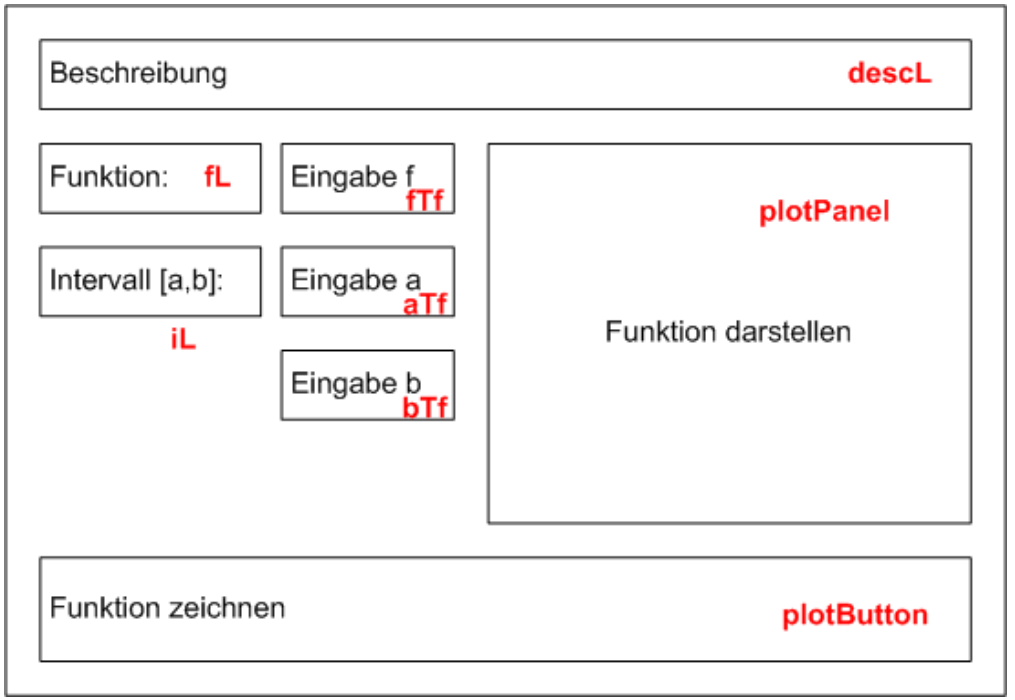
```
JFrame f = new JFrame("Fenster");
f.setSize(300, 300);

f.getContentPane().setLayout(
     new GridLayout(3, 5));

for (int i=0; i<3; i++) {
  for (int j=0; j<5; j++) {
    f.getContentPane().add(new JButton(i+"/"+j));
  }
}
```



Benutzeroberflächen

Design einer Benutzeroberfläche

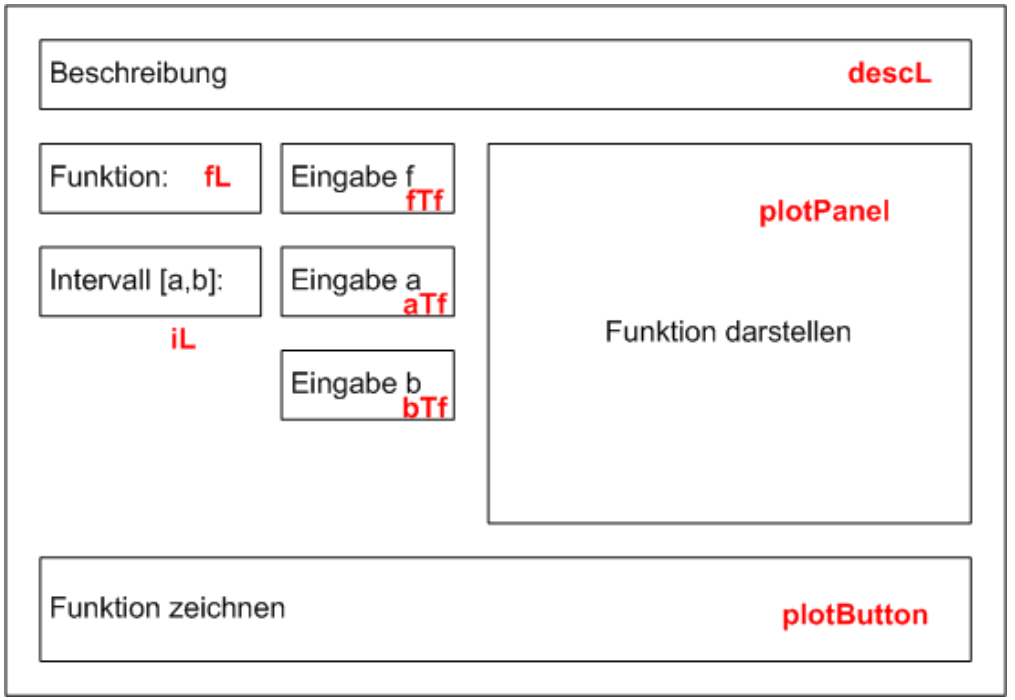
- Ein gutes Design erfordert eine sinnvolle Komponentenanordnung mit Hilfe verschiedener Container.
- Beispiel: Funktion darstellen
  - Labels `descL`, `fL`, `iL`
  - Textfelder `fTf`, `aTf`, `bTf`
  - Button `plotButton`
  - ViewerPanel `plotPanel`



Benutzeroberflächen

Design einer Benutzeroberfläche

- Ein gutes Design erfordert eine sinnvolle Komponentenanordnung mit Hilfe verschiedener Container
- Beispiel: Funktion darstellen
  - Verschiedene Panels mit unterschiedlichen Layout-Manager müssen verwendet werden (Beispiel an der Tafel)



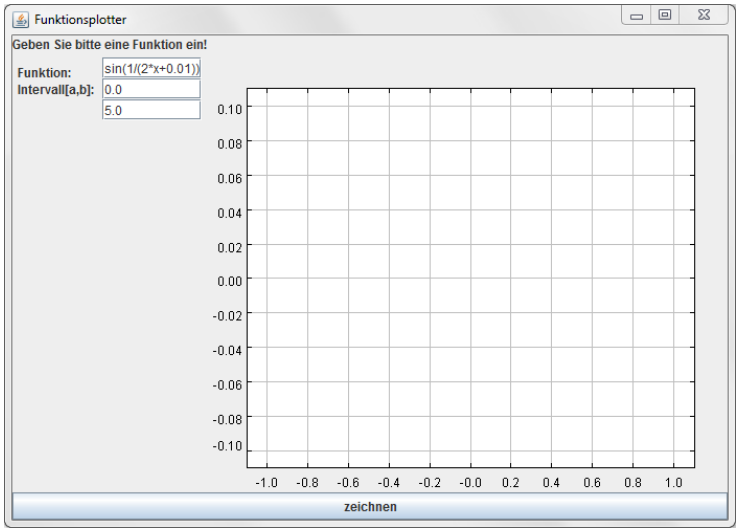
Eigene GUI-Klassen

- In der Regel werden eigene Klassen zur Erstellung komplexer Benutzeroberflächen verwendet, z.B.
  - Klassen für spezielle Fenster (Ableitung von der Klasse `JFrame`)
  - Klassen für bestimmte Darstellungsbereiche (Ableitung von der Klasse `JPanel`)
- Beispiel: Funktion darstellen
  - Klasse `FunctionPlotFrame` für das Fenster mit speziellen Komponenten
  - Klasse `FunctionPlotPanel` zur Visualisierung der Funktion (hier gegebene Klasse)

Benutzeroberflächen

Beispiel Function1DFrame

```
public class FunctionPlotFrameDemo {

 public static void createAndShowGUI(){
    FunctionPlotFrame f = new FunctionPlotFrame("Funktionsplotter");
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setVisible(true);
  }
}
```



Benutzeroberflächen

Beispiel Function1DFrame

```
public class FunctionPlotFrame extends JFrame {

  private JLabel descL, fL, iL;
  private JTextField fTf, aTf, bTf;
  private JButton plotButton;
  private FunctionPlotPanel plotPanel;


  public FunctionPlotFrame(String title) {
    super(title);
    this.init();
    this.setSize(700, 500);
  }

  // ...
}
```

Komponenten als Attribute

Spezielle Methode zum Aufbau der Benutzeroberfläche

Größe kann auch im Konstruktor gesetzt werden

Benutzeroberflächen

Beispiel Function1DFrame

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // Generelles Layout setzen
    this.getContentPane().setLayout(new BorderLayout());
    // Beschreibung erzeugen
    this.descL = new JLabel("Geben Sie bitte eine Funktion ein!");
    this.getContentPane().add(this.descL, BorderLayout.NORTH);
    // Label für Funktion und Intervall
    this.fL = new JLabel("Funktion: ");
    this.iL = new JLabel("Intervall[a,b]: ");
    // Labels in ein Panel einfügen
    JPanel gridLabels = new JPanel();
    gridLabels.setLayout(new GridLayout(3,1));
    gridLabels.add(this.fL);
    gridLabels.add(this.iL);

    // ...
  }
}
```

Benutzeroberflächen

Beispiel Function1DFrame

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // ..

    // Eingabefelder erzeugen
    this.fTf = new JTextField("sin(1/(2*x+0.01))");
    this.aTf = new JTextField("0.0");
    this.bTf = new JTextField("5.0");
    // Eingabefelder in einem eigenen Panel einfügen
    JPanel gridTextFields = new JPanel();
    gridTextFields.setLayout(new GridLayout(3,1));
    gridTextFields.add(this.fTf);
    gridTextFields.add(this.aTf);
    gridTextFields.add(this.bTf);

    // ...
}
```

Benutzeroberflächen

Beispiel Function1DFrame

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // ..

    // Beschriftung und Eingabenfelder in einem eigenen Panel
    JPanel inputPanel = new JPanel();
    inputPanel.setLayout(new FlowLayout());
    inputPanel.add(gridLabels);
    inputPanel.add(gridTextFields);
    // Diese Panel einfügen
    this.getContentPane().add(inputPanel, BorderLayout.WEST);
    // Spezielles Panel zur Darstellung der Funktion erzeugen
    this.plotPanel = new Function1DPanel();
    this.getContentPane().add(this.plotPanel, BorderLayout.CENTER);
    // Button zum Zeichnen der Funktion erzeugen und hinzufügen
    this.plotButton = new JButton("zeichnen");
    this.getContentPane().add(this.plotButton, BorderLayout.SOUTH);
}
```

Ansprechende Benutzeroberflächen

- In Java existieren sehr viele Möglichkeiten, um ansprechende Benutzeroberflächen zu gestalten:
  - Definition von bevorzugten Abmessungen
  - Festlegung von bestimmten Abständen
  - Erstellung von Rahmen
  - ...
- In der Klassenbibliothek `javax.swing` sind weitere Komponenten und Gestaltungselemente zusammengefasst.
- Sehr gute Dokumentationen und Tutoriums sind im Internet, z.B. unter http://java.sun.com/docs/books/tutorial/uiswing/ zu finden.

Benutzeroberflächen

Beispiel Function1DFrame

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // ..

    // Eingabefelder erzeugen
    this.fTf = new JTextField("sin(1/(2*x+0.01))");
    this.fTf.setPreferredSize(new Dimension(120, 25));
    this.aTf = new JTextField("0.0");
    this.bTf = new JTextField("5.0");
    // Eingabefelder in einem eigenen Panel einfügen
    JPanel gridTextFields = new JPanel();
    gridTextFields.setLayout(new GridLayout(3,1,5,5));
    gridTextFields.add(this.fTf);
    gridTextFields.add(this.aTf);
    gridTextFields.add(this.bTf);

    // ...
}
```

Bevorzugte Größe

Abstand zwischen den Zellen

Benutzeroberflächen

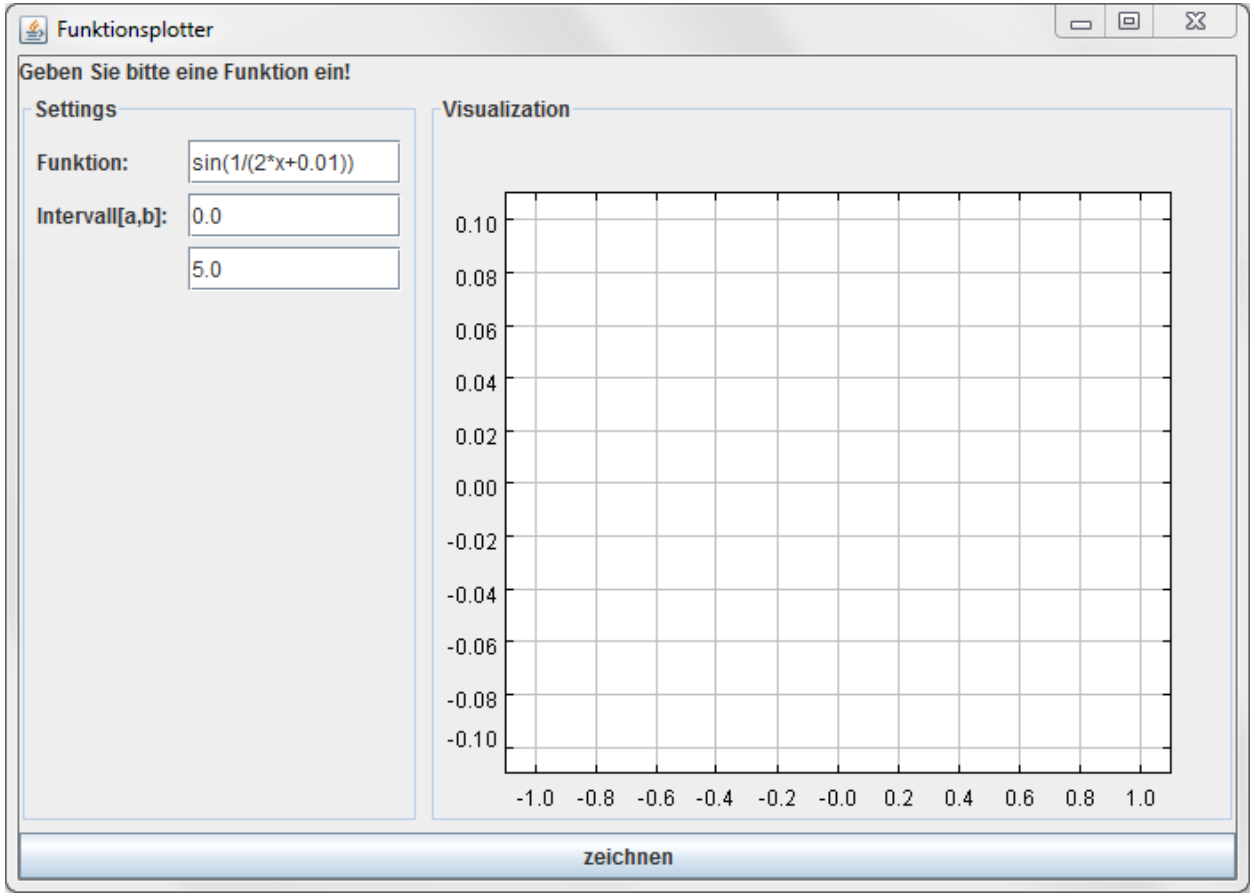
Beispiel Function1DFrame

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // ..

    // Beschriftung und Eingabenfelder in einem eigenen Panel
    JPanel inputPanel = new JPanel();
    inputPanel.setBorder(new TitledBorder("Settings"));
    inputPanel.setLayout(new FlowLayout());
    inputPanel.add(gridLabels);
    inputPanel.add(gridTextFields);
    // Diese Panel einfügen
    this.getContentPane().add(inputPanel, BorderLayout.WEST);
    // Spezielles Panel zur Darstellung der Funktion erzeugen
    this.plotPanel = new FunctionPlotPanel();
    this.getContentPane().add(this.plotPanel, BorderLayout.CENTER);
    // Button zum Zeichnen der Funktion erzeugen und hinzufügen
    this.plotButton = new JButton("zeichnen");
    this.getContentPane().add(this.plotButton, BorderLayout.SOUTH);
}
```

Rahmen mit Beschriftung für das Panel erzeugen

Benutzeroberflächen

Beispiel Function1DFrame



Benutzeroberflächen

Interaktivität

- Mit bestimmten graphischen Komponenten soll der Benutzer interagieren können, um bestimmte Aktionen zu starten, z.B.
  - beim Drücken eines Buttons
  - nach der Eingabe eines Wertes
  - durch die Bewegung des Mauszeigers
- Es wird auch davon gesprochen, dass solche Interkationen bestimmte Events auslösen, die zu bestimmten Reaktionen führen.
- Es muss jedoch explizit programmiert werden bei welchen Events welche Reaktionen ausgeführt werden.

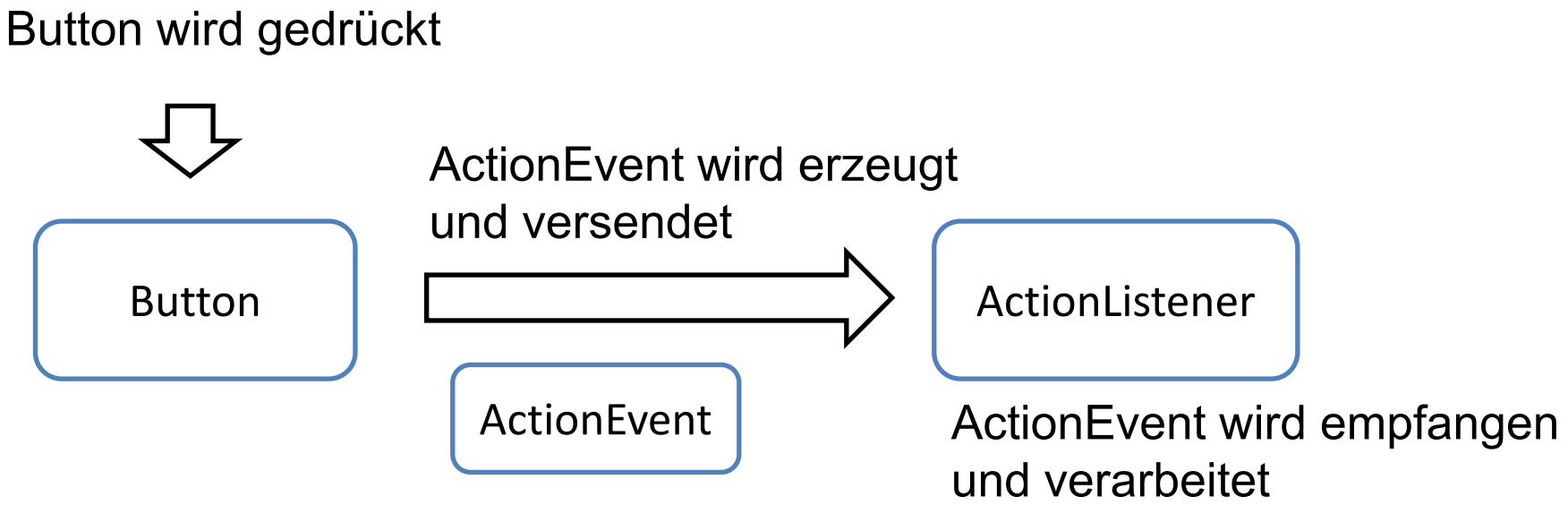
Event-Handling

Ein **Event** ist der Träger von Informationen über die Zustandsänderung eines Objekts (hier der Benutzeroberfläche). Ändert ein Objekt seinen Zustand, kann es ein Ereignis erzeugen. Das Ereignis beinhaltet die Zustandsänderung des Objektes. Das Objekt wird als Quelle des Ereignisses bezeichnet.

Ein **Listener** kann bestimmte Events empfangen und verarbeiten (siehe Observer-Muster). Hierzu muss ein Listener-Objekt bei einem Sender von Events registriert werden. Die Verarbeitung des Events kann durch den Programmierer implementiert werden.

Event-Handling

- Events und Listener für graphische Benutzeroberflächen sind Objekte bestimmter Klassen bzw. Schnittstellen.
- Für die unterschiedlichen graphische Komponenten existieren verschiedene spezialisierte Klassen bzw. Schnittstellen.



Klasse ActionEvent

- Events der Klasse `ActionEvent` werden z.B. beim Drücken eines Buttons automatisch erzeugt.
- Ein `ActionEvent`-Objekt besitzt verschiedene Informationen, z.B. über den Aufrufer, den Zeitpunkt der Aktivierung oder auch eine Bezeichnung der ausgeführten Aktion:
  - `Object getSource()`
  - `long getWhen()`
  - `String getActionCommand()`

Klasse ActionListener

- Auf Events kann ohne ein angemeldetes `Listener`-Objekt nicht reagiert werden.
- Zur Implementierung von Listener-Klassen werden verschiedene Schnittstellen zur Verfügung gestellt.
- In diesen Schnittstellen wird definiert, welche Methoden beim Empfang eines Events automatisch aufgerufen werden.
- Die Schnittstelle `ActionListener` stellt eine Methode zum Empfang von `ActionEvent`-Objekten zur Verfügung:
  - `void actionPerformed(ActionEvent e)`

Beispiel Funktionsplotter

Beim Drücken des Buttons, soll die Funktion gezeichnet werden. Hierzu müssen verschiedene Methoden des Darstellungspanels (`plotPanel` der Klasse `Function1DPanel`) aufgerufen werden:

- `void setFunction(String f, String x)`
- `void setInterval(double a, double b)`
- `void draw()`

Die einzelnen Werte der drei Textfelder müssen zuvor abgefragt, umgewandelt und übergeben werden.

Das ActionListener-Objekt muss somit auf viele Komponenten des Fensters (`Function1DFrame`-Objekt) zu greifen.

Implementierung ActionListener V1

Sehr aufwendig wird die Implementierung, falls eine eigene Klasse für den `ActionListener` erstellt wird.

- Die Beziehungen zu allen Objekten (Komponenten), die im Rahmen der Methode `actionPerformed` verwendet werden sollen, müssen gespeichert werden.

Implementierung ActionListener V1

```
public class ActionListenerPlotButton implements ActionListener {

  // Beziehungen zu Komponenten des FunctionPlotFrame's
  private JTextField fTf, aTf, bTf;
  private FunctionPlotPanel plotPanel;

  // Konstruktor: Beziehungen werden aufgebaut
  public ActionListenerPlotButton(JTextField fTf,
     JTextField aTf, JTextField bTf, FunctionPlotPanel plotPanel) {

     this.fTf = fTf;
     this.aTf = aTf;
     this.bTf = bTf;

     this.plotPanel = plotPanel;
  }
}
```

Implementierung ActionListener V1

```
public class ActionListenerPlotButton implements ActionListener {

  // ...
  // Diese Methode wird aufgerufen, wenn ein Event aufgetreten ist
  public void actionPerformed(ActionEvent e) {

     this.plotPanel.setFunction(this.fTf.getText(), "x");

     double a = Double.parseDouble(this.aTf.getText());
     double b = Double.parseDouble(this.bTf.getText());

     this.plotPanel.setInterval(a, b);
     this.plotPanel.draw();

  }
}
```

Benutzeroberflächen

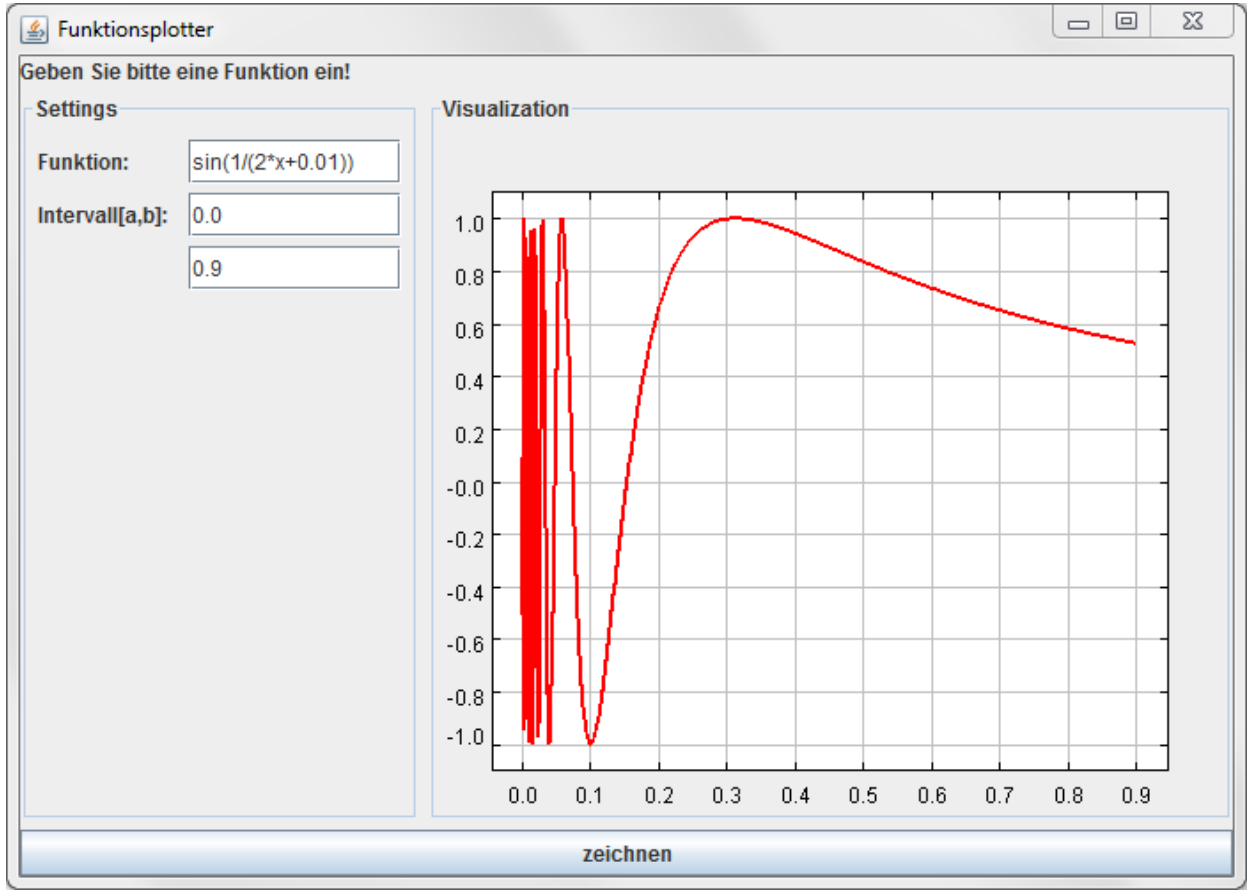
ActionListener anmelden

Im nächsten Schritt wird ein `ActionListenerPlotButton`-Objekt erzeugt und beim entsprechenden Button angemeldet. Die Methode `actionPerformed` dieses `Listener`-Objektes wird jetzt immer aufgerufen, wenn der Button gedrückt wird.

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // ..

    // Button zum Zeichnen der Funktion erzeugen und hinzufügen
    this.plotButton = new JButton("zeichnen");

    // Listener erzeugen und anmelden
    this.plotButton.addActionListener(new ActionListenerPlotButton(
          this.fTf, this.aTf, this.bTf, this.plotPanel));

    this.getContentPane().add(this.plotButton, BorderLayout.SOUTH);
}
```

Benutzeroberflächen

Ergebnis



Implementierung ActionListener V2

Sind mehrere Buttons vorhanden oder muss auf sehr viele Komponenten innerhalb des Listeners zugegriffen werden, kann auch eine Komponente als Listener umgesetzt werden

- Häufig implementieren eigene Klassen für Fenster oder Panels zusätzlich bestimmte Listener-Schnittstellen.
- Es müssen weniger Klassen umgesetzt werden und bestimmte Aktionen können gebündelt ausgeführt werden.
- Jedoch muss dann genau geprüft werden, welche Komponenten ein Event erzeugt haben.

Benutzeroberflächen

Implementierung ActionListener V2

```
public class FunctionPlotFrame extends JFrame
                                    implements ActionListener{
  // Methode wird bei einem ActionEvent aufgerufen
  public void actionPerformed(ActionEvent e) {
     // Wenn des der plotButton ist, dann diese Anweisungen ausfürhen
     if (e.getSource() == this.plotButton) {

        this.plotPanel.setFunction(this.fTf.getText(), "x");
        double a = Double.parseDouble(this.aTf.getText());
        double b = Double.parseDouble(this.bTf.getText());

        this.plotPanel.setInterval(a, b);
        this.plotPanel.draw();

     }
  }
}
```

Benutzeroberflächen

ActionListener anmelden

Jetzt muss das `FunctionPlotFrame`-Objekt beim Button angemeldet werden, da es gleichzeitig auch ein `ActionListener` ist.

```
public class FunctionPlotFrame extends JFrame {
  private void init() {
    // ..

    // Button zum Zeichnen der Funktion erzeugen und hinzufügen
    this.plotButton = new JButton("zeichnen");

    // Listener erzeugen und anmelden
    this.plotButton.addActionListener(this);

    this.getContentPane().add(this.plotButton, BorderLayout.SOUTH);
}
```

RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

Sommersemester 2023

Architekturmuster

Architekturmuster

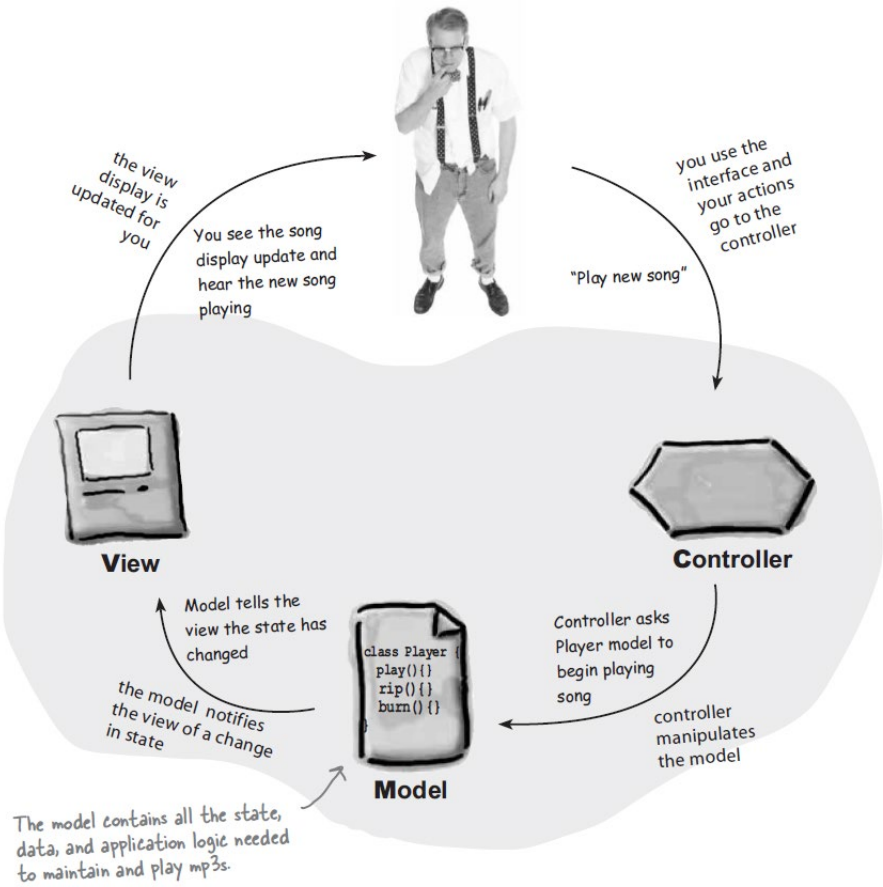
MP3-Player

Es soll ein MP3-Player implementiert werden. Über einer Benutzeroberfläche können neue Songs hinzugefügt, Playlists verwaltet und Titel umbenannt werden. In einer kleinen „Datenbank“ sollen alle Songs und dazugehörigen Namen und Informationen gespeichert werden. Beim Abspielen von Songs und Playlists soll die Benutzeroberfläche laufend mit dem jeweiligen Songtitel, der Spieldauer usw. aktualisiert werden.

Architekturmuster

Model-View-Controller

Komplette Anwendungen (inkl. Ausgabe, Datenhaltung, Steuerung, etc.) werden häufig nach dem Model-View-Controller Prinzip gestaltet.



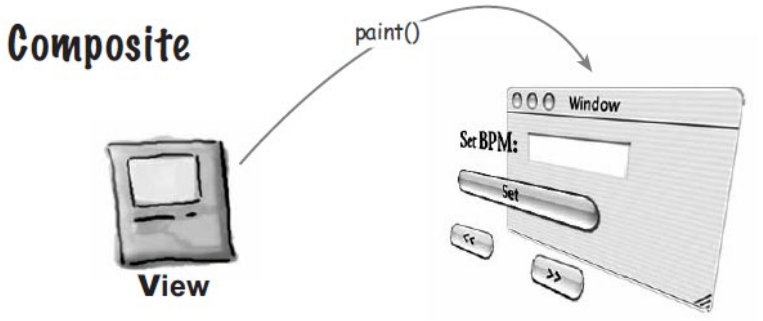
Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

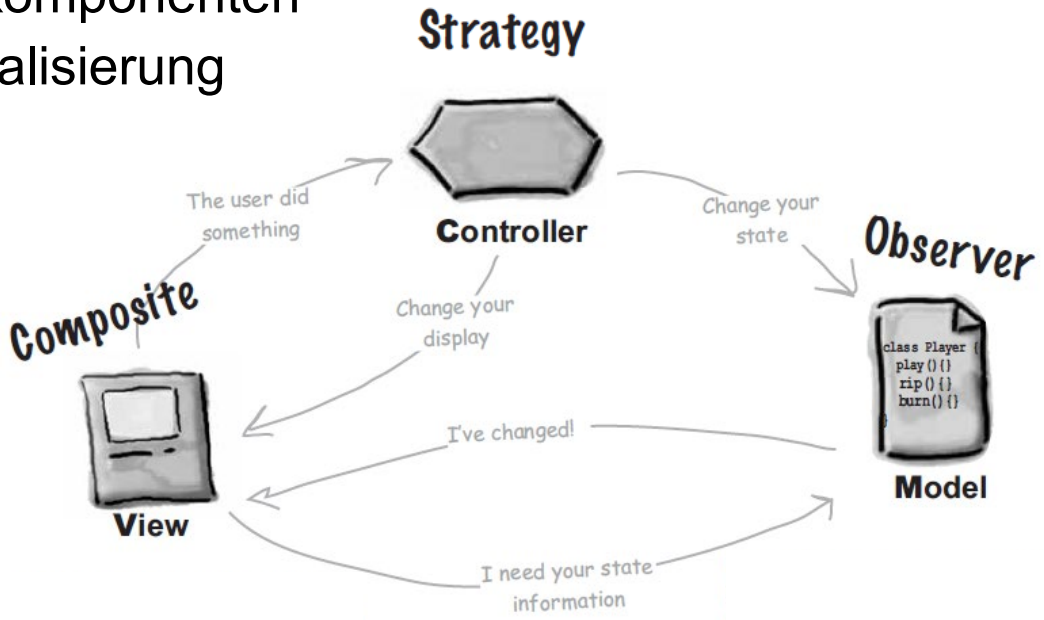
Architekturmuster

Model-View-Controller

Das Model-View-Controller Prinzip verwendet eigentlich drei unterschiedliche Entwurfsmuster: Strategie, Observer, und Composite

Ein View wird aus verschiedenen Komponenten zusammengebaut. Wenn eine Aktualisierung erfolgen soll, muss nur die oberste Komponente informiert werden



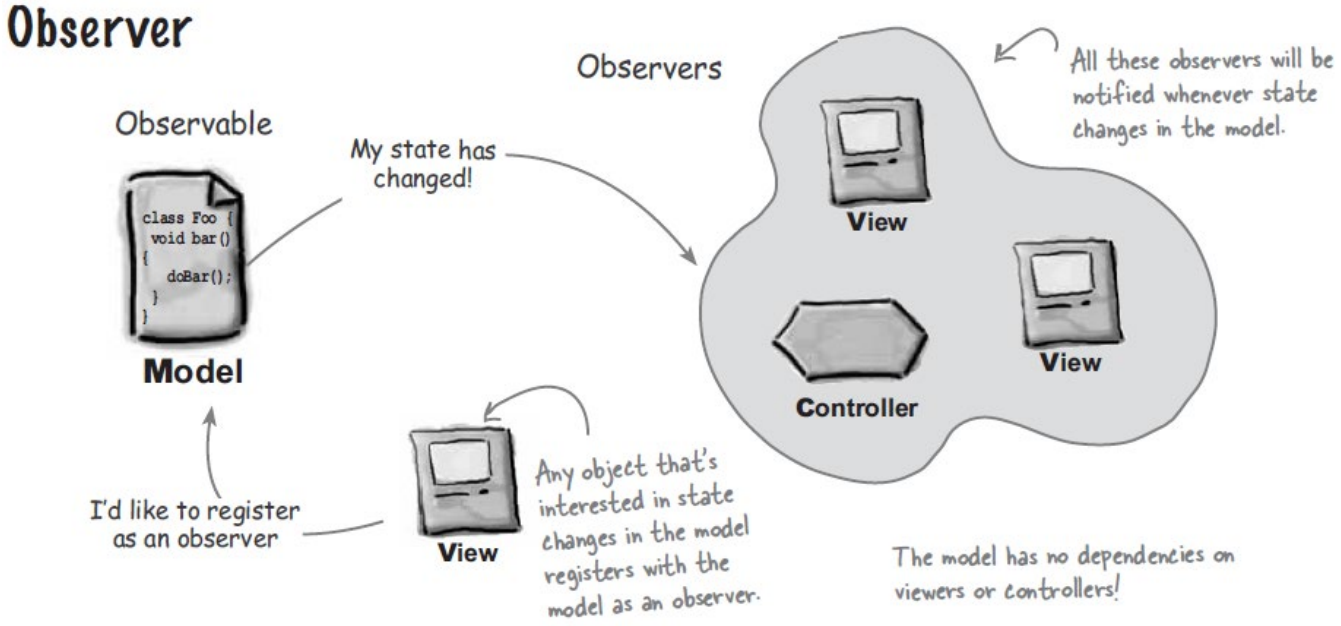


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Architekturmuster

Model-View-Controller

Das Model implementiert das Observer-Muster, um interessierte Objekte bei Änderungen auf dem neusten Stand zu halten. Das Modell ist dann völlig unabhängig vom View und dem Controller

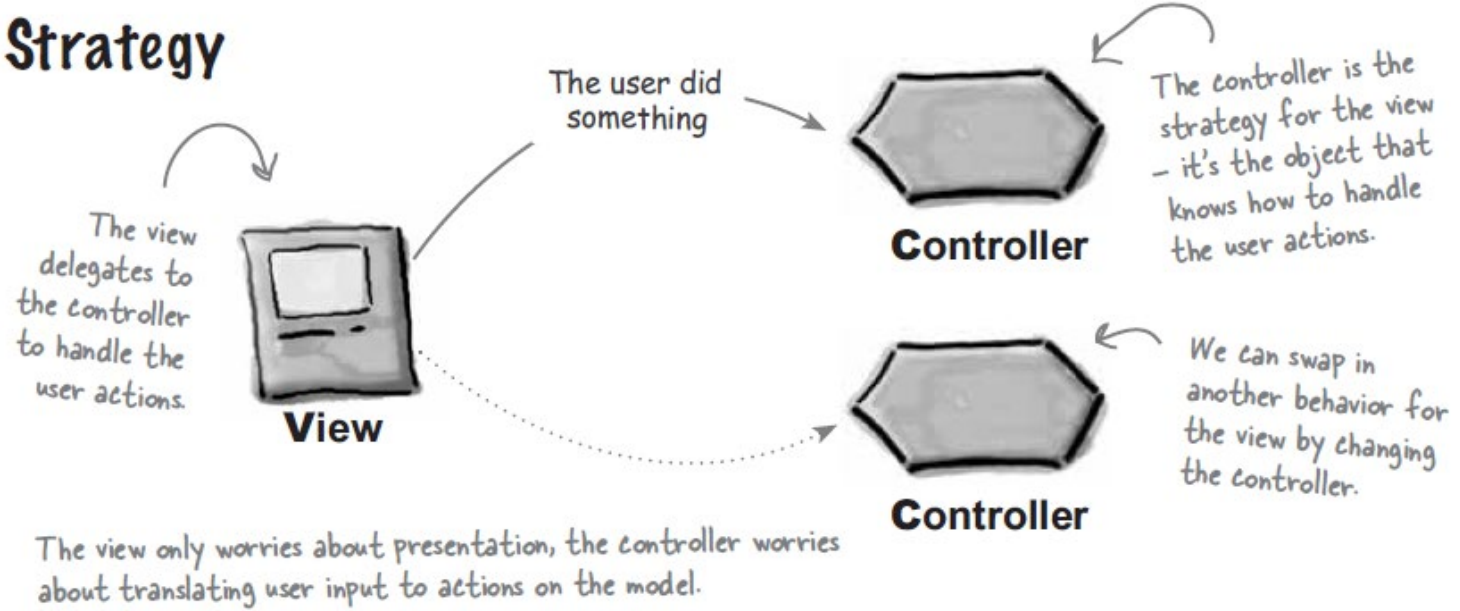


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Architekturmuster

Model-View-Controller

Der View verwendet den Controller zur Änderung des Zustandes. Der Controller liefert das entsprechende Verhalten. Somit wird der View vollständig vom Model getrennt, da nur der Controller für die Interaktionen mit dem Model zuständig ist.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Architekturmuster

MP3-Player mit MVC-Muster