Grafische Benutzeroberflächen

Dr. Annabelle Klarl

Zentralübung zur Vorlesung
„Einführung in die Informatik: Programmierung und Softwareentwicklung“

`http://www.pst.ifi.lmu.de/Lehre/wise-16-17/infoeinf`

WS16/17

Action required now

socrative

1. Smartphone: installiere die App "socrative student" **oder**
   Laptop: öffne im Browser b.socrative.com/login/student

2. Betrete den Raum **InfoEinf**.

3. Beantworte die **erste Frage** sofort!

Aufgabe

Ein Benutzer soll zwei Zahlen in ein Programm eingeben können. Das Programm kann entweder die Summe oder das Produkt der beiden Zahlen berechnen und ausgeben.

Schreiben Sie ein Programm, das möglichst benutzerfreundlich ist, d.h.

- Benutzereingaben geeignet abfragt,
- eine interaktive Auswahl für Summen- oder Produktberechnung bereitstellt,
- das Ergebnis "schön" darstellt.

Programme ohne GUI (**G**raphical **U**ser **I**nterface)

- **Benutzereingaben** (wie zwei Zahlen)
  - direkt im Code ☹
  - beim Aufruf des Programms zu übergeben (in Vorlesung nicht behandelt)
- **Interaktive Auswahl**: nicht möglich ☹
- **Ausgabe des Ergebnisses** (wie Summe/Produkt): per Kommandozeile ☹

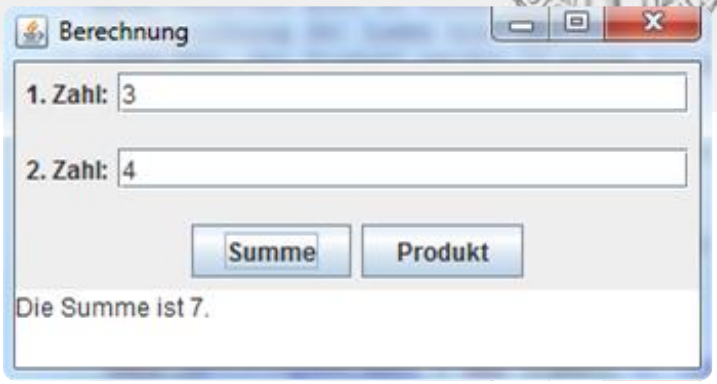
```
public class Berechnung {
  public static void main(String[] args) {
     int a = 2;
     int b = 3;
     int summe = a+b;
     System.out.println("Summe: " + summe);
}}
```

Programme mit grafischer Benutzereingabe

- **Benutzereingaben** (wie zwei Zahlen): mit Hilfe von `JOptionPane`, aber für jede Eingabe ein neues Eingabe-Fenster 😐
- **Interaktive Auswahl**: mit Hilfe von `JOptionPane` 😐
- **Ausgabe des Ergebnisses** (wie Summe/Produkt): per Kommandozeile ☹ oder mit `JOptionPane` (nicht behandelt)

```
public class Berechnung {
  public static void main(String[] args) {
     String a = JOptionPane.showInputDialog("1. Zahl:");
     String b = JOptionPane.showInputDialog("2. Zahl:");
     String berechnung =
       JOptionPane.showInputDialog("Summe oder Produkt:");
    ...
}}
```

Programme mit GUI (**G**raphical **U**ser **I**nterface)



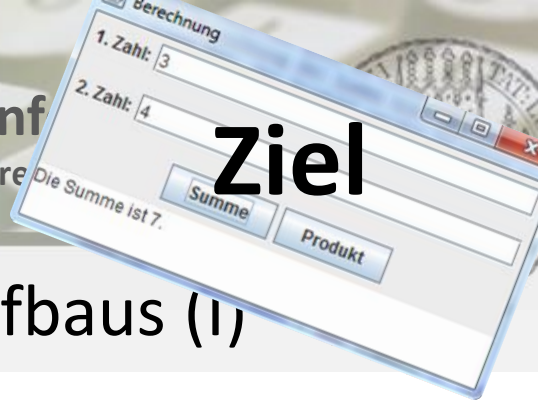
Programm hat eine grafischen Oberfläche
z.B. ein Fenster

- **Benutzereingaben** (wie zwei Zahlen):
  im Eingabebereich des Fensters z.B. durch Textfelder ☺
- **Interaktive Auswahl**:
  im Eingabebereich des Fensters z.B. durch Buttons ☺
- **Ausgabe des Ergebnis** (wie Summe/Produkt der beiden Zahlen):
  im Ausgabebereich des Fenster z.B. durch Textfeld ☺

=> Funktion mehrfach verwendbar ohne Programmneustart ☺

Aufgabe: Vorgehensweise zur Erstellung der GUI

1. **Erstellung des strukturellen Aufbaus der GUI**:
   a. Erstellung des Basis-Fensters
   b. Attribute für Eingabefelder, Buttons und Ausgabebereich
   c. Initialisierung der Eingabefelder, Buttons und Ausgabebereichs
   d. Platzierung der Eingabefelder, Buttons und Ausgabebereichs auf dem Basis-Fenster
2. **Verbindung der GUI mit inhaltlichen Objekten der Anwendung**: hier nicht nötig (nur zwei Zahlen)
3. **Ereignisgesteuerte Behandlung von Benutzereingaben**:
   a. Berechnung der Summe nach Knopfdruck des Buttons für Summe
   b. Berechnung des Produkts nach Knopfdruck des Buttons für Produkt
   c. Ordnungsgemäßes Schließen des Programms

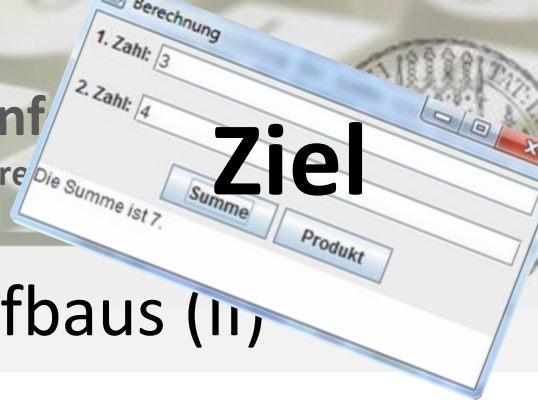


Aufgabe: 1. Erstellung des strukturellen Aufbaus (I)

1a. Erstellung des Basis-Fensters:

- Anlegen einer neuen Klasse `BerechnungsFrame`
- Erben von `javax.swing.JFrame`
- Setzen von Titel und Größe des Basis-Fensters

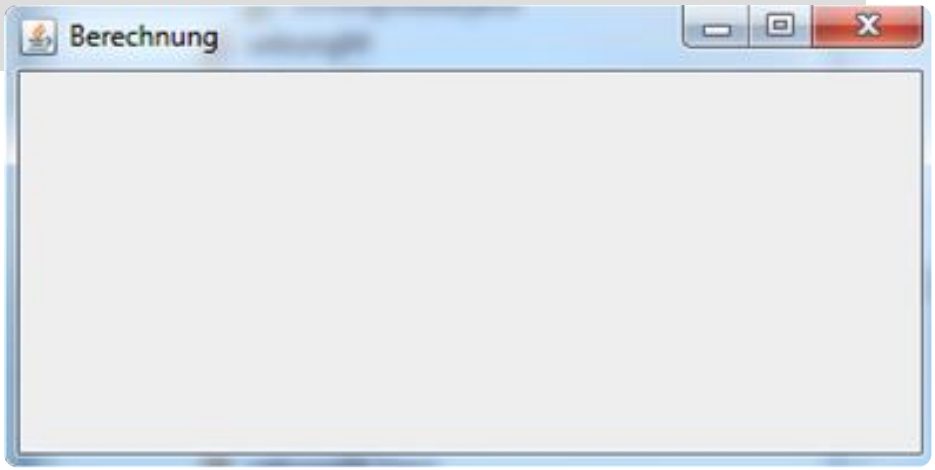
```
import javax.swing.*;

public class BerechnungsFrame extends JFrame {
  public BerechnungsFrame() {
     this.setTitle("Berechnung");
     this.setSize(350,150);
  }
}
```

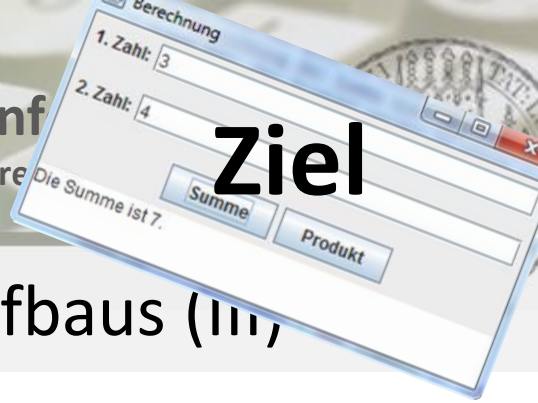


Aufgabe: 1. Erstellung des strukturellen Aufbaus (II)

Einschub: Ausführen des Programms

```
public class BerechnungsFrameMain {
  public static void main(String[] args) {
     BerechnungsFrame berechnung = new BerechnungsFrame();
     berechnung.setVisible(true);
  }
}
```

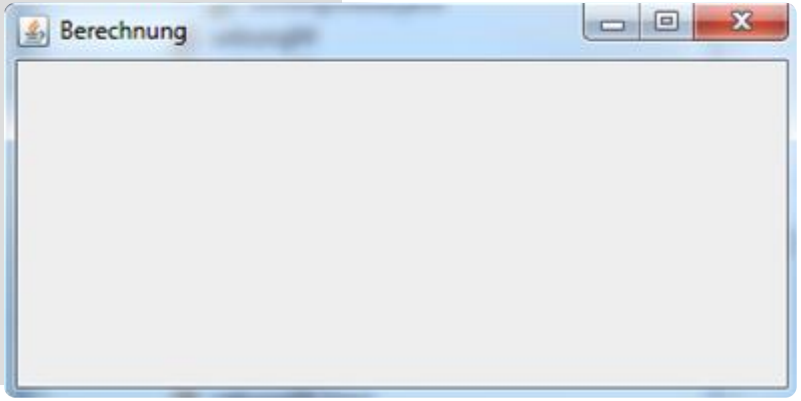


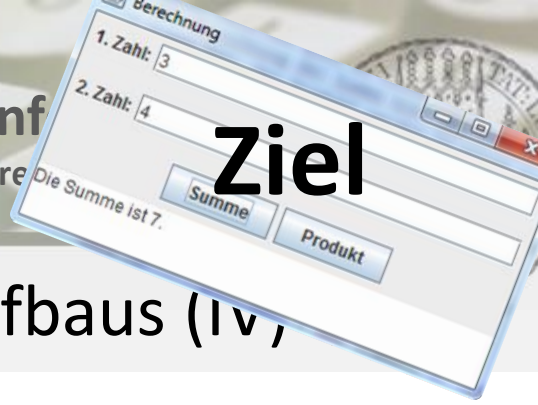


Aufgabe: 1. Erstellung des strukturellen Aufbaus (II)

1b. Attribute für Eingabefelder, Buttons und Ausgabebereich

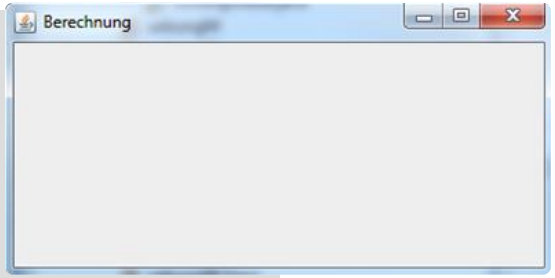
```
import javax.swing.*;
public class BerechnungsFrame extends JFrame {
  private JLabel zahl1EingabeLabel;
  private JTextField zahl1EingabeFeld;
  private JLabel zahl2EingabeLabel;
  private JTextField zahl2EingabeFeld;
  private JButton summeButton;
  private JButton produktButton;
  private JTextArea ausgabeBereich;

  public BerechnungsFrame() {
     this.setTitle("Berechnung");
     this.setSize(350,150);
}}
```



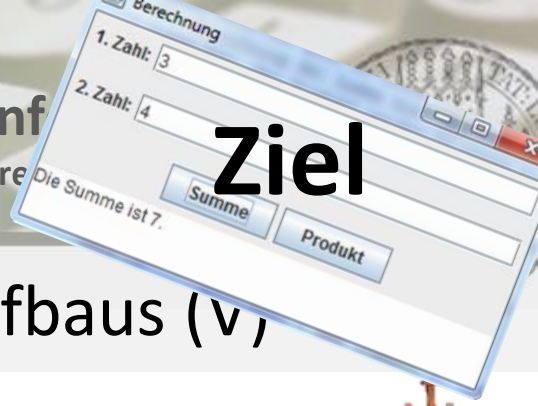


Aufgabe: 1. Erstellung des strukturellen Aufbaus (IV)

1c. Initialisierung der Eingabefelder, Buttons und Ausgabebereichs




```
import javax.swing.*;
public class BerechnungsFrame extends JFrame
  ... // Attribute wie vorher
  public BerechnungsFrame() {
     this.setTitle("Berechnung");
     this.setSize(350,150);
     this.zahl1EingabeLabel = new JLabel("1. Zahl:");
     this.zahl1EingabeFeld = new JTextField(25);
     this.zahl2EingabeLabel = new JLabel("2. Zahl:");
     this.zahl2EingabeFeld = new JTextField(25);
     this.summeButton = new JButton("Summe");
     this.produktButton = new JButton("Produkt");
     this.ausgabeBereich = new JTextArea(3, 33);
}}
```



Aufgabe: 1. Erstellung des strukturellen Aufbaus (v)

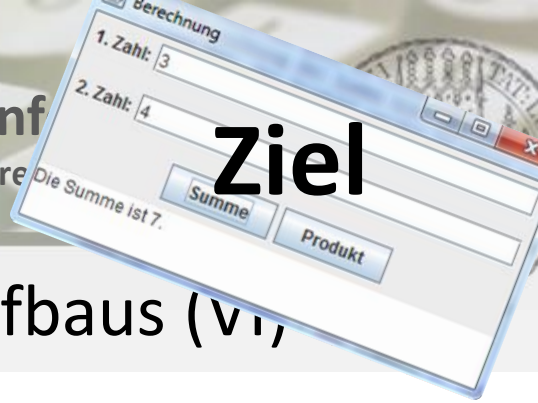
1d. Platzierung auf dem Basis-Fenster

- Staffelei: Basis-Fenster `JFrame` (bzw. `BerechnungsFrame`)

- Leinwand auf der Staffelei: `ContentPane` des Basis-Fensters
  - Zugriff innerhalb des Basis-Fensters:

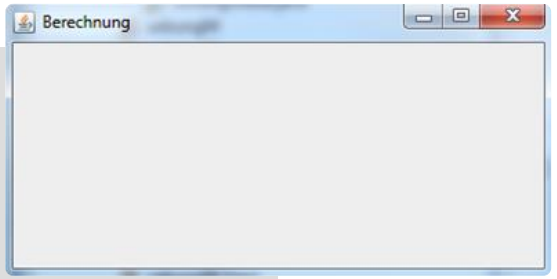
```
Container contentPane = this.getContentPane();
```

- Bild auf der Leinwand: Interaktionselemente wie Buttons

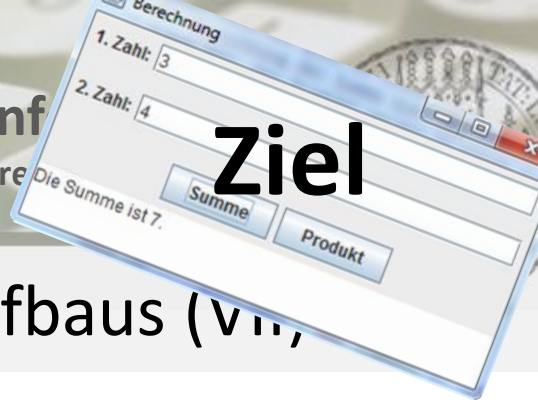


Aufgabe: 1. Erstellung des strukturellen Aufbaus (v1)

1d. Platzierung auf dem Basis-Fenster




```
import javax.swing.*;
public class BerechnungsFrame extends JFrame
  ... // Attribute wie vorher
  public BerechnungsFrame() {
     ... // Initialisierung wie vorher
     Container contentPane = this.getContentPane();
     contentPane.add(this.zahl1EingabeLabel);
     contentPane.add(this.zahl1EingabeFeld);
     contentPane.add(this.zahl2EingabeLabel);
     contentPane.add(this.zahl2EingabeFeld);
     contentPane.add(this.summeButton);
     contentPane.add(this.produktButton);
     contentPane.add(this.ausgabeBereich);
}}
```

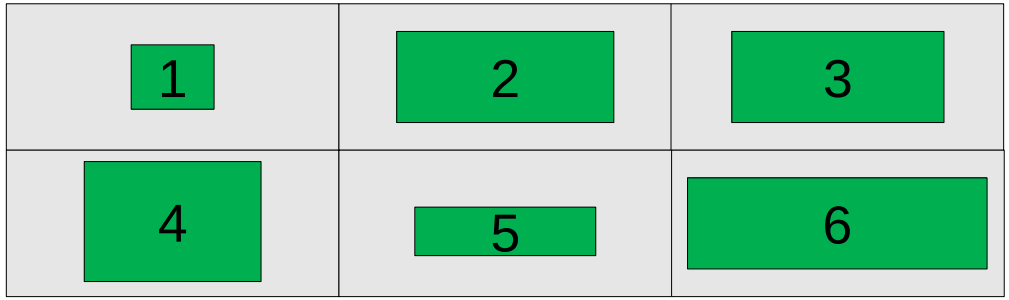


Aufgabe: 1. Erstellung des strukturellen Aufbaus (vi

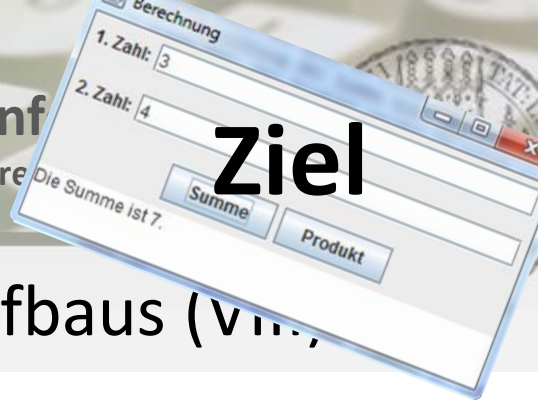
Einschub: `LayoutManager`

- `GridLayout`:
  - Es kann angegeben werden, **wie viele Zeilen und Spalten** das Fenster haben soll.
  - Jedes Interaktionselement wird der Reihe nach in eine der so entstandenen Zellen gesetzt.

=> Jede der Zelle ist so groß wie das **breiteste und höchste** Interaktionselement



`GridLayout` mit zwei Zeilen und drei Spalten

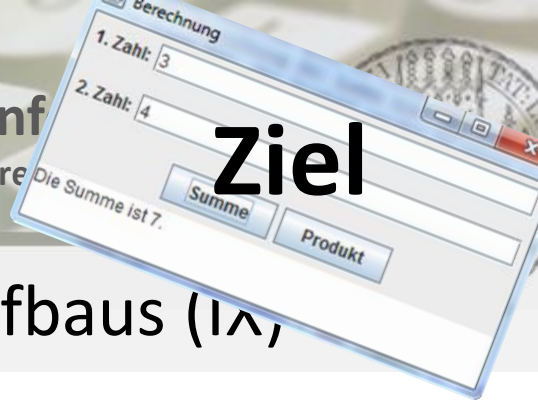


Aufgabe: 1. Erstellung des strukturellen Aufbaus (vi

Einschub: `LayoutManager`

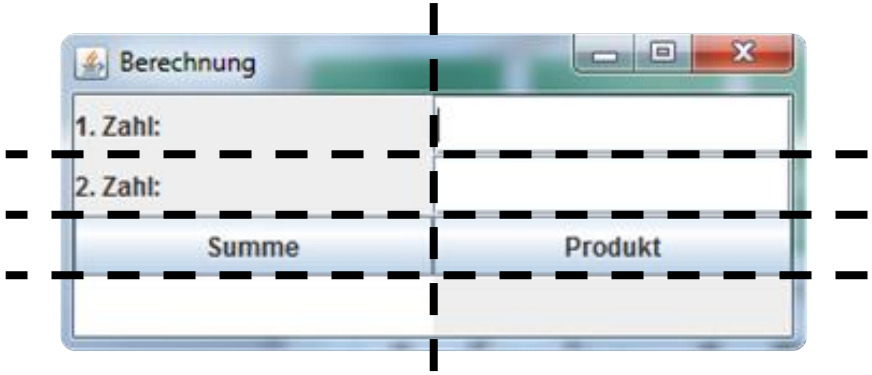
- `GridLayout`:
  - Implementierung

```
import javax.swing.*;
public class BerechnungsFrame extends JFrame {
  ... // Attribute wie vorher
  public BerechnungsFrame() {
     ... // Initialisierung wie vorher
     Container contentPane = this.getContentPane();
     contentPane.setLayout(new GridLayout(4,2))
     contentPane.add(this.zahl1EingabeLabel);
     ... // Hinzufügen wie vorher
}}
```



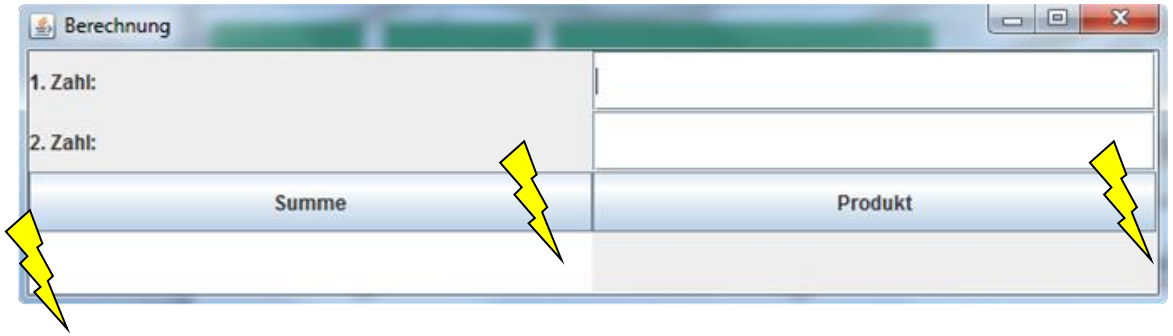
Aufgabe: 1. Erstellung des strukturellen Aufbaus (IX)

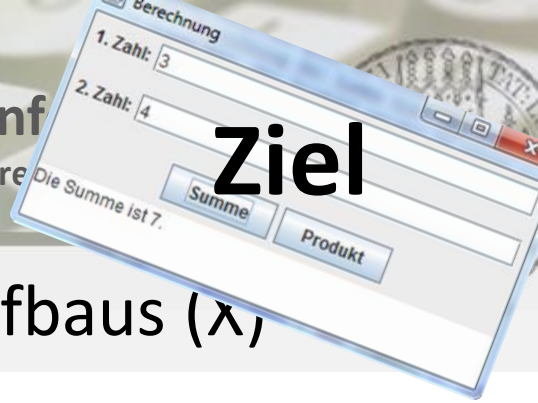
Einschub: Ausführen des Programms



GridLayout
(mit 4 Zeilen und 2 Spalten)

größer ziehen



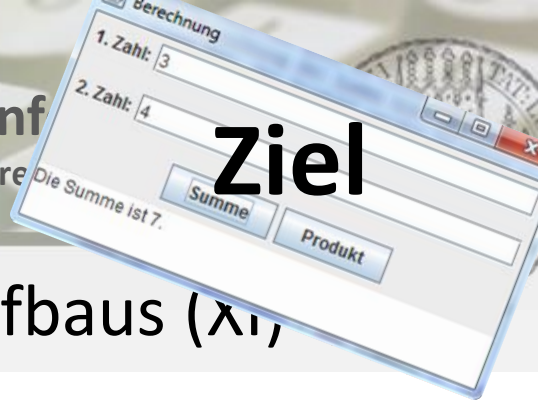


Aufgabe: 1. Erstellung des strukturellen Aufbaus (x)

Einschub: Gruppierung von Interaktionselementen

- Interaktionselemente können in einem `JPanel` gruppiert werden.
- Dieses `JPanel` kann wie alle anderen Interaktionselemente platziert werden.

```
import javax.swing.*;
public class BerechnungsFrame extends JFrame {
  ... // Attribute wie vorher
  public BerechnungsFrame() {
     ... // Initialisierung wie vorher
     JPanel eingabePanel1 = new JPanel();
     eingabePanel1.add(this.zahl1EingabeLabel);
     eingabePanel1.add(this.zahl1EingabeFeld);
     ... // analog für 2. Zahl und Buttons
// -> Fortsetzung nächste Folie
```

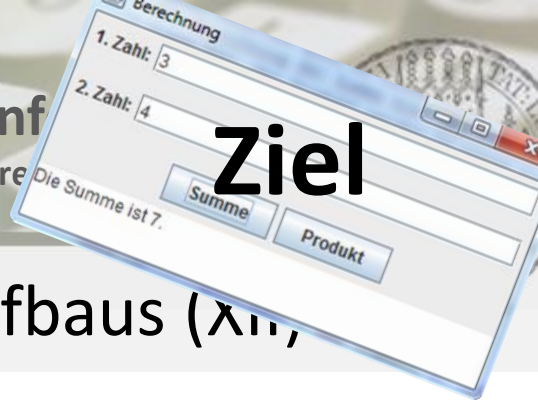


Aufgabe: 1. Erstellung des strukturellen Aufbaus (XI)

Einschub: Gruppierung von Interaktionselementen

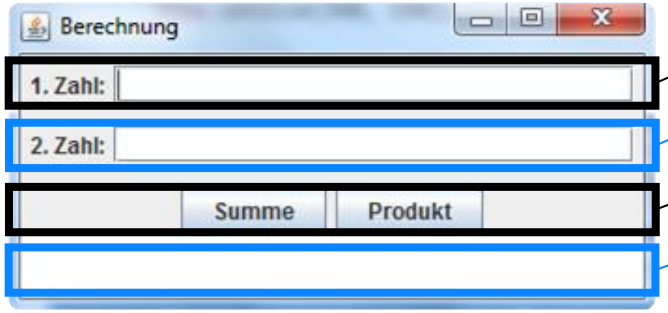
- Interaktionselemente können in einem `JPanel` gruppiert werden.
- Dieses `JPanel` kann wie alle anderen Interaktionselemente platziert werden.

```
// -> Fortsetzung von vorheriger Folie
    Container contentPane = this.getContentPane();
    contentPane.setLayout(new GridLayout(4,1))
    contentPane.add(eingabePanel1);
    contentPane.add(eingabePanel2);
    contentPane.add(buttonPanel);
    contentPane.add(this.ausgabeBereich);
  }
}
```



Aufgabe: 1. Erstellung des strukturellen Aufbaus (XML)

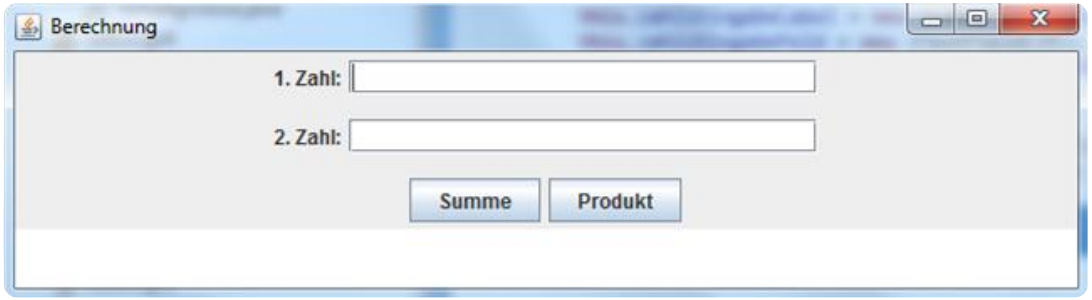
Einschub: Ausführen des Programms



eingabePanel1
eingabePanel2
buttonPanel
ausgabeBereich

GridLayout
(mit 4 Zeilen und 1 Spalte und Gruppierungen)

größer ziehen



Aufgabe: 2. Verbindung der GUI mit inhaltlichen Objekten

Es werden lediglich zwei Zahlen eingelesen und addiert oder multipliziert

=> Keine Speicherung nötig!

Aufgabe: 3. Behandlung von Benutzereingaben (I)

3a. Berechnung der Summe nach Knopfdruck des Summe-Buttons

Ein `ActionListener` muss beim Summe-Button **registriert** werden, der auf den Knopfdruck wartet und entsprechend reagiert

- `BerechnungsFrame` implementiert `ActionListener` (d.h. er wird zu einem `ActionListener`)
- Der `BerechnungsFrame` wird beim Summe-Button registriert

```
import javax.swing.*;
public class BerechnungsFrame
  extends JFrame implements ActionListener {
     ... // Attribute wie vorher
     public BerechnungsFrame() {
        ... // Initialisierung und Platzierung wie vorher
       this.summeButton.addActionListener(this);
}}
```

Aufgabe: 3. Behandlung von Benutzereingaben (II)

3a. Berechnung der Summe nach Knopfdruck des Summe-Buttons

Ein `ActionListener` muss beim Summe-Button registriert werden, der auf den Knopfdruck **wartet** und entsprechend reagiert

- `BerechnungsFrame` wartet in der Methode `actionPerformed` auf den Knopfdruck des Summe-Buttons

```
import javax.swing.*;
public class BerechnungsFrame
  extends JFrame implements ActionListener {
     ... // Attribute und Konstruktor wie vorher
     public void actionPerformed(ActionEvent e) {
        if (e.getSource() == this.summeButton) {
           this.summiere();
        }
}}
```

Aufgabe: 3. Behandlung von Benutzereingaben (III)

3a. Berechnung der Summe nach Knopfdruck des Summe-Buttons

Ein `ActionListener` muss beim Summe-Button registriert werden, der auf den Knopfdruck wartet und entsprechend **reagiert**

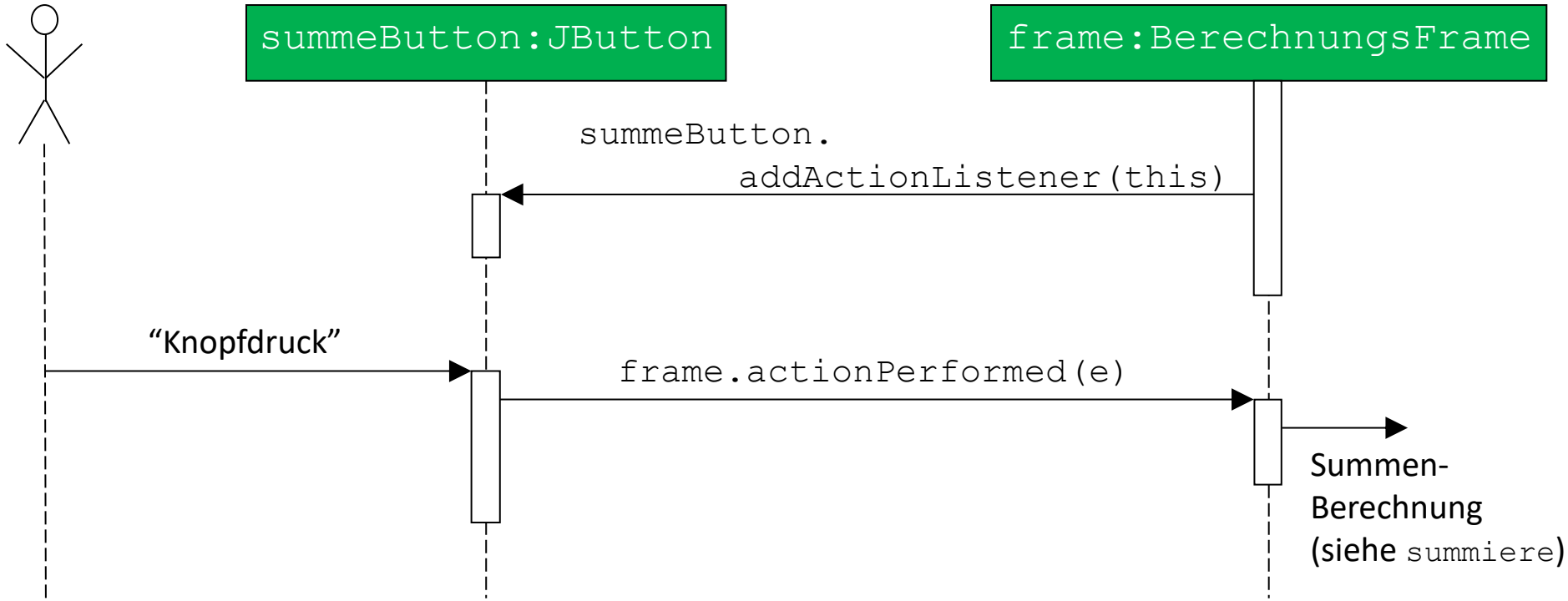
- `BerechnungsFrame` reagiert in der Methode `summiere` entsprechend

```
import javax.swing.*;
public class BerechnungsFrame extends … implements … {
  ... // Attribute, Konstruktor, actionPerformed wie vorher
  private void summiere() {
     int zahl1 =
        Integer.parseInt(this.zahl1EingabeFeld.getText());
     int zahl2 =
        Integer.parseInt(this.zahl2EingabeFeld.getText());
     this.ausgabeBereich.setText("Summe: "+(zahl1+zahl2));
}}
```

Aufgabe: 3. Behandlung von Benutzereingaben (IV)

3a. Berechnung der Summe nach Knopfdruck des Summe-Buttons

Was passiert eigentlich?



Aufgabe: 3. Behandlung von Benutzereingaben (V)

3b. Berechnung des Produkts nach Knopfdruck des Prod.-Buttons

Ein `ActionListener` muss beim Produkt-Button **registriert** werden, der auf den Knopfdruck **wartet** und entsprechend **reagiert**

- Der `BerechnungsFrame` wird beim Produkt-Button registriert
- `BerechnungsFrame` wartet in der Methode `actionPerformed` auf den Knopfdruck des Produkt-Buttons
- `BerechnungsFrame` reagiert in der Methode `multipliziere` entsprechend

```
import javax.swing.*;
public class BerechnungsFrame extends … implements … {
  ... // Implementierung analog zu vorher erweitern
}
```

Aufgabe: 3. Behandlung von Benutzereingaben (VI)

3c. Ordnungsgemäßes Schließen des Programms

Das Programm muss beim Drücken des X-Buttons (rechts-oben) sowohl das Fenster als auch das Programm selbst beenden.

- Fenster wird automatisch beendet.
- Programm muss von Hand beendet werden.

```
import javax.swing.*;
public class BerechnungsFrame extends … implements … {
  ... // alles wie vorher mit Erweiterung des Konstruktors
  public BerechnungsFrame() {
     ... // wie vorher
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Aufgabe: gelöst ☺

Code: siehe Webseite