

Kapitel 7

Grafische Benutzeroberflächen

Ziele GUI

- Grafische Benutzeroberflächen (**G**raphical **U**ser **I**nterfaces) als Anwendungsbeispiel für die objektorientierte Programmierung kennenlernen.
- Erstellung individueller GUI-Klassen durch Erweiterung existierender Klassen der Java Bibliotheken AWT und Swing.
- Die Vorgehensweise zur Erstellung einer GUI verstehen und durchführen können:
 1. Erstellung des strukturellen Aufbaus der GUI,
 2. Verbindung der Ansicht (GUI) mit den inhaltlichen Objekten der Anwendung (Modell),
 3. Ereignisgesteuerte Behandlung von Benutzereingaben (z.B. Knopfdruck).

AWT und Swing

Plattform-
unabhängig

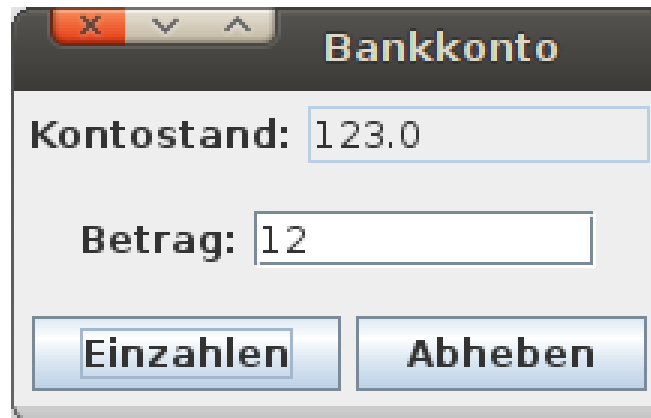
- AWT und Swing bieten eine Klassenbibliothek zur Programmierung grafischer Benutzerschnittstellen (GUIs) für Java-Programme.
- Java 1.0 wurde 1996 mit dem **Abstract Window Toolkit (AWT)** veröffentlicht.
- In Java 1.2 wurde 1998 eine verbesserte Bibliothek namens **Swing** eingeführt.
- Swing baut auf AWT auf (es werden Klassen aus dem AWT benutzt).
- Typische Import-Deklarationen in einem Programm, das AWT/Swing benutzt:

```
import java.awt.*;
```

```
import javax.swing.*;
```

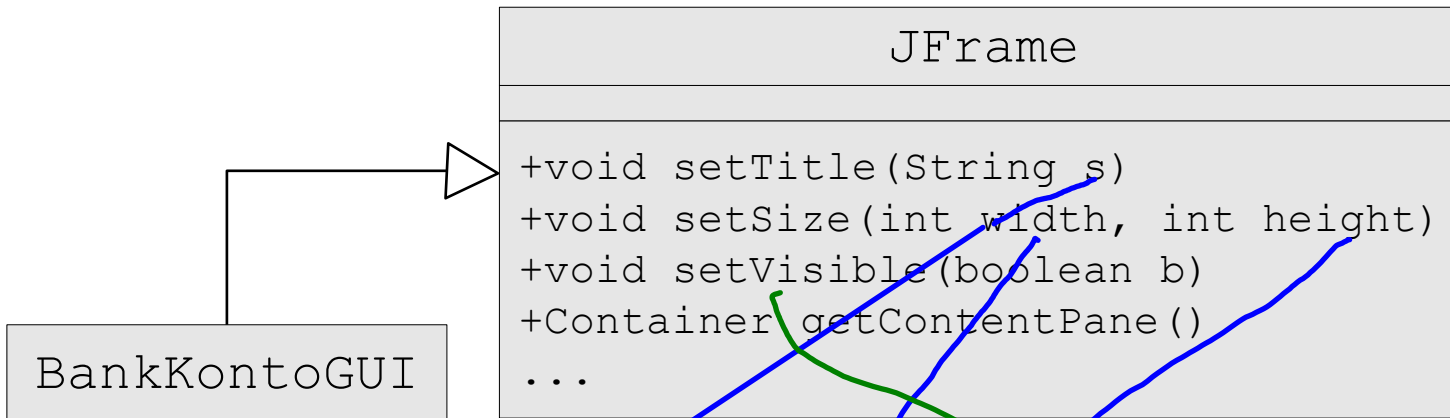
Beispiel

Wir wollen eine einfache grafische Anwendung für Bankkonten erstellen:



Fenster

Die Klasse `JFrame` stellt ein leeres Fenster zur Verfügung.



Wir können das Fenster durch Subklassenbildung spezialisieren und benutzen:

```

import java.awt.*;
import javax.swing.*;
public class BankKontoGUI extends JFrame {
    public BankKontoGUI() {
        this.setTitle("Bankkonto");
        this.setSize(300, 200);
        ...
    }
}
    
```

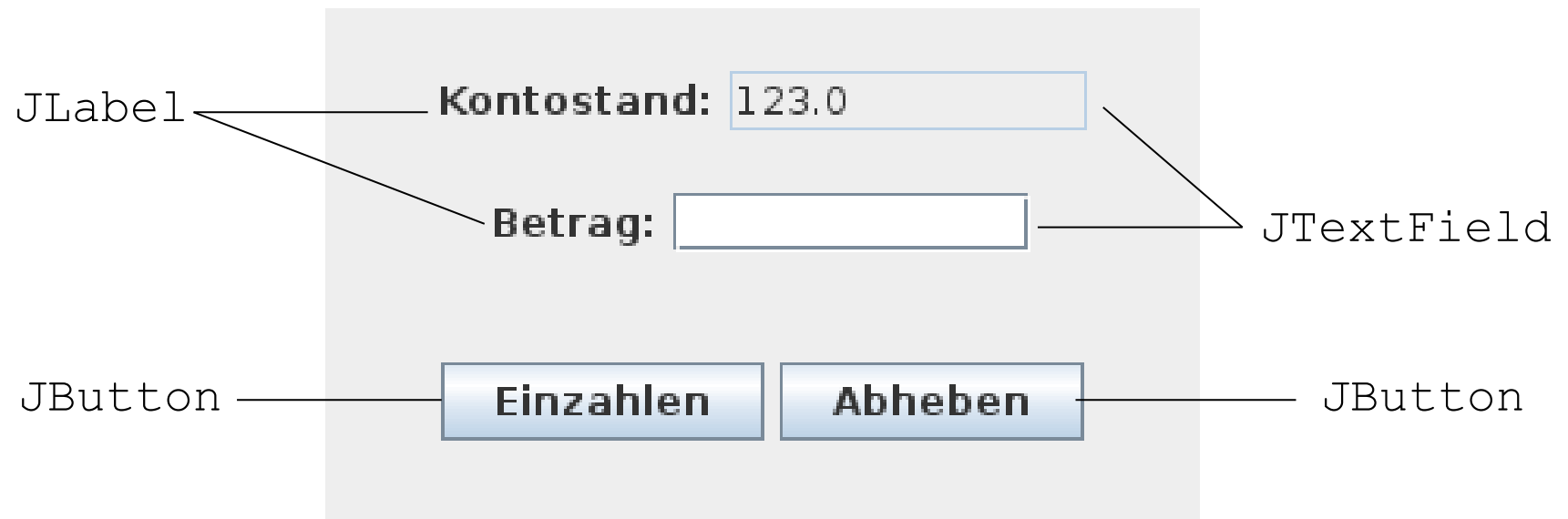
```

public class Main {
    public static void main(String[] args) {
        BankKontoGUI gui = new BankKontoGUI();
        gui.setVisible(true);
    }
}
    
```

GUI-Elemente

- In Swing gibt es viele Klassen für die verschiedenen Interaktionselemente.
- Textaufschriften, Knöpfe, Textfelder für Ein/Ausgabe usw. werden durch Objekte der Klassen `JLabel`, `JButton`, `JTextField` usw. repräsentiert.

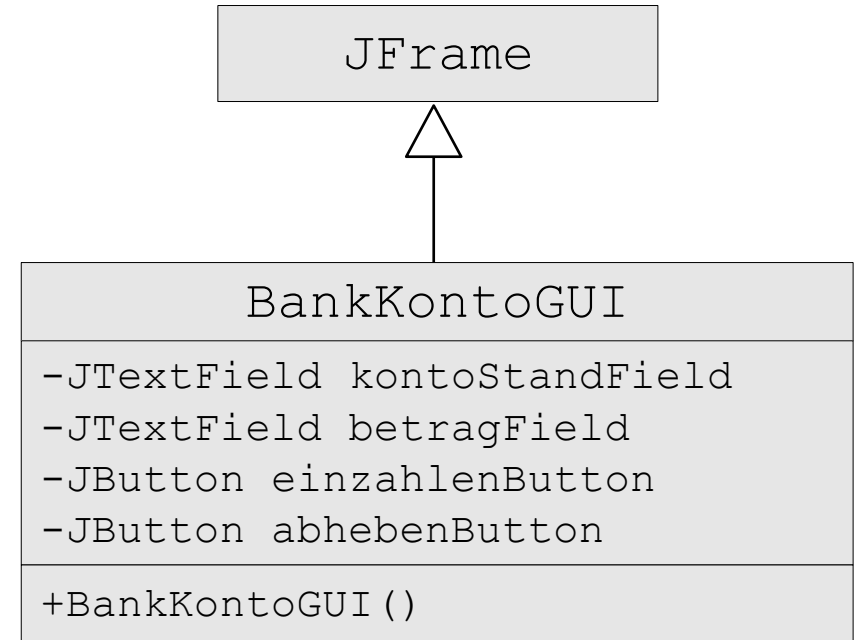
Beispiel:



Anlegen von Instanzvariablen (Attributen) für die GUI-Elemente

```
import java.awt.*;
import javax.swing.*;
public class BankKontoGUI extends JFrame {
    private JTextField kontoStandField;
    private JTextField betragField;
    private JButton einzahlenButton;
    private JButton abhebenButton;

    public BankKontoGUI() {
        this.setTitle("Bankkonto");
        this.setSize(300, 200);
        /* Initialisierung der Attribute */
        this.kontoStandField = new JTextField("123.0", 10);
        this.kontoStandField.setEditable(false);
        this.betragField = new JTextField(10);
        this.einzahlenButton = new JButton("Einzahlen");
        this.abhebenButton = new JButton("Abheben");
        ... }
}
```



Handwritten notes:
- 123.0 → *Feldlänge*
- *nicht ausführbar*
- *Kontoschrift der Kontof*

Strukturierung von Fenster-Inhalten

- Der Hintergrund eines JFrame-Fensters ist ein Container-Objekt; zugreifbar durch die Methode `getContentPane()`.
- Der Inhalt des Fensters ist darin als ein Baum von Objekten organisiert.
- Für die Gruppierung verwenden wir Objekte der Klasse `JPanel`.

Kontostand Panel

Container

JPanel

JPanel

JPanel

JLabel JTextField

JLabel

JTextField

JButton

JButton

Aufbau der Baumstruktur und Setzen des Layouts

```

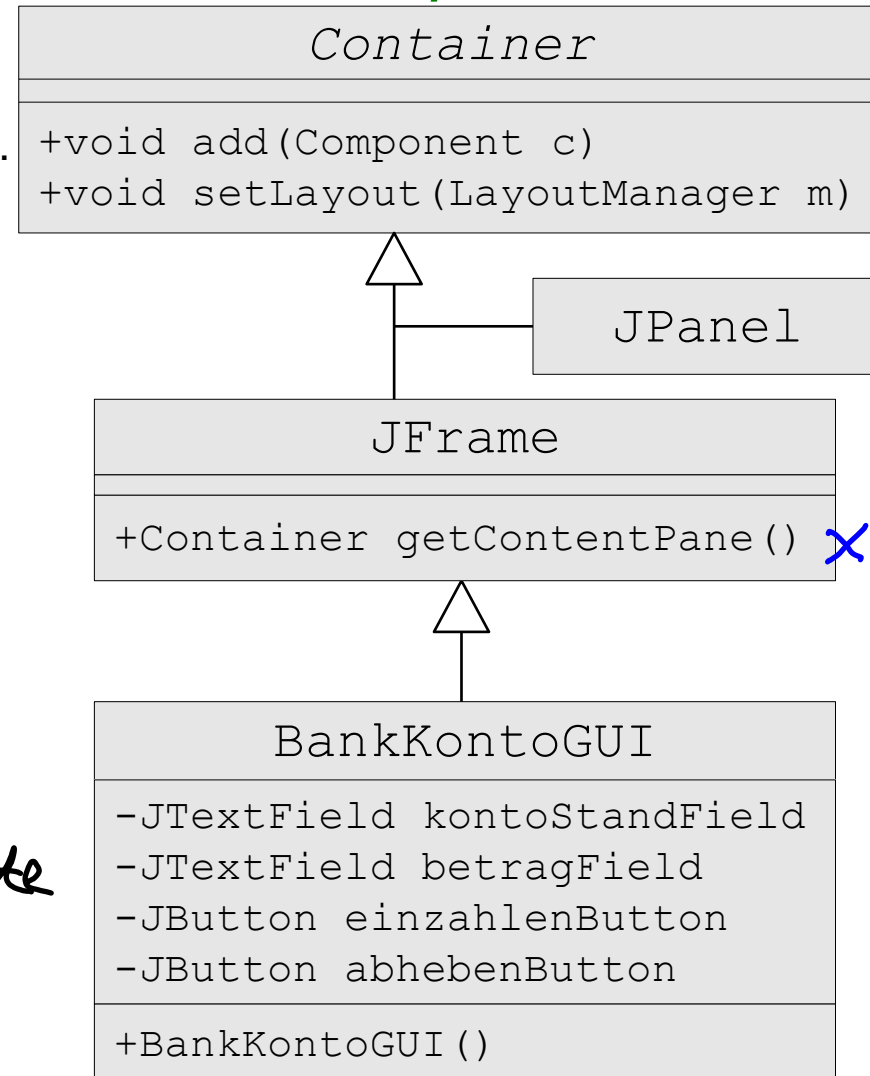
public BankKontoGUI() {
    ... // Initialisierung der Attribute wie vorher
    JPanel kontoStandPanel = new JPanel(); //lokale Var.
    kontoStandPanel.add(new JLabel("Kontostand:"));
    kontoStandPanel.add(this.kontoStandField);

    JPanel betragPanel = new JPanel(); //lokale Variable
    betragPanel.add(new JLabel("Betrag:"));
    betragPanel.add(this.betragField);

    JPanel buttonPanel = new JPanel(); //lokale Variable
    buttonPanel.add(this.einzahlenButton);
    buttonPanel.add(this.abhebenButton);

    ✗ Container contentPane = this.getContentPane();
    contentPane.setLayout(new GridLayout(3,1));
        //Grid mit 3 Zeilen und 1 Spalte
    contentPane.add(kontoStandPanel);
    contentPane.add(betragPanel);
    contentPane.add(buttonPanel);
}
    
```

↑ Spalte
Zeilen

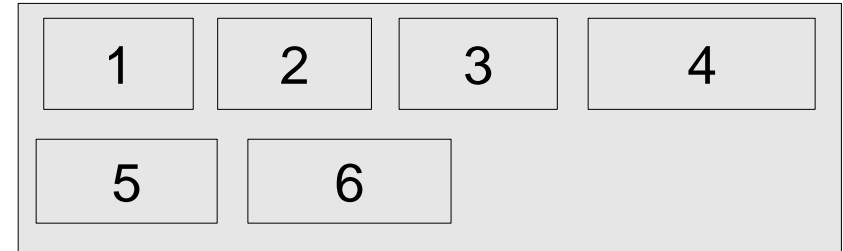


Einige LayoutManager

FlowLayout

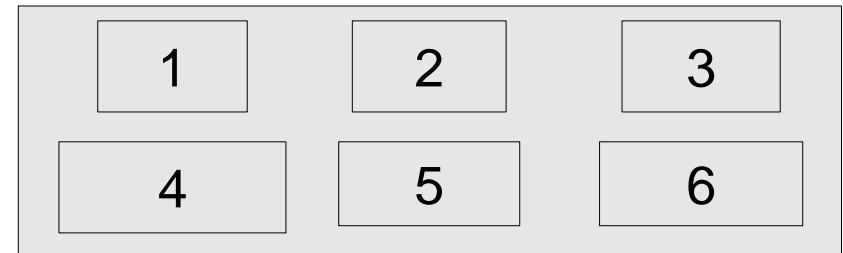
```
setLayout(new FlowLayout());
```

Standard bei JPanel.



GridLayout

```
setLayout(new GridLayout(2, 3));
```



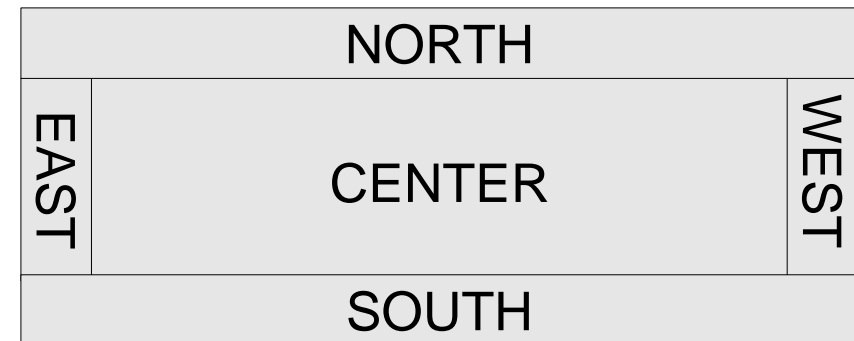
BorderLayout

```
setLayout(new BorderLayout());
```

Standard für den durch `getContentPane()` gelieferten Container in JFrame.

Einfügen an bestimmten Positionen, z.B.

```
add(component, BorderLayout.SOUTH);
```



Ansicht und Modell

- Bisher haben wir nur die **Ansicht** der Kontodaten implementiert.
- Wir brauchen noch die eigentlichen Daten, welche die GUI anzeigt, in unserem Beispiel ein Bankkonto. Man spricht von dem **Modell**.
Wir benutzen die uns bekannte Klasse `BankKonto`.
- Ansicht und Modell sollen entkoppelt sein, denn:
 - Modell existiert unabhängig von der Ansicht.
 - Häufig gibt es verschiedene Ansichten für dasselbe Modell, z.B. GUI-Interface und Web-Interface.
 - Getrennte Entwicklung von Modell und Ansicht unterstützt die Wartbarkeit.

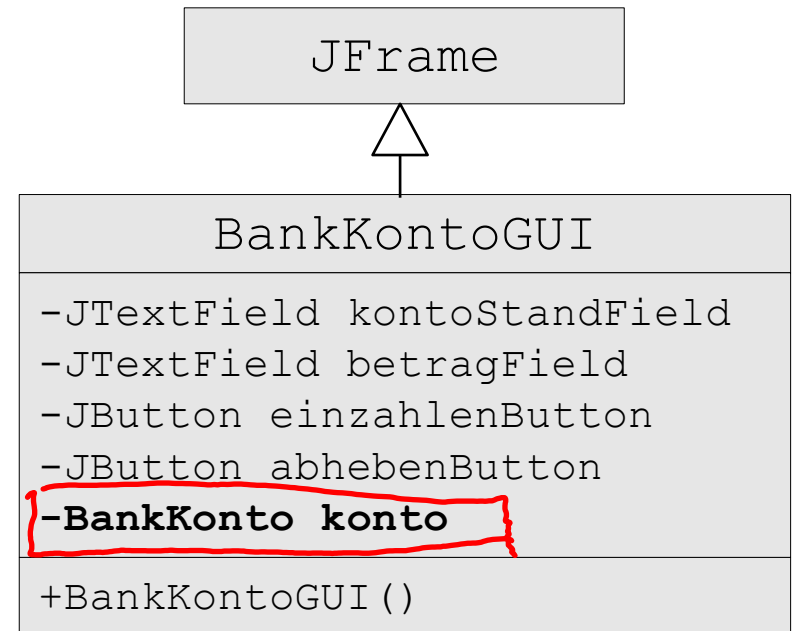
↳ Anwendungslogik

Ansicht und Modell im Beispiel

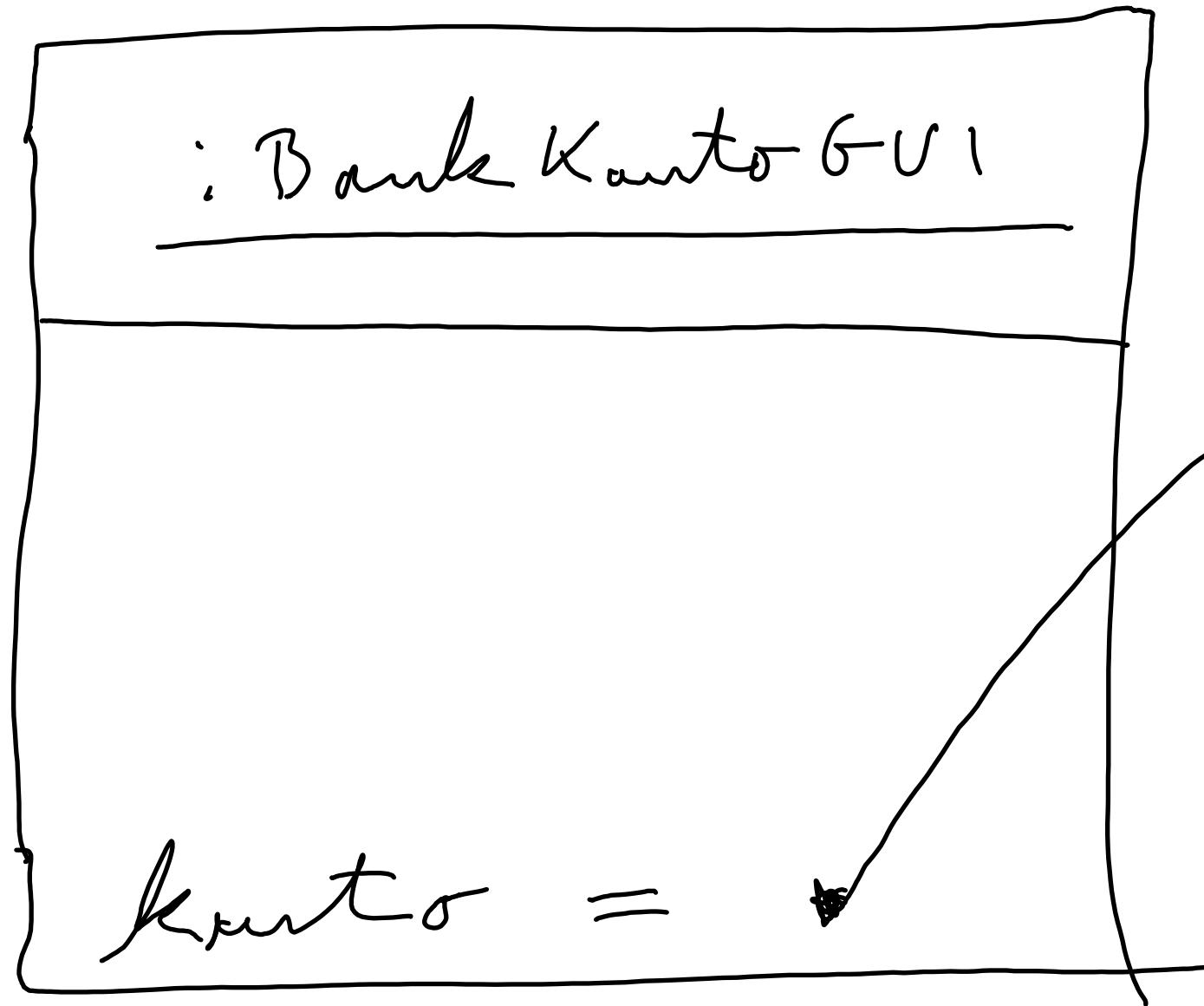
- Ansicht: Klasse BankKontoGUI, Modell: Klasse BankKonto,
- Ansicht-Objekte erhalten eine Referenz auf das anzuzeigende Modell.

```
public class BankKontoGUI extends JFrame {
    private JTextField kontoStandField;
    private JTextField betragField;
    private JButton einzahlenButton;
    private JButton abhebenButton;
    /* Referenz auf das Modell */
    private BankKonto konto;

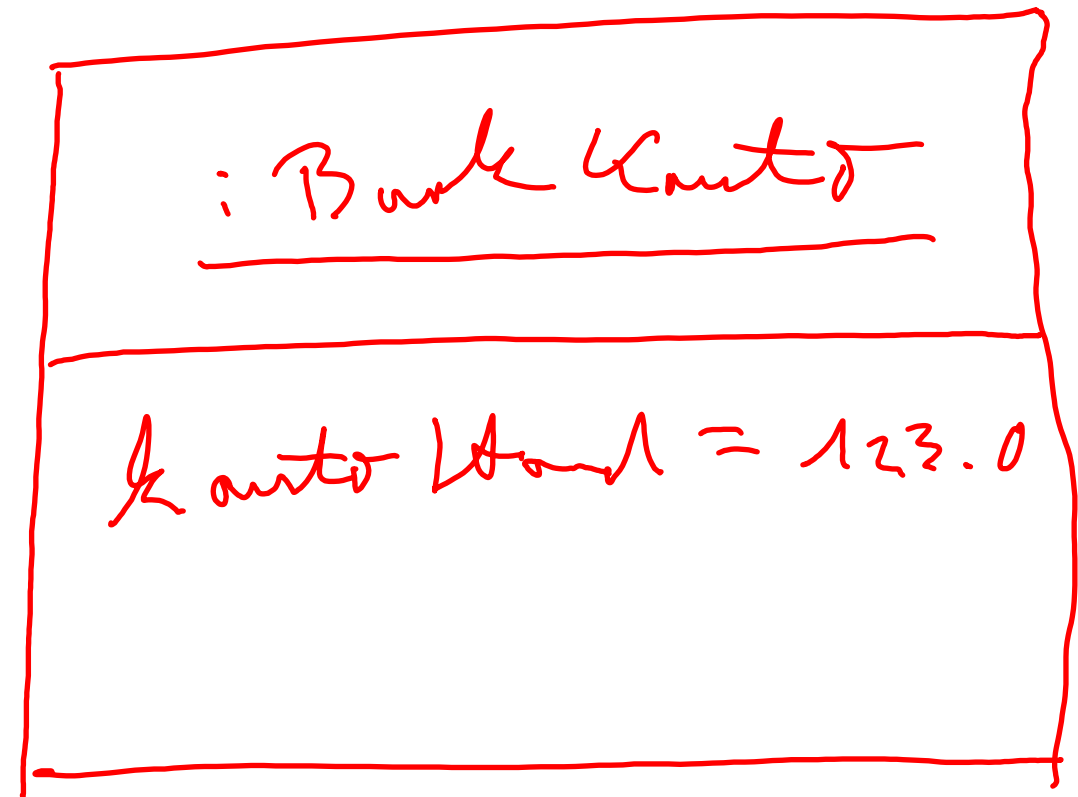
    public BankKontoGUI() {
        this.konto = new BankKonto(123.0);
        this.setTitle("Bankkonto");
        this.setSize(300, 200);
        this.kontoStandField = new JTextField(10);
        this.kontoStandField.setText(Double.toString(this.konto.getKontoStand()));
        this.kontoStandField.setEditable(false);
        ... //Initialisierung der anderen Attribute, Baumstruktur, Layout wie vorher
    }
}
```



Ansicht



Modell

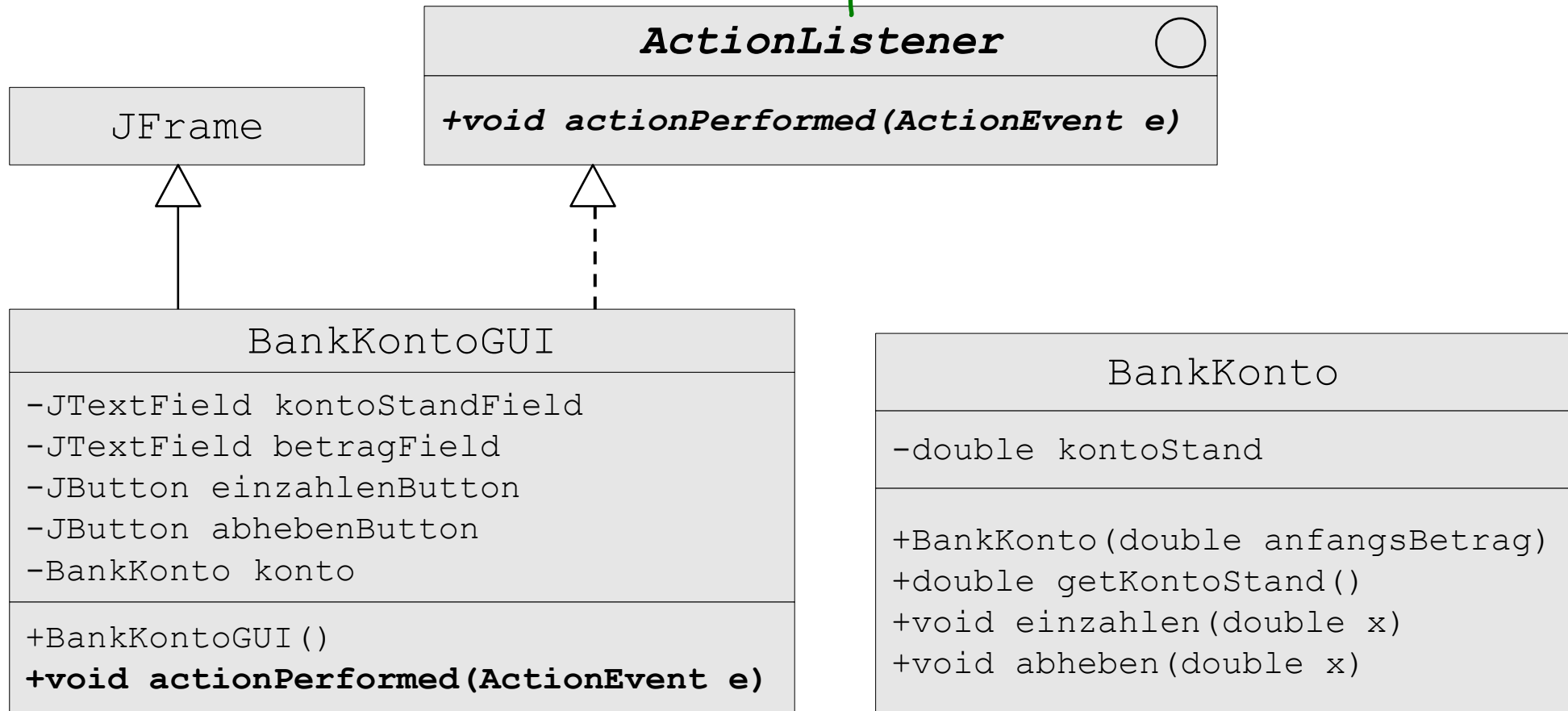


Ereignisse und ihre Behandlung

- In grafischen Anwendungen kann eine Vielzahl verschiedener Ereignisse auftreten, z.B. Tastatur betätigen, Maus klicken, Fenster verschieben, vergrößern, verkleinern, schließen, ...
- In AWT/Swing werden verschiedene Ereignisklassen unterschieden: `ActionEvent`, `WindowEvent`, `KeyEvent`, `MouseEvent`, ...
- Ist eine Komponente (z.B. ein `JFrame`) an Ereignissen eines bestimmten Typs (z.B. `ActionEvent`) interessiert und möchte darauf reagieren, dann muss sie:
 1. sich bei der Komponente, bei der ein solches Ereignis auftreten kann (z.B. ein `JButton`) als „Listener“ registrieren (z.B. Aufruf der Methode `addActionListener`),
 2. die beim Eintritt eines solchen Ereignisses von der Java-Laufzeitumgebung aufgerufene Operation (z.B. `actionPerformed`) des passenden Listener-Interfaces implementieren (z.B. `implements ActionListener`), indem sie
 3. die Reaktionen auf Ereignisse festlegt.

Ereignisbehandlung im Beispiel (UML-Darstellung)

Interface



Ereignisbehandlung im Beispiel (Java-Programm)

```
public class BankKontoGUI extends JFrame implements ActionListener {
    private JTextField kontoStandField;
    private JTextField betragField;
    private JButton einzahlenButton;
    private JButton abhebenButton;
    /* Referenz auf das Modell: */
    private BankKonto konto;

    public BankKontoGUI() {
        this.konto = new BankKonto(123.0);
        this.setTitle("Bankkonto");
        this.setSize(300, 200);
        ... //Initialisierung der Attribute, Baumstruktur und Layout wie vorher

        /* Registrierung der BankKontoGUI als Listener bei den Buttons */
        this.einzahlenButton.addActionListener(this);
        this.abhebenButton.addActionListener(this);
        /* Für ordnungsgemäße Beendigung der Anwendung bei Schließen (X) des
           Fensters */
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); } //Fortsetzung ->
```


Implementierung der Methode `actionPerformed`

```
public void actionPerformed(ActionEvent e) {
    Object source = e.getSource();
    if (source == this.einzahlenButton) {
        double betrag = Double.parseDouble(this.betragField.getText());
        this.konto.einzahlen(betrag);
        this.kontoStandField.
            setText(Double.toString(this.konto.getKontoStand()));
    }
    else if (source == this.abhebenButton) {
        double betrag = Double.parseDouble(this.betragField.getText());
        this.konto.abheben(betrag);
        this.kontoStandField.
            setText(Double.toString(this.konto.getKontoStand()));
    }
}

//Ende Klasse BankKontoGUI:
}
```

Vollständiges Programm (1)

Interface

```
import java.awt.*;
import javax.swing.*;
public class BankKontoGUI extends JFrame implements ActionListener {
```

1.

3.

⇒ BankKonto ist ein Subtyp von ActionListener

```
/* Attribute für GUI-Elemente */
private JTextField kontoStandField;
private JTextField betragField;
private JButton einzahlenButton;
private JButton abhebenButton;
```

1. Strukturvoller Aufbau

```
/* Referenz auf das Modell */
private BankKonto konto;
```

2. Verbindung Ansicht → Modell

```
public BankKontoGUI() {
    /* Objektverbindung Ansicht -> Modell herstellen*/
    this.konto = new BankKonto(123.0);
```

2.

```
/* Titel und Groesse der GUI */
this.setTitle("Bankkonto");
this.setSize(300, 200);
```

```
/* Initialisierung der Attribute */
this.kontoStandField = new JTextField(10);
this.kontoStandField.setText(Double.toString(this.konto.getKontoStand()));
this.kontoStandField.setEditable(false);
this.betragField = new JTextField(10);
this.einzahlenButton = new JButton("Einzahlen");
this.abhebenButton = new JButton("Abheben");
```

2.

1.

Vollständiges Programm (2)

```
/* Baumstruktur anlegen und Layout setzen */
JPanel kontoStandPanel = new JPanel(); //lokale Variable
kontoStandPanel.add(new JLabel("Kontostand:"));
kontoStandPanel.add(this.kontoStandField);

JPanel betragPanel = new JPanel(); //lokale Variable
betragPanel.add(new JLabel("Betrag:"));
betragPanel.add(this.betragField);

JPanel buttonPanel = new JPanel(); //lokale Variable
buttonPanel.add(this.einzahlenButton);
buttonPanel.add(this.abhebenButton);

Container contentPane = this.getContentPane();
contentPane.setLayout(new GridLayout(3,1)); //Grid mit 3 Zeilen und 1 Spalte
contentPane.add(kontoStandPanel);
contentPane.add(betragPanel);
contentPane.add(buttonPanel);
```

Vorbereitung
für 1. Zeile

} für 2. Zeile

} für 3. Zeile

1.

```
/* Registrierung der BankKontoGUI als Listener bei den Buttons */
this.einzahlenButton.addActionListener(this);
this.abhebenButton.addActionListener(this);
/* Für ordnungsgemäße Beendigung der Anwendung bei Schließen (X) des
   Fensters */
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
} //Ende Konstruktor BankKontoGUI
```

3.

Vollständiges Programm (3)

Methode aus dem Interface abimplementieren

```
/* Implementierung der Reaktionen auf Knopfdruck-Ereignisse */
```

```
public void actionPerformed(ActionEvent e) {
```

```
    Object source = e.getSource();
```

```
    if (source == this.einzahlenButton) {
```

```
        double betrag = Double.parseDouble(this.betragField.getText()); //Einlesen des Betrags
```

```
2. this.konto.einzahlen(betrag); //Einzahlen auf Konto
```

```
        this.kontoStandField.setText(Double.toString(this.konto.getKontoStand()));
```

2.

```
//Ausgabe neuer Kontostand
```

```
    }
```

```
    else if (source == this.abhebenButton) {
```

```
        double betrag = Double.parseDouble(this.betragField.getText()); //Einlesen des Betrags
```

```
2. this.konto.abheben(betrag); //Abheben vom Konto
```

```
        this.kontoStandField.setText(Double.toString(this.konto.getKontoStand()));
```

2.

```
//Ausgabe neuer Kontostand
```

```
    }
```

```
} //Ende Methode actionPerformed
```

```
① } //Ende Klasse BankKontoGUI
```

```
/* Main-Klasse zum Start des Programms */
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        BankKontoGUI gui = new BankKontoGUI();
```

```
        gui.setVisible(true);
```

```
    }
```

```
}
```