Objektorientierte Programmierung in Java

Vorlesung 6 - Abstrakte Elemente

Emily Lucia Antosch

HAW Hamburg

30.06.2025

Inhaltsverzeichnis

1. Einleitung

1.1 Wo sind wir gerade?

- In der letzten Vorlesung ging es um Schnittstellen und abstrakte Klassen
- Sie können nun
  - abstrakte Klassen verwenden, um ihren Code noch genau zu strukturieren,
  - Schnittstellen implementieren, um Eigenschaften von Klassen darzustellen,
  - Klassen und Objekten mit einer Ordnung mittels `Comparable` belegen,
- Heute geht es weiter mit den graphischen Oberflächen.

1.1 Wo sind wir gerade?

1. Imperative Konzepte
2. Klassen und Objekte
3. Klassenbibliothek
4. Vererbung
5. Schnittstellen
6. Graphische Oberflächen
7. Ausnahmebehandlung
8. Eingaben und Ausgaben
9. Multithreading (Parallel Computing)

1.2 Das Ziel dieses Kapitels

- Sie erstellen grafische Oberflächen mit z.B. Menüs, Buttons und Textfeldern.
- Sie zeichnen Diagramme aus einfachen geometrischen Formen (z.B. Linien, Kreise).
- Sie reagieren auf Ereignisse (z.B. Drücken eines Buttons), indem Sie grafische Elemente mit bei Benutzereingaben auszuführenden Methoden verbinden.
- Sie verwenden das Observer-Pattern, damit Objekte beliebiger Datentypen auf Ereignisse reagieren können.

2. Grundlegende Struktur

? Frage

- Welche Arten von Elementen sehen Sie?
- Wie reagieren die Elemente? Hängen hierbei Elemente zusammen?

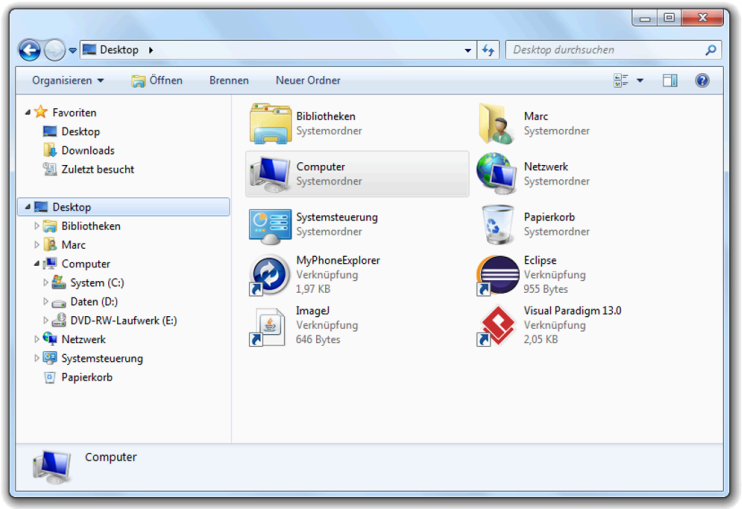


Abbildung 1: Windows 7 Explorer

2.1 Graphische UI

- Grafische Benutzeroberfläche: Graphical user interface (GUI)
- Klassenbibliotheken AWT und Swing bereits im Java SDK enthalten
- Abstract Window Toolkit (AWT):
  - Bereits mit Java 1.0 eingeführt
  - Nur grundlegende Oberflächenelemente, um möglichst viele Betriebssysteme zu unterstützen („Kleinster gemeinsamer Nenner“)
  - Setzt die nativen Elemente („Widgets“) des Betriebssystems ein
  - Ursprünglich voller Design-Fehler, da unter großem Druck in nur knapp zwei Monaten entstanden
- Swing:
  - Erweiterung des AWT
  - Kein direktes Ansprechen der Fensterfunktionen der aktuellen Plattform mehr
  - Komplette Kontrolle über die Anzeigeelemente

2.1 Graphische UI

- Basiselement: Rahmen (Frame)
- Beinhaltet Fensterleiste mit Titel und Steuerelementen (z.B. „Schließen“)
- Beinhaltet Bereich, in dem Elemente platziert werden können (Content pane)
- Kann zusätzlich Menüleiste (Menu bar) enthalten

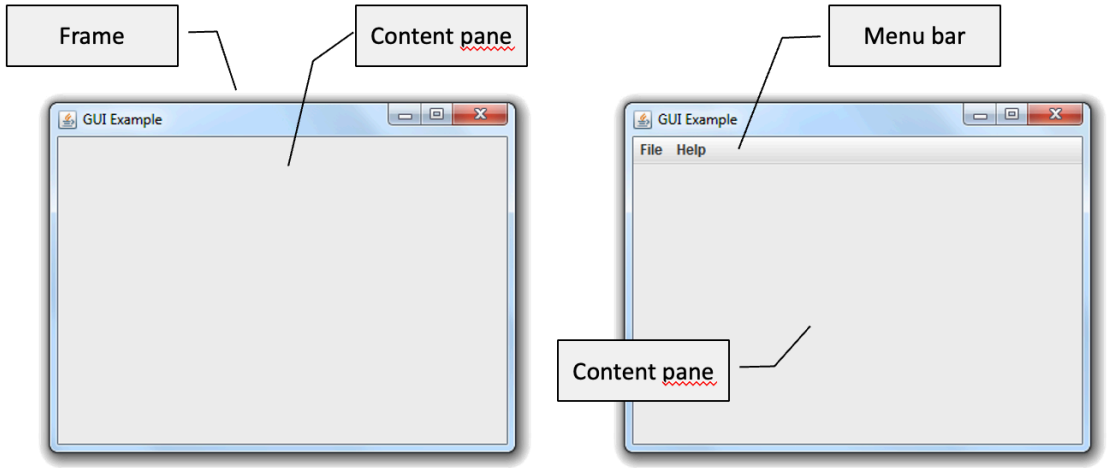


Abbildung 2: Aufbau eines Frames

2.1 Graphische UI

- Elemente werden hierarchisch hinzugefügt.
- Für Elemente, die andere Elemente aufnehmen, lässt sich das Layout angeben.

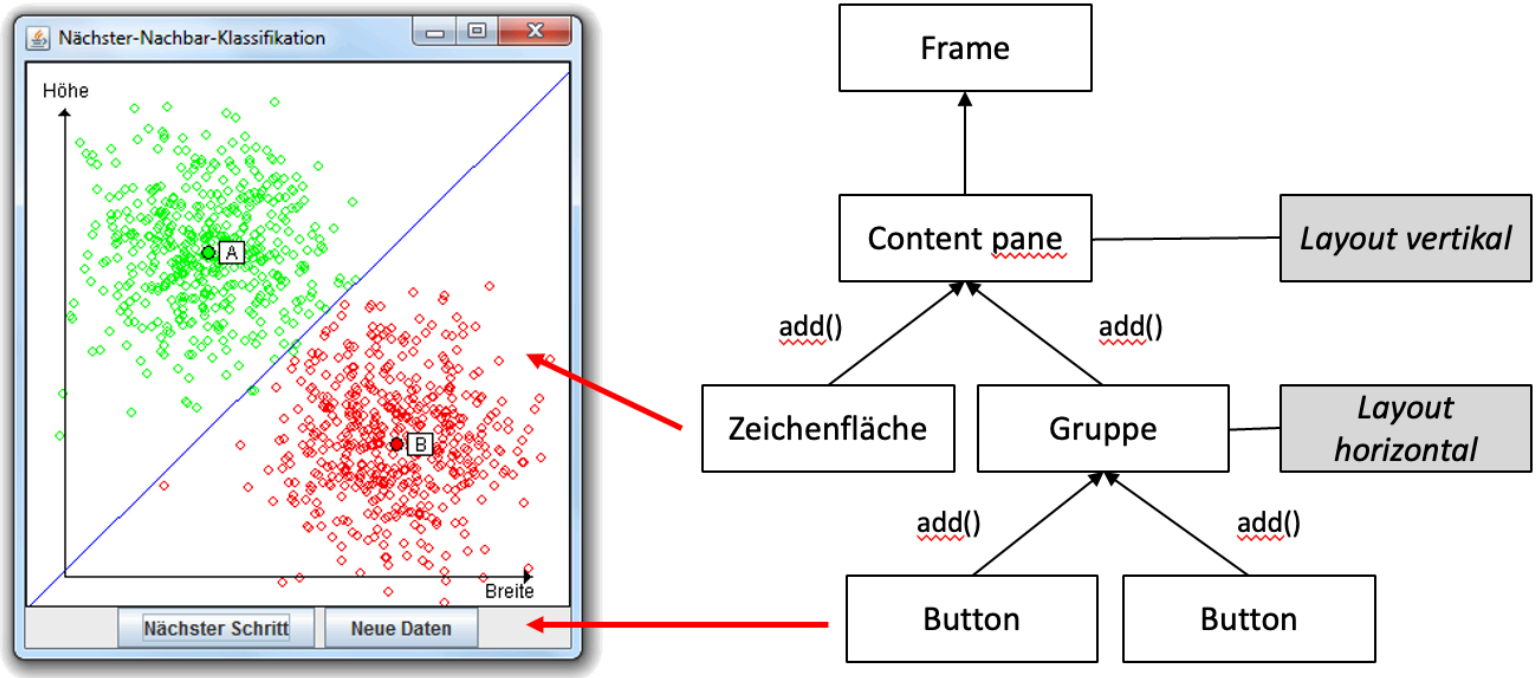


Abbildung 3: Hierarchie eines Windows

3. Graphische Oberflächen erzeugen

3.1 Einfaches Programm

- Ausführbare main()-Methode erzeugt Objekt der Klasse
- Klasse erzeugt im Konstruktur Frame mit grafischer Oberfläche
- „Close Operation“ so angeben, dass Anwendung bei Schließen des Fensters beendet wird

Java

```java
public class HelloWorld {
    public HelloWorld() {
        JFrame frame = new JFrame("GUI example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new HelloWorld();
    }
}
```

3.1 Einfaches Programm

? Frage

- Was passiert, wenn die „Close Operation“ nicht auf „Exit on close“ gesetzt wird?
- Warum muss man das Fenster extra über setVisible(true) anzeigen?

- Und es sieht nicht wirklich schön aus:
  - Das Fenster ist zu klein!
  - Das Fenster „klebt“ in der linken oberen Ecke!

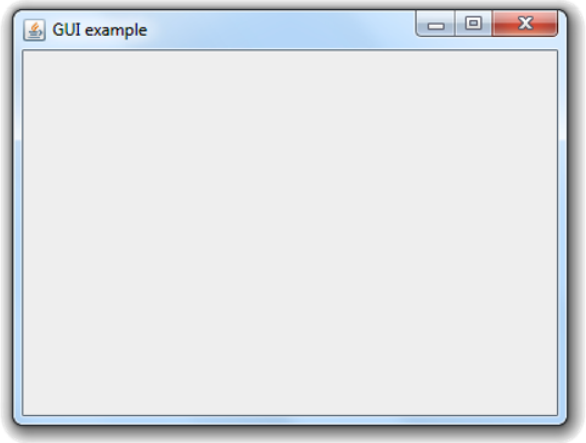
Aufgabe 1

- Vergrößern Sie es auf 400 x 300 Pixel (Breite x Höhe).
- Platzieren Sie es jeweils 50 Pixel vom linken und vom oberen Rand.
- Hinweis: Lassen Sie sich die Methoden von frame anzeigen.

3.1 Einfaches Programm

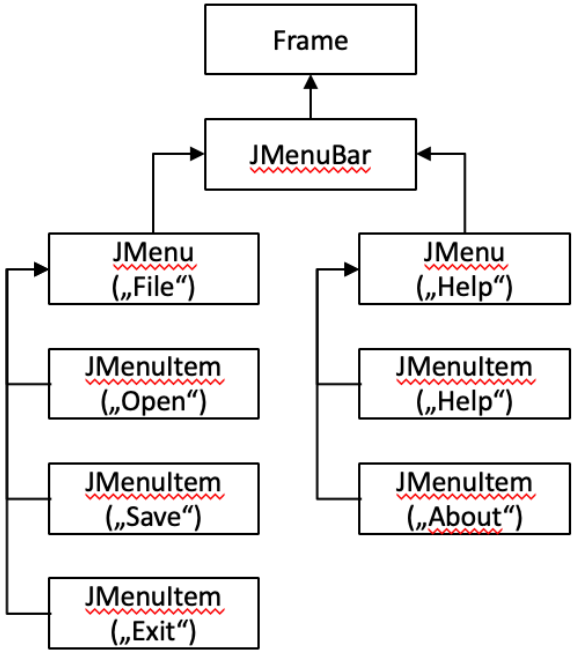
- Korrigierte Größe und Position:

```java
public class HelloWorld {
    public HelloWorld() {
        JFrame frame = new JFrame("GUI example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setLocation(50, 50);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new HelloWorld();
    }
}
```



3.1 Einfaches Programm

- Klassen:
  - JMenuBar: Menüleiste
  - JMenu: Menü in Menüleiste (z.B. Datei, Hilfe)
  - JMenuItem: Eintrag in einem Menü (z.B. Neu, Speichern als)
- Fügen Sie unserem Programm folgende Menüs hinzu:
  - Menü File mit den Menüpunkten Open, Save und Exit
  - Menü Help mit den Menüpunkten Help und About

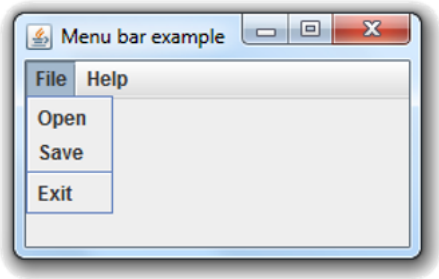


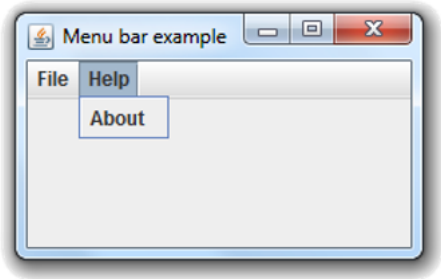
🔥 Tipp

- Elemente werden meist über add() hinzugefügt.
- Die Menüleiste wird über setJMenuBar() hinzugefügt.

3.1 Einfaches Programm

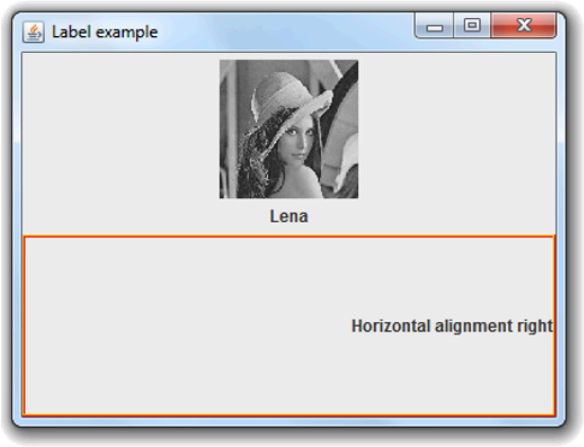
```java
public MenuBar() {
    JFrame frame = new JFrame("Menu bar example");
    // Set frame properties ...

    JMenuBar menuBar = new JMenuBar();   // Create menu bar and add to frame
    frame.setJMenuBar(menuBar);

    JMenu menuFile = new JMenu("File");  // Create menu "File"
    menuBar.add(menuFile);
    menuFile.add(new JMenuItem("Open"));
    menuFile.add(new JMenuItem("Save"));
    menuFile.addSeparator();
    menuFile.add(new JMenuItem("Exit"));

    JMenu menuHelp = new JMenu("Help");  // Create menu "Help"
    menuBar.add(menuHelp);
    menuHelp.add(new JMenuItem("About"));

    frame.setVisible(true);
}
```





3.1 Einfaches Programm

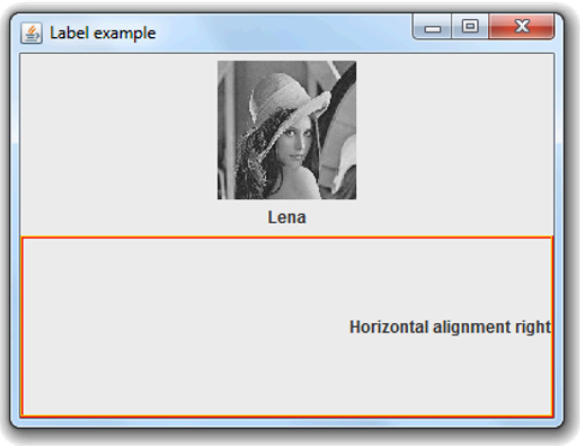
- Klasse JLabel stellt nicht editierbaren Text dar
  - Kann horizontal und vertikal ausgerichtet werden (z.B. zentriert)
  - Kann Rahmen zeichnen
  - Kann auch Bilder darstellen
- Erzeugen wir das rechts abgebildete Fenster:
  - Bild über new ImageIcon() laden
  - Rahmen über BorderFactory.createEtchedBorder()
  - Label der Content pane über add() hinzufügen
  - Layout über frame.setLayout(new GridLayout(2, 1))



3.1 Einfaches Programm

Java

```java
// Create frame and set properties
JFrame frame = new JFrame("Label example");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400, 300);
frame.setLocation(50, 50);
frame.setLayout(new GridLayout(2, 1));  // 2 rows, 1 column

// Create labels
ImageIcon image = new ImageIcon("folien07_gui/Lena100.jpg");
JLabel label1 = new JLabel("Lena", image, JLabel.CENTER);
label1.setHorizontalTextPosition(JLabel.CENTER);
label1.setVerticalTextPosition(JLabel.BOTTOM);

JLabel label2 = new JLabel("Horizontal alignment right");
label2.setHorizontalAlignment(JLabel.RIGHT);
label2.setBorder(BorderFactory.createEtchedBorder(Color.RED, Color.ORANGE));

// Add labels to content pane
Container contentPane = frame.getContentPane();
contentPane.add(label1);
contentPane.add(label2);

frame.setVisible(true);
```



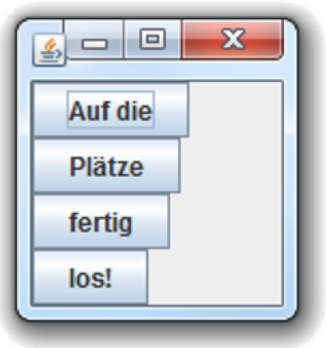
4. Layout

4.1 Layout-Manager

- Legen die Anordnung der GUI-Elemente fest
- Verschiedene Layout-Manager definiert, z. B.:
  - BoxLayout:
    - Elemente übereinander („vertikal“) oder nebeneinander („horizontal“)
  - GridLayout:
    - Elemente in gleichmäßigem Gitter platziert
    - Alle Zellen haben die gleiche Größe
  - FlowLayout:
    - Elemente wie bei horizontalem BoxLayout in Zeile platziert
    - Allerdings Zeilenumbruch, sobald eine Zeile „voll“ ist

4.1 Layout-Manager

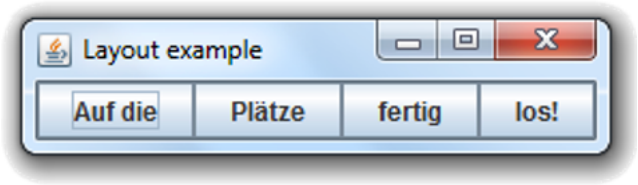
```java
// Create frame and set properties
JFrame frame = new JFrame("Layout example");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setLocation(50, 50);

// Create contents
Container contentPane = frame.getContentPane();
contentPane.setLayout(new BoxLayout(contentPane,
BoxLayout.Y_AXIS));
contentPane.add(new JButton("Auf die"));
contentPane.add(new JButton("Plätze"));
contentPane.add(new JButton("fertig"));
contentPane.add(new JButton("los!"));

frame.pack();
frame.setVisible(true);
```



4.1 Layout-Manager

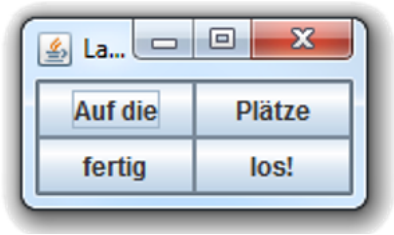
- Horizontales BoxLayout:

```java
contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.X_AXIS));
```



- GridLayout:

```java
contentPane.setLayout(new GridLayout(2, 2));
```

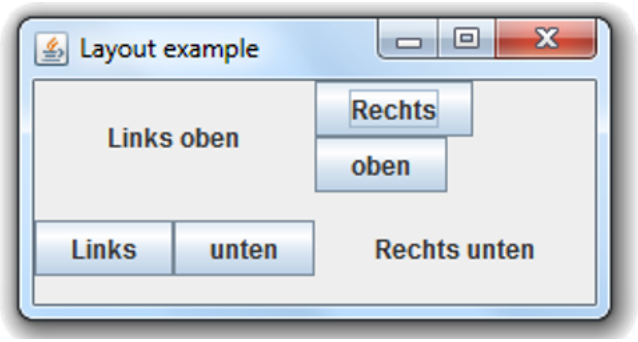


4.1 Layout-Manager

- Elemente lassen sich in Objekten der Klasse JPanel gruppieren.
- Jedes JPanel-Objekt besitzt einen eigenen Layout-Manager.

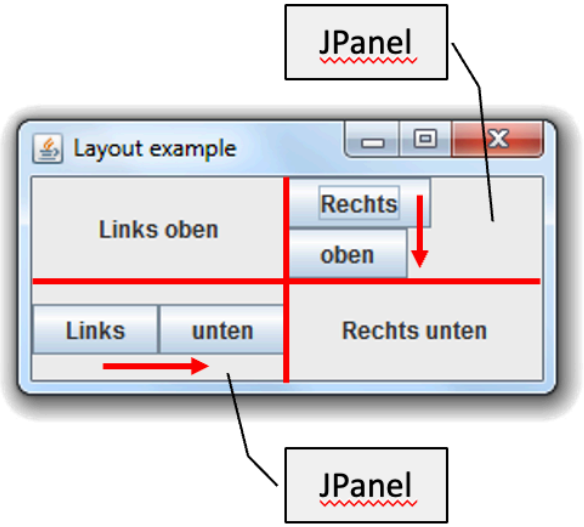
? Frage

- Welche Elemente beinhaltet das gezeigte Fenster?
- Über welche Objekte und Layout-Manager sind diese angeordnet?



4.1 Layout-Manager

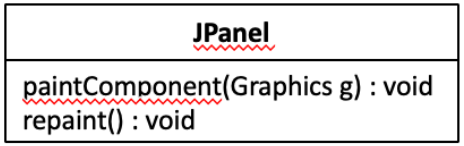
```java
JFrame frame = new JFrame("Layout example");

JPanel panel1 = new JPanel();
panel1.setLayout(new BoxLayout(panel1, BoxLayout.Y_AXIS));
panel1.add(new JButton("Rechts"));
panel1.add(new JButton("oben"));

JPanel panel2 = new JPanel();
panel2.setLayout(new BoxLayout(panel2, BoxLayout.X_AXIS));
panel2.add(new JButton("Links"));
panel2.add(new JButton("unten"));

Container contentPane = frame.getContentPane();
contentPane.setLayout(new GridLayout(2, 2));
contentPane.add(new JLabel("Links oben", JLabel.CENTER));
contentPane.add(panel1);
contentPane.add(panel2);
contentPane.add(new JLabel("Rechts unten", JLabel.CENTER));

frame.pack();
frame.setVisible(true);
```



5. Zeichnen

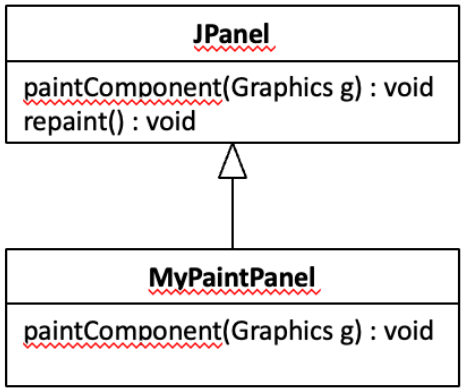
5.1 Zeichnen

- Klasse JPanel als Zeichenfläche:
  - Auf Panel lässt sich frei zeichnen.
- Zeichenmethode:
  - System führt zum Zeichnen Methode paintComponent() aus
  - Wird automatisch bei Änderungen am Fenster aufgerufen
  - Methode erhält Parameter vom Typ Graphics (Grafik-Kontext)
  - Graphics besitzt Methoden zum Zeichnen (Texte, Linien, Rechtecke, Bögen, …)
- Explizites Neuzeichnen:
  - Neuzeichnen kann auch über Methode repaint() veranlasst werden.
  - Diese ruft intern paintComponent() auf.



5.1 Zeichnen

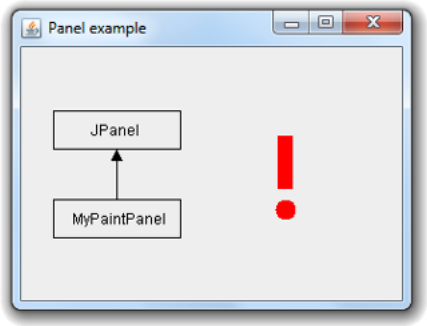
- Okay, es gibt also JPanel mit der Methode paintComponent().
- Es wird gezeichnet, was in paintComponent() steht.
- Aber wie kann man Zeichenbefehle zu dieser Methode hinzufügen?!
- Lösung:
  - Leiten Sie von JPanel ab und überlagern Sie paintComponent().
  - Hierdurch: Panel-Klasse mit frei definierbarer Zeichenmethode




```java
class MyPaintPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // Code for own drawings ...
    }
}
```

5.1 Zeichnen

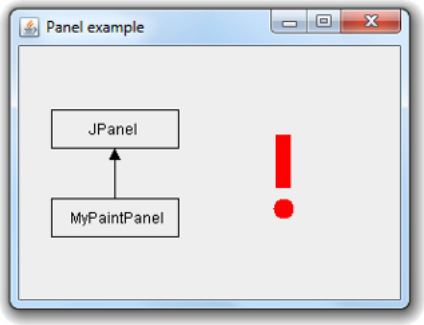
```java
class MyPaintPanel extends JPanel {
    public Dimension getPreferredSize() {
        return new Dimension(300, 200);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        g.setColor(Color.BLACK);
        g.drawRect(25, 50, 100, 30);  // Super class
        g.drawString("JPanel", 55, 70);
        g.drawRect(25, 120, 100, 30); // Sub class
        g.drawString("MyPaintPanel", 40, 140);
        g.drawLine(75, 80, 75, 120);  // Arrow
        g.fillPolygon(new int[]{70, 75, 80}, new int[]{90, 80, 90}, 3);

        g.setColor(Color.RED);
        g.fillRect(202, 70, 12, 42);
        g.fillOval(200, 120, 16, 16);
    }
}
```



5.1 Zeichnen

- Einbinden in graphische Oberfläche:

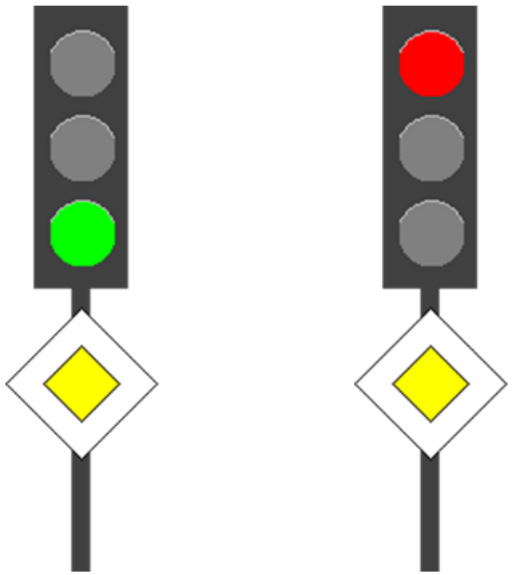
```java
public class PaintPanel {
    public PaintPanel() {
        JFrame frame = new JFrame("Panel example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocation(50, 50);

        frame.add(new MyPaintPanel());
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new PaintPanel();
    }
}
```



5.1 Zeichnen

Aufgabe 2

- Entdecken Sie Ihre künstlerische Ader!
- Erstellen Sie ein Programm, das eine Ampel anzeigt.



6. Buttons & Ereignisse

6.1 Aufgabe

- Unser Ziel ist folgende Anwendung:
  - Fenster mit drei Buttons und einem Panel
  - Auswahl der Buttons färben das Panel rot, blau bzw. in zufälliger Farbe

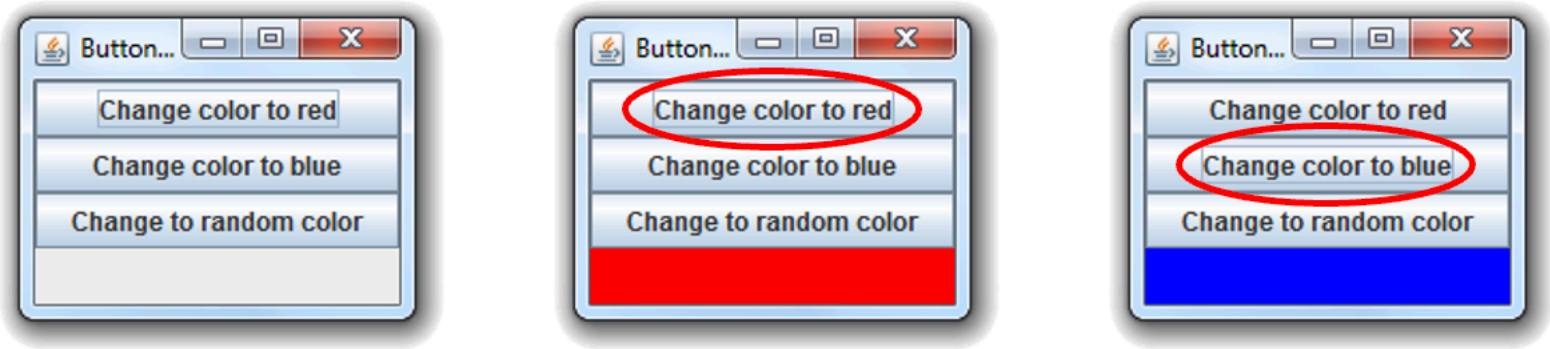


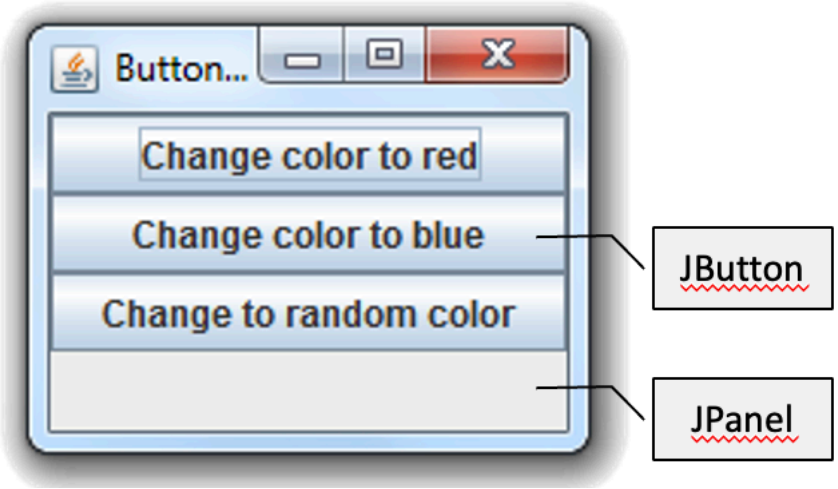
Abbildung 21: Buttons, die eine Farbe ändern

- Wir benötigen hierzu:
  - Buttons als Elemente
  - Möglichkeit auf gedrückten Button zu reagieren

6.1 Aufgabe

Aufgabe 3

- Erzeugen Sie zunächst die GUI mit ihren Elementen.



6.1 Aufgabe

- Erzeugen von Elementen der Klasse JButton:

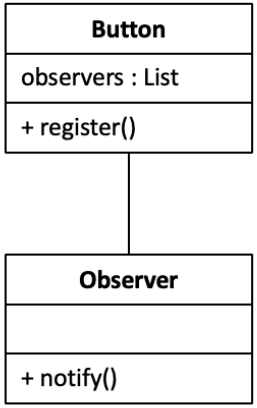
```java
public class ButtonEvent {
    public ButtonEvent() {
        JFrame frame = new JFrame("Button example");
        // Set frame properties ...

        // Create and layout contents
        frame.setLayout(new GridLayout(4, 1));  // 4 rows, 1 column
        Container contentPane = frame.getContentPane();
        contentPane.add(new JButton("Change color to red"));
        contentPane.add(new JButton("Change color to blue"));
        contentPane.add(new JButton("Change to random color"));
        contentPane.add(new JPanel());
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new ButtonEvent();
    }
}
```

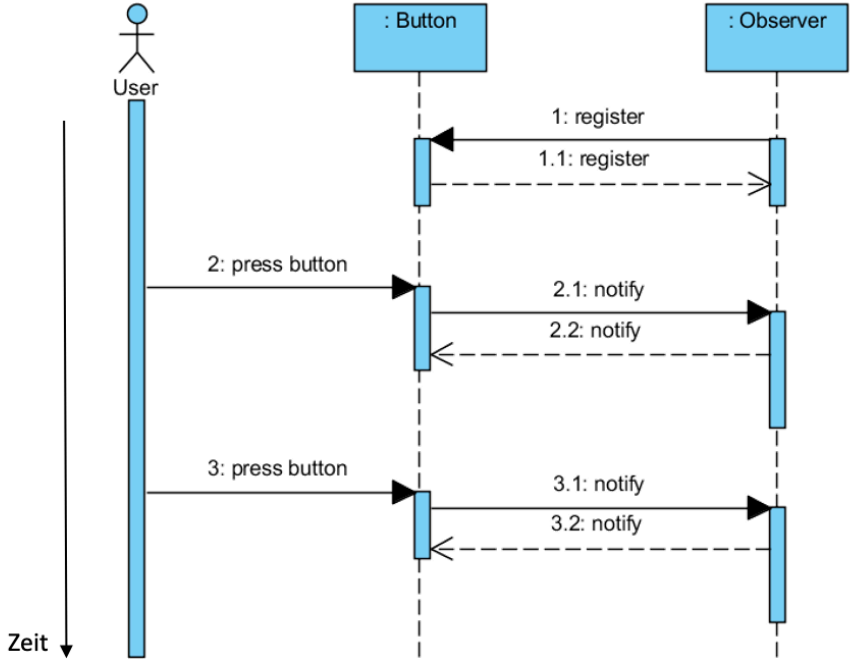
6.1 Aufgabe

- Wie können wir aber darauf reagieren, wenn ein Button gedrückt wird?
- Beteiligte Objekte:
  - Button mit Zustand (z.B. „nicht gedrückt“, „gedrückt“)
  - Objekt, das bei Änderungen des Buttons benachrichtigt werden soll
- Prinzipielles Vorgehen:
  - Registrierung:
    - Objekt „sagt dem Button“, dass es bei Änderungen benachrichtigt werden möchte
    - Button merkt sich (z. B. in Liste), welche Objekte benachrichtigt werden sollen
  - Button wird gedrückt:
    - Button benachrichtigt Objekte in der Liste, dass sein Zustand geändert wurde

6.1 Aufgabe

- Mögliche Umsetzung:
  - Button: Methode register(), um Beobachter zur Liste hinzuzufügen
  - Observer: Methode notify(), die Button-Objekt zur Benachrichtigung aufruft





6.1 Aufgabe

- Aaaaaaaber:
  - JButton kann von uns erstellte Klassen nicht kennen.
  - Kann daher auch nicht wissen, ob wir Methode notify() implementiert haben.
- Lösung:
  - Beobachter implementieren ein definiertes Interface
  - Button braucht die Klasse des Beobachters nicht kennen, nur das Interface

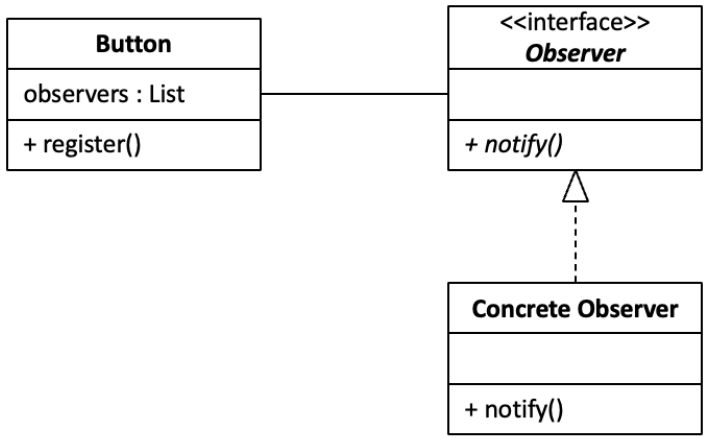
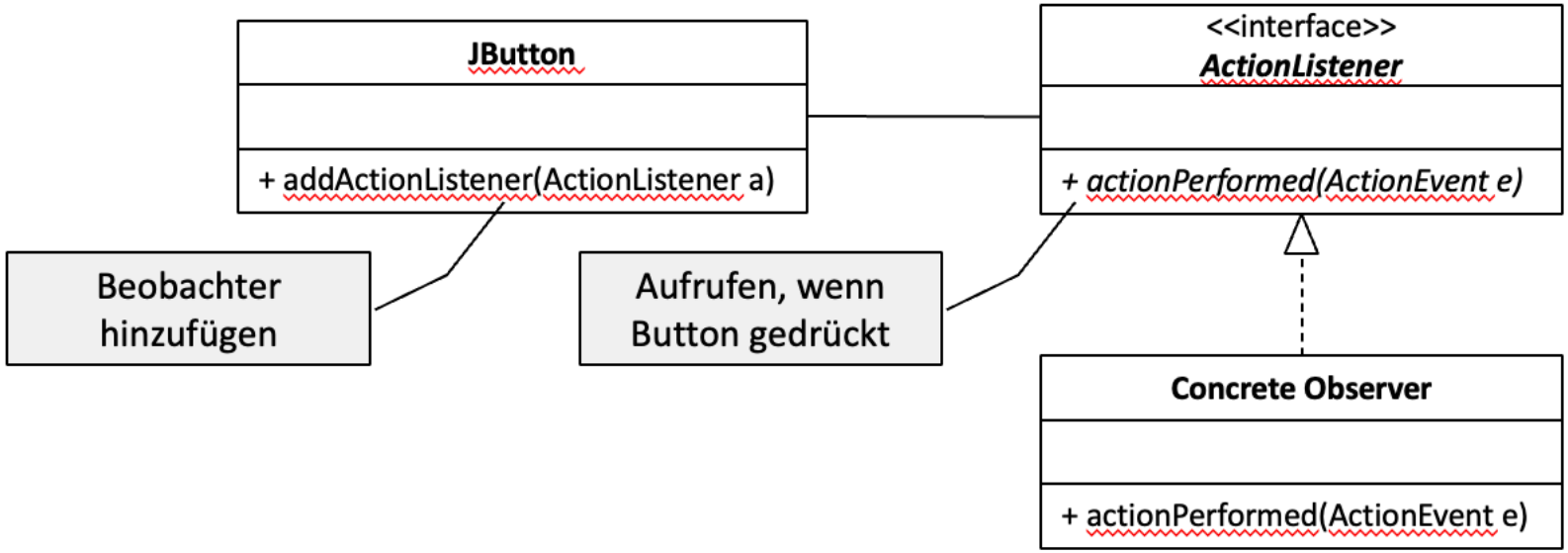


Abbildung 24: Interface `Oberserver`

6.1 Aufgabe

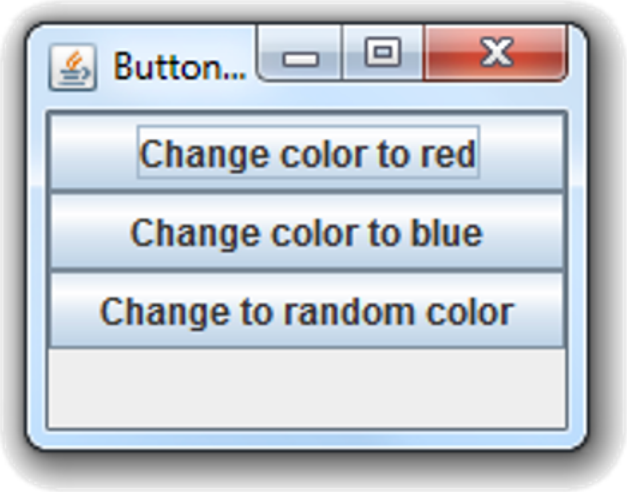
- Ansatz wird auch Observer pattern genannt
- Es können sich mehr als ein Beobachter registrieren.
- In Swing Namen des Interfaces und der Methoden anders gewählt:



6.1 Aufgabe

Aufgabe 4

- Ausführbare Klasse implementiert Interface ActionListener
- Objekt der ausführbaren Klasse registriert sich bei den Buttons



6.1 Aufgabe

- Auszug aus Quelltext:

```java
public class ButtonEvent implements ActionListener {
    private JPanel panel;
    private JButton buttonRed, buttonBlue, buttonRandom;

    public ButtonEvent() {
        // ...

        // Buttons with event handling
        buttonRed = new JButton("Change color to red");
        buttonBlue = new JButton("Change color to blue");
        buttonRandom = new JButton("Change to random color");

        buttonRed.addActionListener(this);
        buttonBlue.addActionListener(this);
        buttonRandom.addActionListener(this);

        // ...
    }
```

```
19    }
```

6.1 Aufgabe

- Reaktion auf Ereignisse (Auszug aus Quelltext):
  - Button über Methode `getSource()` des Ereignisobjektes identifiziert

Java

```java
public class ButtonEvent implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == buttonRed) {
            panel.setBackground(Color.RED);
        } else if (event.getSource() == buttonBlue) {
            panel.setBackground(Color.BLUE);
        } else if (event.getSource() == buttonRandom) {
            Random random = new Random();
            float red = random.nextFloat();
            float green = random.nextFloat();
            float blue = random.nextFloat();
            Color color = new Color(red, green, blue);
            panel.setBackground(color);
        }
    }
}
```

6.1 Aufgabe

- Alternativ (in actionPerformed() herausfinden, welcher Button gedrückt wurde):
- Buttons mit einem String verbinden, z.B.:

```java
buttonRed.setActionCommand("Change color to red");
buttonBlue.setActionCommand("Change color to blue");
buttonRandom.setActionCommand("Change to random color");
```

- String in Methode `actionPerformed()` abfragen und nutzen:

```java
public void actionPerformed(ActionEvent event) {
    String actionCommand = event.getActionCommand();

    if (actionCommand.equals("Change color to red")) {
        // ...
    } else if (actionCommand.equals("Change color to blue")) {
        // ...
    } else if (actionCommand.equals("Change to random color")) {
        // ...
    }
}
```

6.1 Aufgabe

- Neuen `ActionListener` mit Methode `actionPerformed()` inline definieren

```java
public class ButtonEvent2 {
    private JPanel panel;

    public ButtonEvent2() {
        // ...

        buttonRed.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                panel.setBackground(Color.RED);
            }
        });

        // ...
    }

    public static void main(String[] args) {
        new ButtonEvent2();
    }
```

```
19 }
```

6.1 Aufgabe

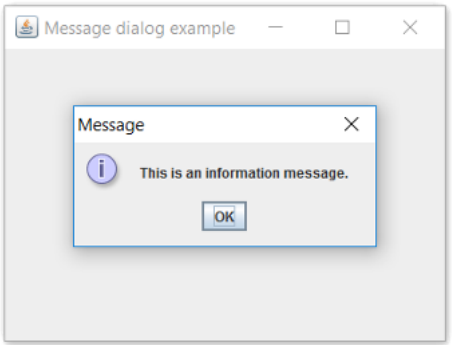
- Alle Swing-Komponenten können folgende Beobachter registrieren:
  - Component listener: Änderungen der Größe, Position oder Sichtbarkeit
  - Focus listener: Komponente erhält oder verliert den Tastatur-Fokus
  - Key listener: Tastatur-Ereignisse (nur wenn die Komponente den Keyboard-Fokus hat)
  - Mouse listener: Maus-Klicks, Drücken, Loslassen und Mausbewegungen
  - Mouse motion listener: Änderungen der Cursor-Position über der Komponente
  - Mouse wheel listener: Änderung des Mausrads über der Komponente

7. Einfache Dialoge

7.1 Einfache Dialoge

- Beispiele für Dialoge über JOptionPane:

```java
public class MessageDialogs {
    public MessageDialogs() {
        // Create and show frame
        JFrame frame = new JFrame("Message dialog example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setLocationByPlatform(true);
        frame.setVisible(true);

        // Display dialogs
        JOptionPane.showMessageDialog(frame, "This is a plain message.",
"Message",
            JOptionPane.PLAIN_MESSAGE);
        JOptionPane.showMessageDialog(frame, "This is an information
message.", "Message",
            JOptionPane.INFORMATION_MESSAGE);
        JOptionPane.showMessageDialog(frame, "This is a warning.", "Message",
            JOptionPane.WARNING_MESSAGE);
    }

```



7.1 Einfache Dialoge

```
19      public static void main(String[] args) {
20          new MessageDialogs();
21      }
22  }
```

8. Anregungen

8.1 Ideen zum Experimentieren

- Einige weitere GUI-Elemente:
  - Textfelder über `JTextField`, `JPasswordField` und `JTextArea`
  - Auswahlboxen über `JCheckBox`
  - Listen über `JComboBox` und `JList`
  - Tooltips über Methode `setToolTipText()`
  - Datei auswählen über `JFileChooser`

9. License Notice

9.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- https://creativecommons.org/licenses/by-nc-sa/4.0/
- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.