

Objektorientierte Programmierung in Java

Vorlesung 6 - Abstrakte Elemente

Emily Lucia Antosch

HAW Hamburg

29.10.2024

Inhaltsverzeichnis

1. Einleitung	3
2. Grundlegende Struktur	7
3. Graphische Oberflächen erzeugen	12
4. Layout	20
5. Zeichnen	26
6. Buttons & Ereignisse	32
7. Einfache Dialoge	46
8. Anregungen	49
9. License Notice	51

1. Einleitung

- 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)

- 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?

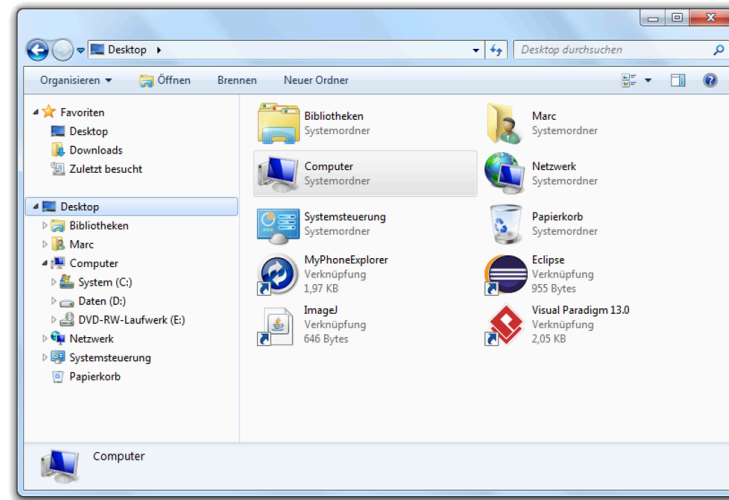


Abbildung 1: Windows 7 Explorer

- 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 („Kleinsten 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

2. Grundlegende Struktur

- 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

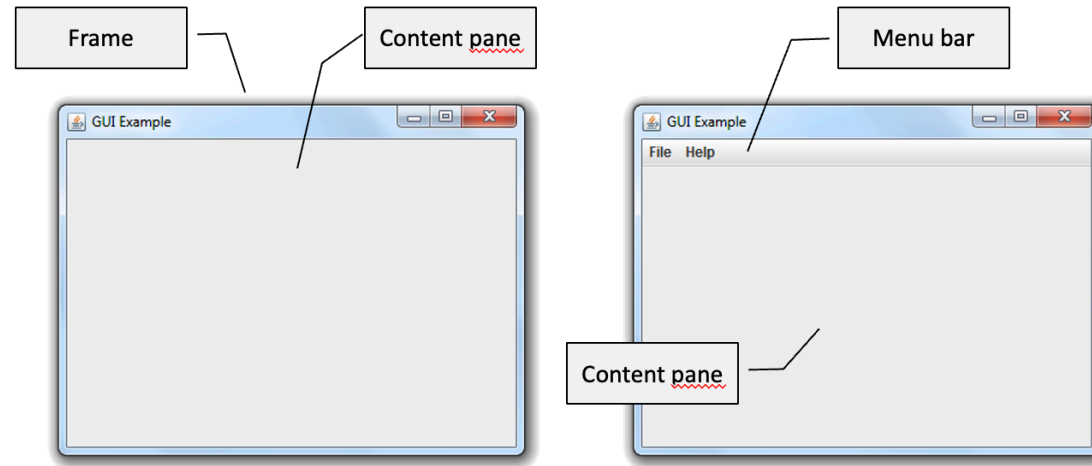


Abbildung 2: Aufbau eines Frames

2.1 Graphische UI

2. Grundlegende Struktur

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

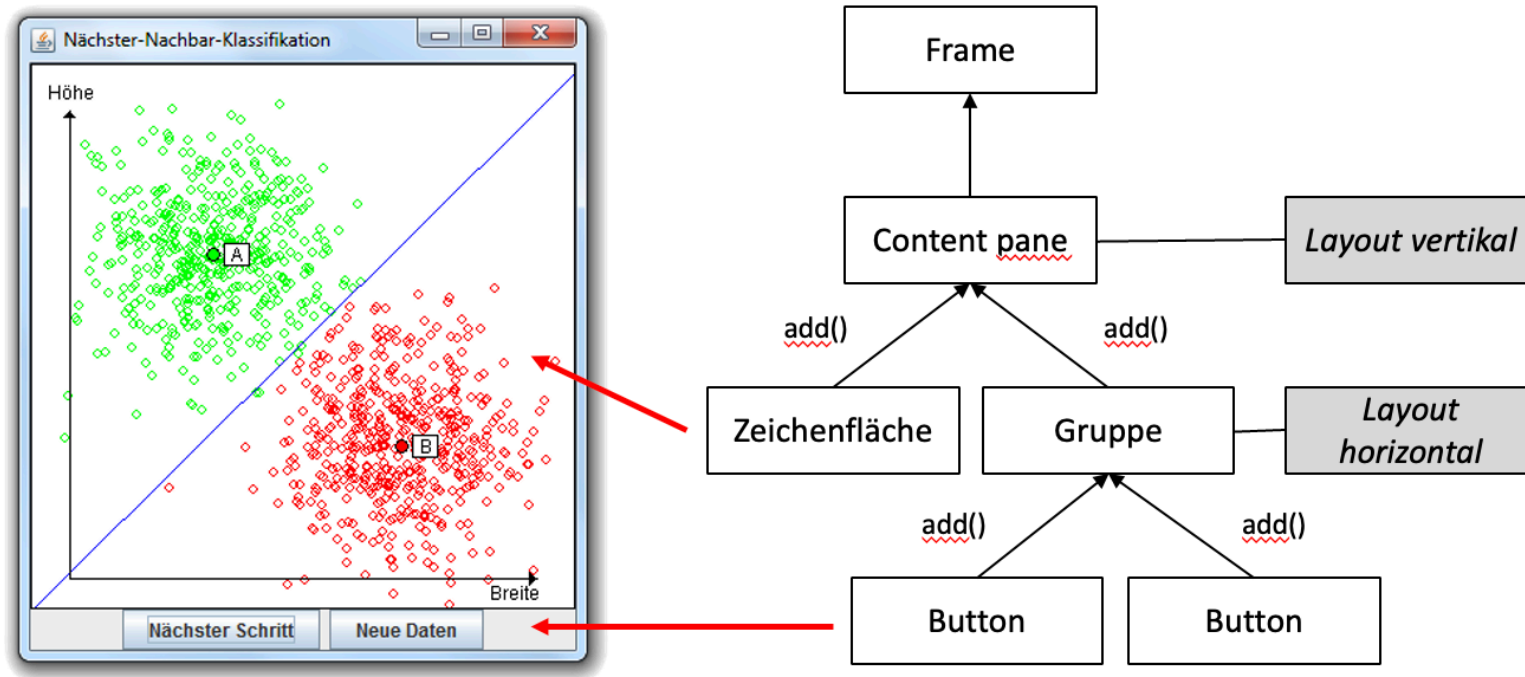


Abbildung 3: Hierarchie eines Windows

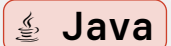
3. Graphische Oberflächen erzeugen

3.1 Einfaches Programm

3. Graphische Oberflächen erzeugen

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

```
1  public class HelloWorld {  
2      public HelloWorld() {  
3          JFrame frame = new JFrame("GUI example");  
4          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
5          frame.setVisible(true);  
6      }  
7  
8      public static void main(String[] args) {  
9          new HelloWorld();  
10     }  
11 }
```



? 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

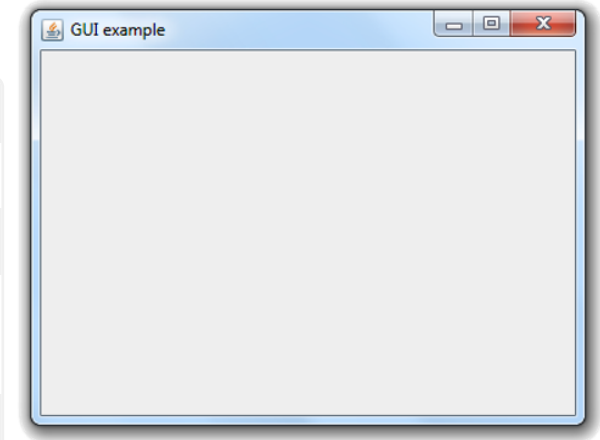
- Vergrößern Sie es auf 400 x 300 Pixel (Breite x Höhe).
- Platzieren Sie es jeweils 50 Pixel vom linken und vom oberen Rand.
- Hinweis: Lassen Sie sich die Methoden von frame anzeigen.

3.1 Einfaches Programm

3. Graphische Oberflächen erzeugen

- Korrigierte Größe und Position:

```
1  public class HelloWorld {
2      public HelloWorld() {
3          JFrame frame = new JFrame("GUI example");
4      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
5          frame.setSize(400, 300);
6          frame.setLocation(50, 50);
7          frame.setVisible(true);
8      }
9
10     public static void main(String[] args) {
11         new HelloWorld();
12     }
13 }
```



3.1 Einfaches Programm

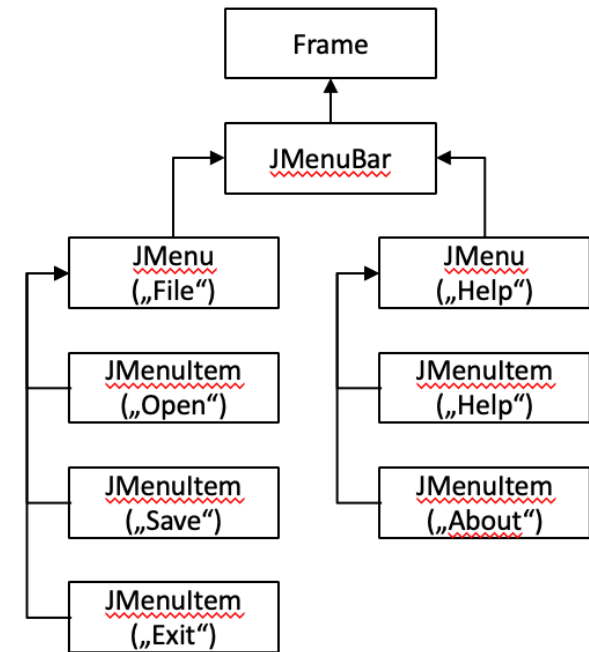
3. Graphische Oberflächen erzeugen

- 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.

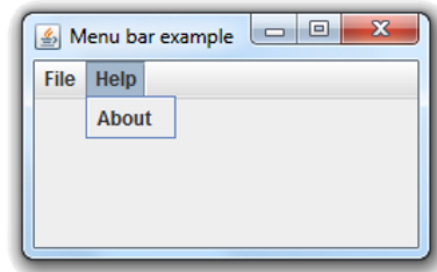
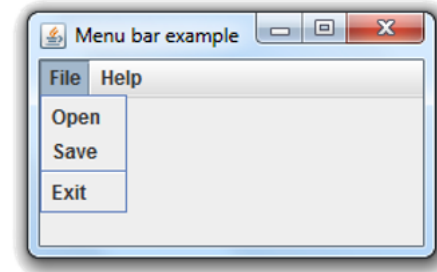


3.1 Einfaches Programm

3. Graphische Oberflächen erzeugen

```
1  public MenuBar() {
2      JFrame frame = new JFrame("Menu bar example");
3      // Set frame properties ...
4
5      JMenuBar menuBar = new JMenuBar(); // Create menu bar and add to frame
6      frame.setJMenuBar(menuBar);
7
8      JMenu menuFile = new JMenu("File"); // Create menu "File"
9      menuBar.add(menuFile);
10     menuFile.add(new JMenuItem("Open"));
11     menuFile.add(new JMenuItem("Save"));
12     menuFile.addSeparator();
13     menuFile.add(new JMenuItem("Exit"));
14
15     JMenu menuHelp = new JMenu("Help"); // Create menu "Help"
16     menuBar.add(menuHelp);
17     menuHelp.add(new JMenuItem("About"));
18
19     frame.setVisible(true);
20 }
```

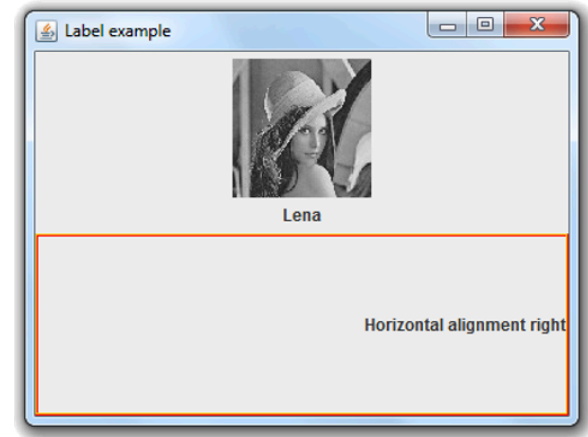
 Java



3.1 Einfaches Programm

3. Graphische Oberflächen erzeugen

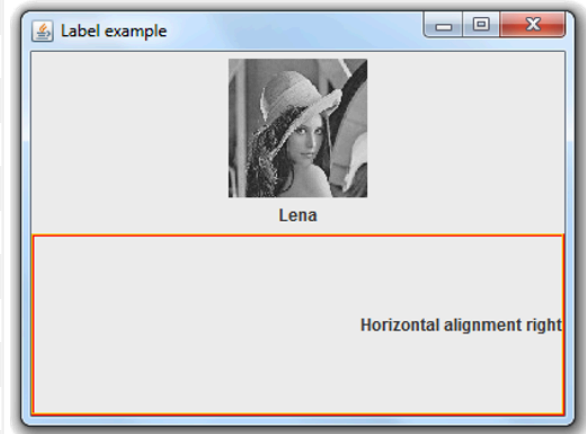
- Klasse JLabel stellt nicht editierbaren Text dar
 - Kann horizontal und vertikal ausgerichtet werden (z.B. zentriert)
 - Kann Rahmen zeichnen
 - Kann auch Bilder darstellen
- Erzeugen wir das rechts abgebildete Fenster:
 - Bild über `new ImageIcon()` laden
 - Rahmen über `BorderFactory.createEtchedBorder()`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



3.1 Einfaches Programm

3. Graphische Oberflächen erzeugen

```
1 // Create frame and set properties
2 JFrame frame = new JFrame("Label example");
3 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
4 frame.setSize(400, 300);
5 frame.setLocation(50, 50);
6 frame.setLayout(new GridLayout(2, 1)); // 2 rows, 1 column
7
8 // Create labels
9 ImageIcon image = new ImageIcon("folien07_gui/Lena100.jpg");
10 JLabel label1 = new JLabel("Lena", image, JLabel.CENTER);
11 label1.setHorizontalTextPosition(JLabel.CENTER);
12 label1.setVerticalTextPosition(JLabel.BOTTOM);
13
14 JLabel label2 = new JLabel("Horizontal alignment right");
15 label2.setHorizontalAlignment(JLabel.RIGHT);
16 label2.setBorder(BorderFactory.createEtchedBorder(Color.RED, Color.ORANGE));
17
18 // Add labels to content pane
19 Container contentPane = frame.getContentPane();
20 contentPane.add(label1);
21 contentPane.add(label2);
22
23 frame.setVisible(true);
```



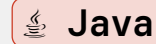
4. Layout

4.1 Layout-Manager

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

4.1 Layout-Manager

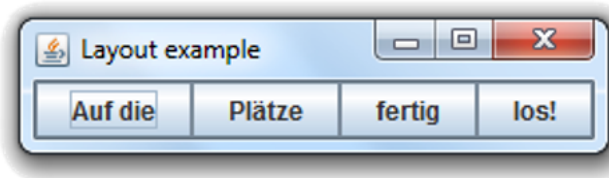
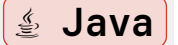
```
1 // Create frame and set properties
2 JFrame frame = new JFrame("Layout example");
3 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
4 frame.setLocation(50, 50);
5
6 // Create contents
7 Container contentPane = frame.getContentPane();
8 contentPane.setLayout(new BoxLayout(contentPane,
9   BoxLayout.Y_AXIS));
10 contentPane.add(new JButton("Auf die"));
11 contentPane.add(new JButton("Plätze"));
12 contentPane.add(new JButton("fertig"));
13 contentPane.add(new JButton("los!"));
14
15 frame.pack();
16 frame.setVisible(true);
```



4.1 Layout-Manager

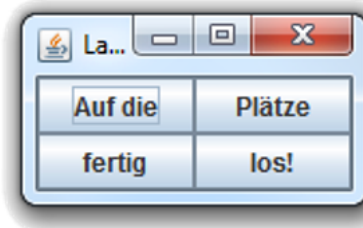
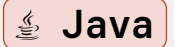
- Horizontales BorderLayout:

```
1 contentPane.setLayout(new BorderLayout(contentPane, BorderLayout.X_AXIS));
```



- GridLayout:

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

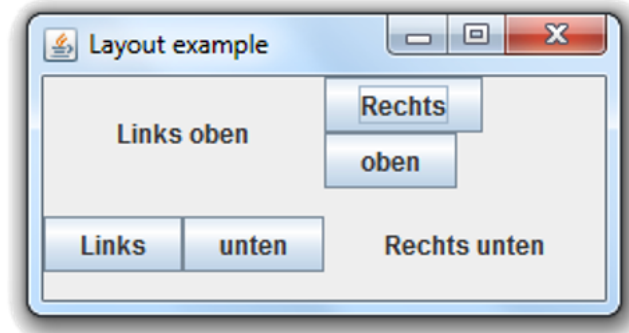


4.1 Layout-Manager

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

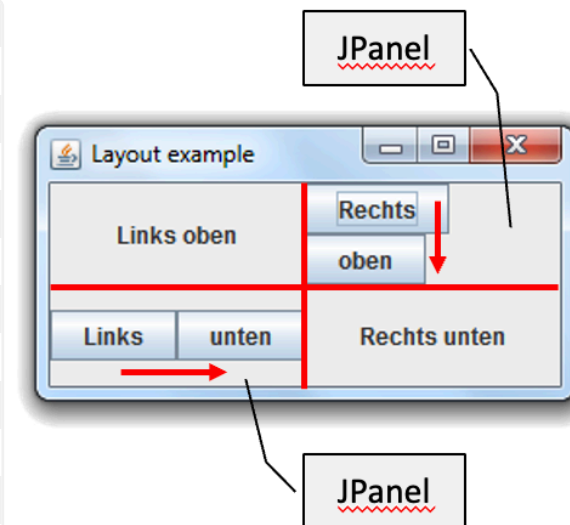
? Frage

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



4.1 Layout-Manager

```
1  JFrame frame = new JFrame("Layout example");
2
3  JPanel panel1 = new JPanel();
4  panel1.setLayout(new BorderLayout(panel1, BorderLayout.Y_AXIS));
5  panel1.add(new JButton("Rechts"));
6  panel1.add(new JButton("oben"));
7
8  JPanel panel2 = new JPanel();
9  panel2.setLayout(new BorderLayout(panel2, BorderLayout.X_AXIS));
10 panel2.add(new JButton("Links"));
11 panel2.add(new JButton("unten"));
12
13 Container contentPane = frame.getContentPane();
14 contentPane.setLayout(new GridLayout(2, 2));
15 contentPane.add(new JLabel("Links oben", JLabel.CENTER));
16 contentPane.add(panel1);
17 contentPane.add(panel2);
18 contentPane.add(new JLabel("Rechts unten", JLabel.CENTER));
19
20 frame.pack();
21 frame.setVisible(true);
```



5. Zeichnen

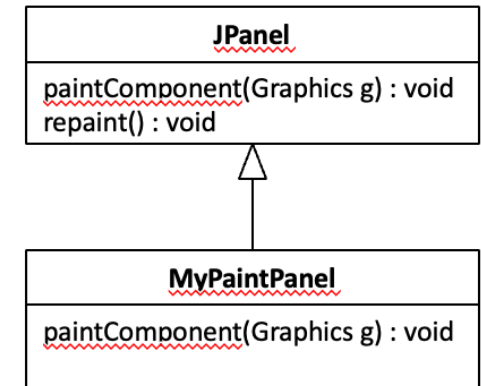
5.1 Zeichnen

- 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.

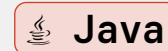
<u>JPanel</u>
<u>paintComponent(Graphics g) : void</u> <u>repaint() : void</u>

5.1 Zeichnen

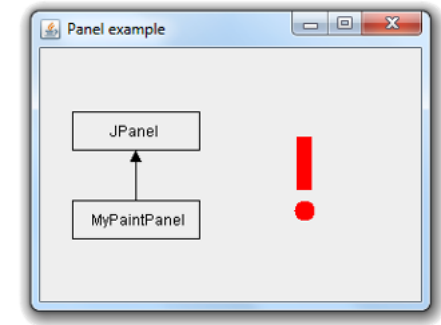
- Okay, es gibt also JPanel mit der Methode `paintComponent()`.
- Es wird gezeichnet, was in `paintComponent()` steht.
- Aber wie kann man Zeichenbefehle zu dieser Methode hinzufügen?!
- Lösung:
 - Leiten Sie von `JPanel` ab und überlagern Sie `paintComponent()`.
 - Hierdurch: Panel-Klasse mit frei definierbarer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



```
1  class MyPaintPanel extends JPanel {
2      public void paintComponent(Graphics g) {
3          super.paintComponent(g);
4          // Code for own drawings ...
5      }
6  }
```

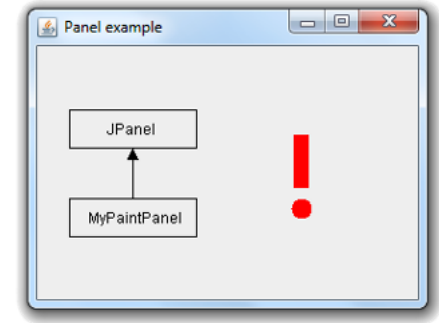


```
1  class MyPaintPanel extends JPanel {
2      public Dimension getPreferredSize() {
3          return new Dimension(300, 200);
4      }
5
6      public void paintComponent(Graphics g) {
7          super.paintComponent(g);
8
9          g.setColor(Color.BLACK);
10         g.drawRect(25, 50, 100, 30); // Super class
11         g.drawString("JPanel", 55, 70);
12         g.drawRect(25, 120, 100, 30); // Sub class
13         g.drawString("MyPaintPanel", 40, 140);
14         g.drawLine(75, 80, 75, 120); // Arrow
15         g.fillPolygon(new int[]{70, 75, 80}, new int[]{90, 80, 90}, 3);
16
17         g.setColor(Color.RED);
18         g.fillRect(202, 70, 12, 42);
19         g.fillOval(200, 120, 16, 16);
20     }
21 }
```



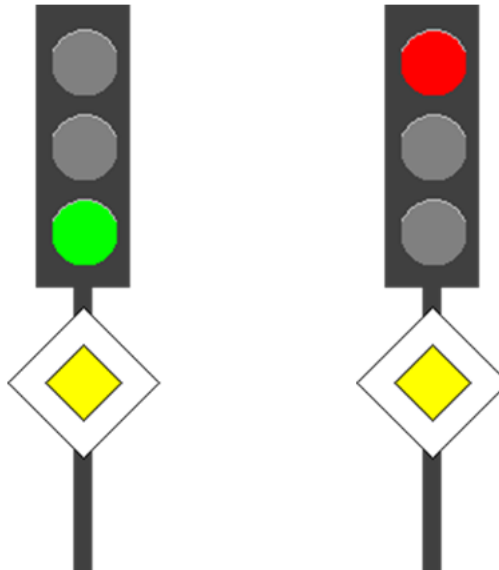
- Einbinden in graphische Oberfläche:

```
1  public class PaintPanel {
2      public PaintPanel() {
3          JFrame frame = new JFrame("Panel example");
4          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
5          frame.setLocation(50, 50);
6
7          frame.add(new MyPaintPanel());
8          frame.pack();
9          frame.setVisible(true);
10     }
11
12     public static void main(String[] args) {
13         new PaintPanel();
14     }
15 }
```



☰ Aufgabe 2

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



6. Buttons & Ereignisse

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

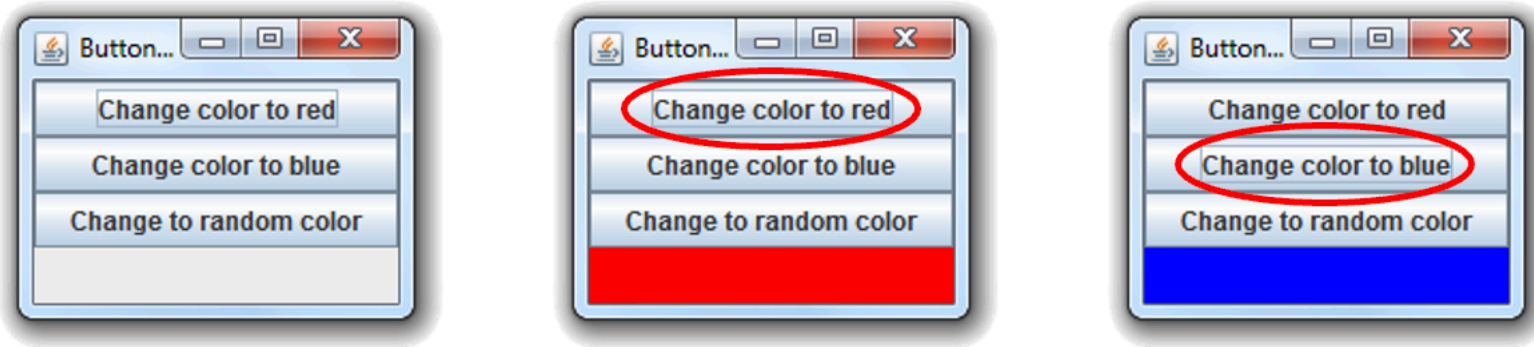
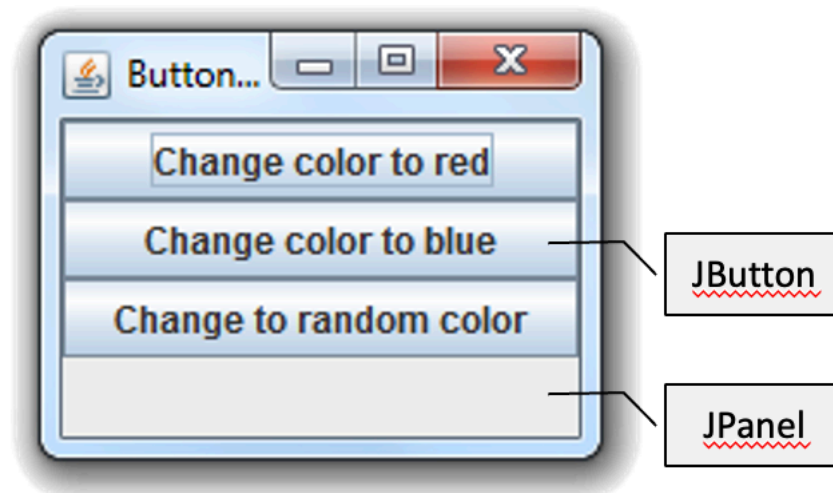


Abbildung 21: Buttons, die eine Farbe ändern

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

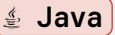
☰ Aufgabe 3

- Erzeugen Sie zunächst die GUI mit ihren Elementen.



- Erzeugen von Elementen der Klasse JButton:

```
1  public class ButtonEvent {
2      public ButtonEvent() {
3          JFrame frame = new JFrame("Button example");
4          // Set frame properties ...
5
6          // Create and layout contents
7          frame.setLayout(new GridLayout(4, 1)); // 4 rows, 1 column
8          Container contentPane = frame.getContentPane();
9          contentPane.add(new JButton("Change color to red"));
10         contentPane.add(new JButton("Change color to blue"));
11         contentPane.add(new JButton("Change to random color"));
12         contentPane.add(new JPanel());
13         frame.pack();
14         frame.setVisible(true);
15     }
16
17     public static void main(String[] args) {
18         new ButtonEvent();
19     }
20 }
```

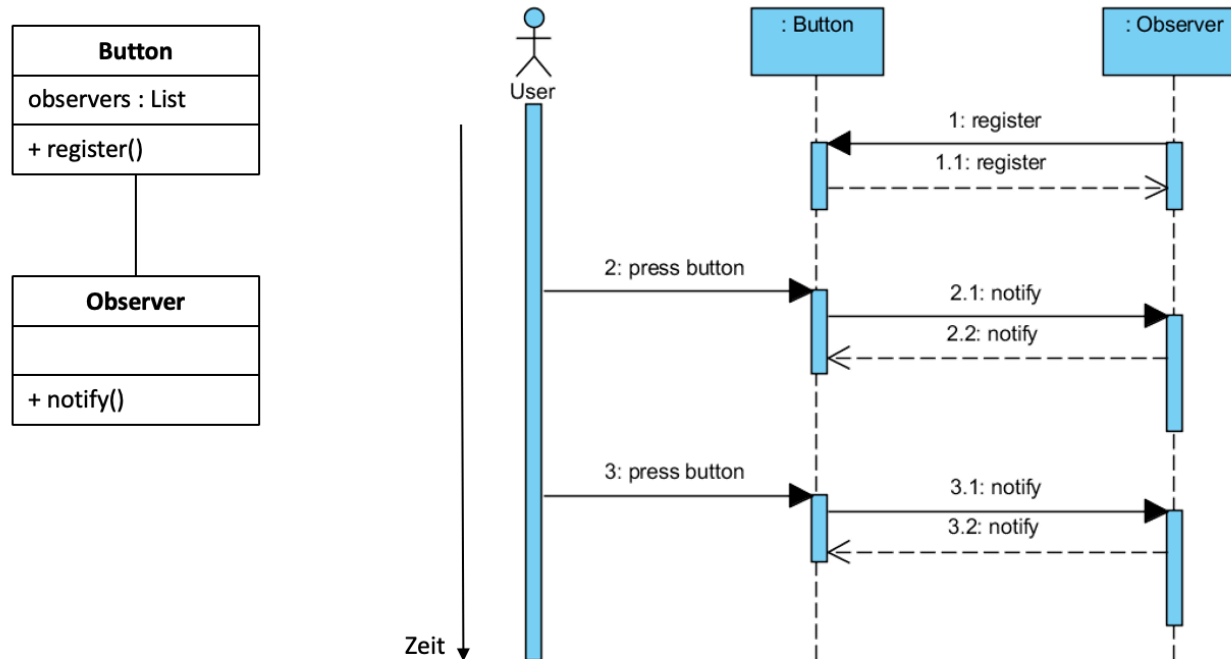


- Wie können wir aber darauf reagieren, wenn ein Button gedrückt wird?
- Beteiligte Objekte:
 - Button mit Zustand (z.B. „nicht gedrückt“, „gedrückt“)
 - Objekt, das bei Änderungen des Buttons benachrichtigt werden soll
- Prinzipielles Vorgehen:
 - Registrierung:
 - Objekt „sagt dem Button“, dass es 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

6. Buttons & Ereignisse

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



- 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

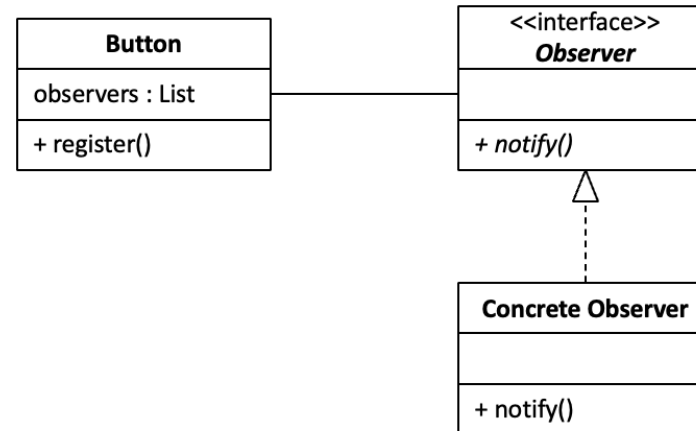
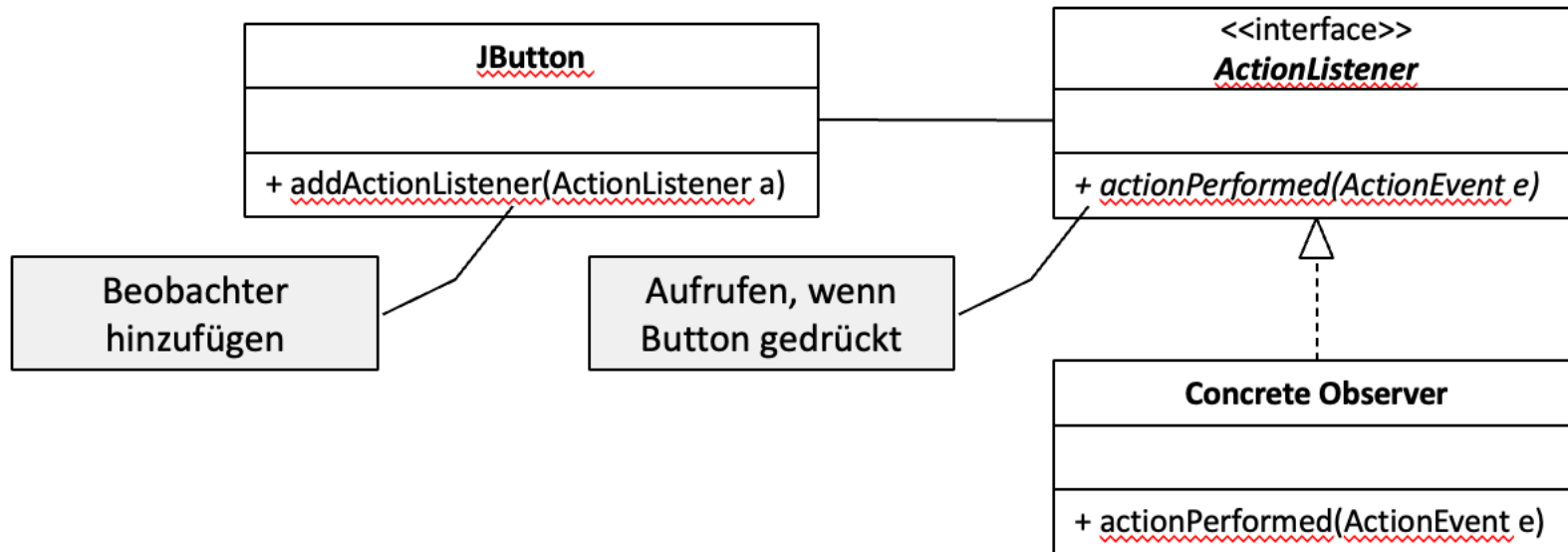


Abbildung 24: Interface Observer

6.1 Aufgabe

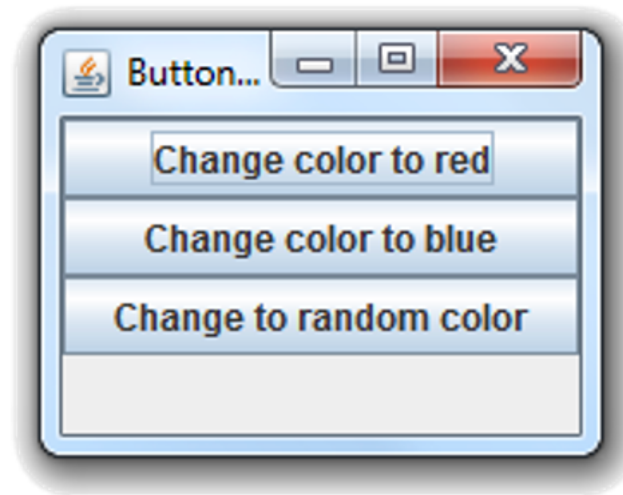
6. Buttons & Ereignisse

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



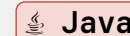
☰ Aufgabe 4

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



- Auszug aus Quelltext:

```
1  public class ButtonEvent implements ActionListener {
2      private JPanel panel;
3      private JButton buttonRed, buttonBlue, buttonRandom;
4
5      public ButtonEvent() {
6          // ...
7
8          // Buttons with event handling
9          buttonRed = new JButton("Change color to red");
10         buttonBlue = new JButton("Change color to blue");
11         buttonRandom = new JButton("Change to random color");
12
13         buttonRed.addActionListener(this);
14         buttonBlue.addActionListener(this);
15         buttonRandom.addActionListener(this);
16
17         // ...
18     }
19 }
```



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

```
1  public class ButtonEvent implements ActionListener {
2      public void actionPerformed(ActionEvent event) {
3          if (event.getSource() == buttonRed) {
4              panel.setBackground(Color.RED);
5          } else if (event.getSource() == buttonBlue) {
6              panel.setBackground(Color.BLUE);
7          } else if (event.getSource() == buttonRandom) {
8              Random random = new Random();
9              float red = random.nextFloat();
10             float green = random.nextFloat();
11             float blue = random.nextFloat();
12             Color color = new Color(red, green, blue);
13             panel.setBackground(color);
14         }
15     }
16 }
```




6.1 Aufgabe

6. Buttons & Ereignisse


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

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

 Java

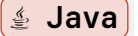
- String in Methode actionPerformed() abfragen und nutzen:

```
1  public void actionPerformed(ActionEvent event) {
2      String actionCommand = event.getActionCommand();
3
4      if (actionCommand.equals("Change color to red")) {
5          // ...
6      } else if (actionCommand.equals("Change color to blue")) {
7          // ...
8      } else if (actionCommand.equals("Change to random color")) {
9          // ...
10     }
11 }
```

 Java

- Neuen ActionListener mit Methode actionPerformed() inline definieren

```
1  public class ButtonEvent2 {
2      private JPanel panel;
3
4      public ButtonEvent2() {
5          // ...
6
7          buttonRed.addActionListener(new ActionListener() {
8              public void actionPerformed(ActionEvent event) {
9                  panel.setBackground(Color.RED);
10             }
11         });
12
13         // ...
14     }
15
16     public static void main(String[] args) {
17         new ButtonEvent2();
18     }
19 }
```



- 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 Mausekkrads über der Komponente

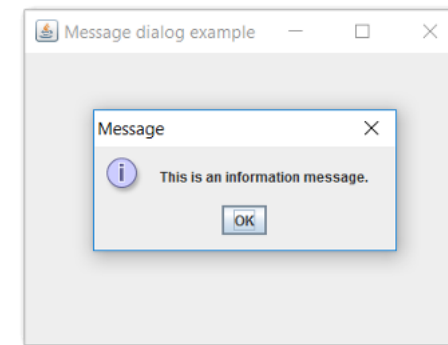
7. Einfache Dialoge

7.1 Einfache Dialoge

7. Einfache Dialoge

- Beispiele für Dialoge über JOptionPane:

```
1 public class MessageDialogs {
2     public MessageDialogs() {
3         // Create and show frame
4         JFrame frame = new JFrame("Message dialog example");
5         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6         frame.setSize(400, 300);
7         frame.setLocationByPlatform(true);
8         frame.setVisible(true);
9
10        // Display dialogs
11        JOptionPane.showMessageDialog(frame, "This is a plain message.",
12            "Message",
13            JOptionPane.PLAIN_MESSAGE);
14        JOptionPane.showMessageDialog(frame, "This is an information
15            message.", "Message",
16            JOptionPane.INFORMATION_MESSAGE);
17        JOptionPane.showMessageDialog(frame, "This is a warning.", "Message",
18            JOptionPane.WARNING_MESSAGE);
19    }
20 }
```



7.1 Einfache Dialoge

7. 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 `TextField`, `PasswordField` und `TextArea`
 - Auswahlboxen über `CheckBox`
 - Listen über `ComboBox` und `List`
 - Tooltips über Methode `setToolTipText()`
 - Datei auswählen über `FileChooser`

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.