Object-Oriented Programming in Java

Lecture 7 - Graphical User Interfaces

Emily Lucia Antosch

HAW Hamburg

14.08.2025

Contents

1. Introduction

1.1 Where Are We Currently?

- The last lecture was about interfaces and abstract classes
- You can now
  - use abstract classes to structure your code more precisely,
  - implement interfaces to represent properties of classes,
  - assign classes and objects with an order using `Comparable`,
- Today we continue with **graphical user interfaces**.

1.1 Where Are We Currently?

1. Imperative Concepts
2. Classes and Objects
3. Class Library
4. Inheritance
5. Interfaces
6. **Graphical User Interfaces**
7. Exception Handling
8. Input and Output
9. Multithreading (Parallel Computing)

1.2 The Goal of This Chapter

- You create graphical user interfaces with e.g. menus, buttons and text fields.
- You draw diagrams from simple geometric shapes (e.g. lines, circles).
- You respond to events (e.g. pressing a button) by connecting graphical elements with methods to be executed on user input.
- You use the Observer pattern so that objects of any data type can react to events.

2. Basic Structure

2.1 Graphical UI

? Question

- What types of elements do you see?
- How do the elements react? Do elements interact with each other?

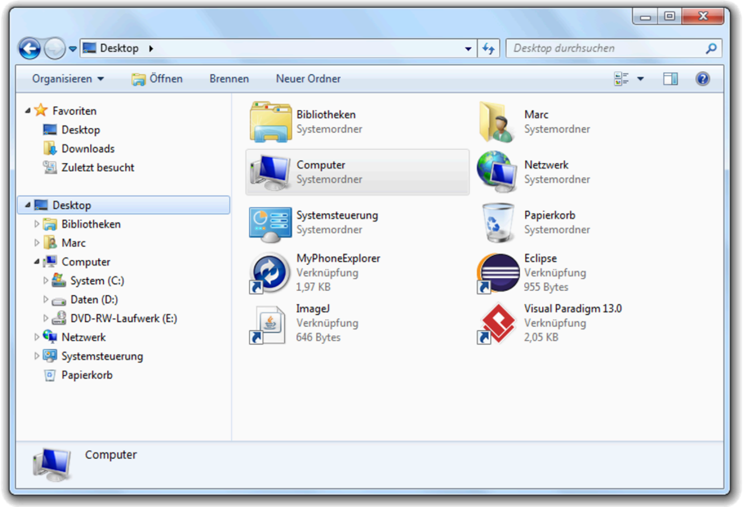


Figure 1: Windows 7 Explorer

2.1 Graphical UI

- Graphical user interface: Graphical user interface (GUI)
- Class libraries AWT and Swing already included in the Java SDK
- Abstract Window Toolkit (AWT):
  - Already introduced with Java 1.0
  - Only basic interface elements to support as many operating systems as possible ("Lowest common denominator")
  - Uses the native elements ("widgets") of the operating system
  - Originally full of design errors, as it was created under great pressure in just under two months
- Swing:
  - Extension of AWT
  - No more direct addressing of window functions of the current platform
  - Complete control over display elements

2.1 Graphical UI

- Base element: Frame
- Contains window bar with title and control elements (e.g. “Close”)
- Contains area where elements can be placed (Content pane)
- Can additionally contain menu bar

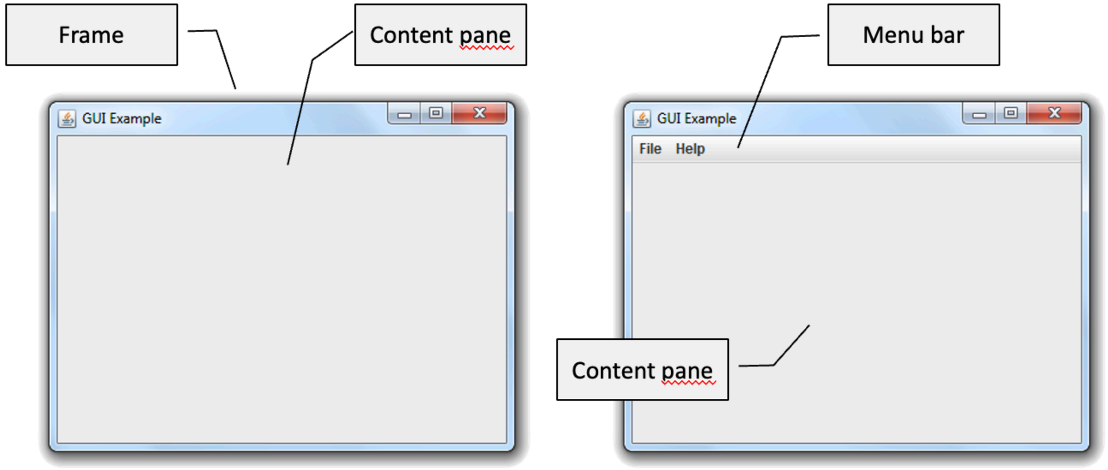


Figure 2: Structure of a frame

2.1 Graphical UI

- Elements are added hierarchically.
- For elements that contain other elements, the layout can be specified.

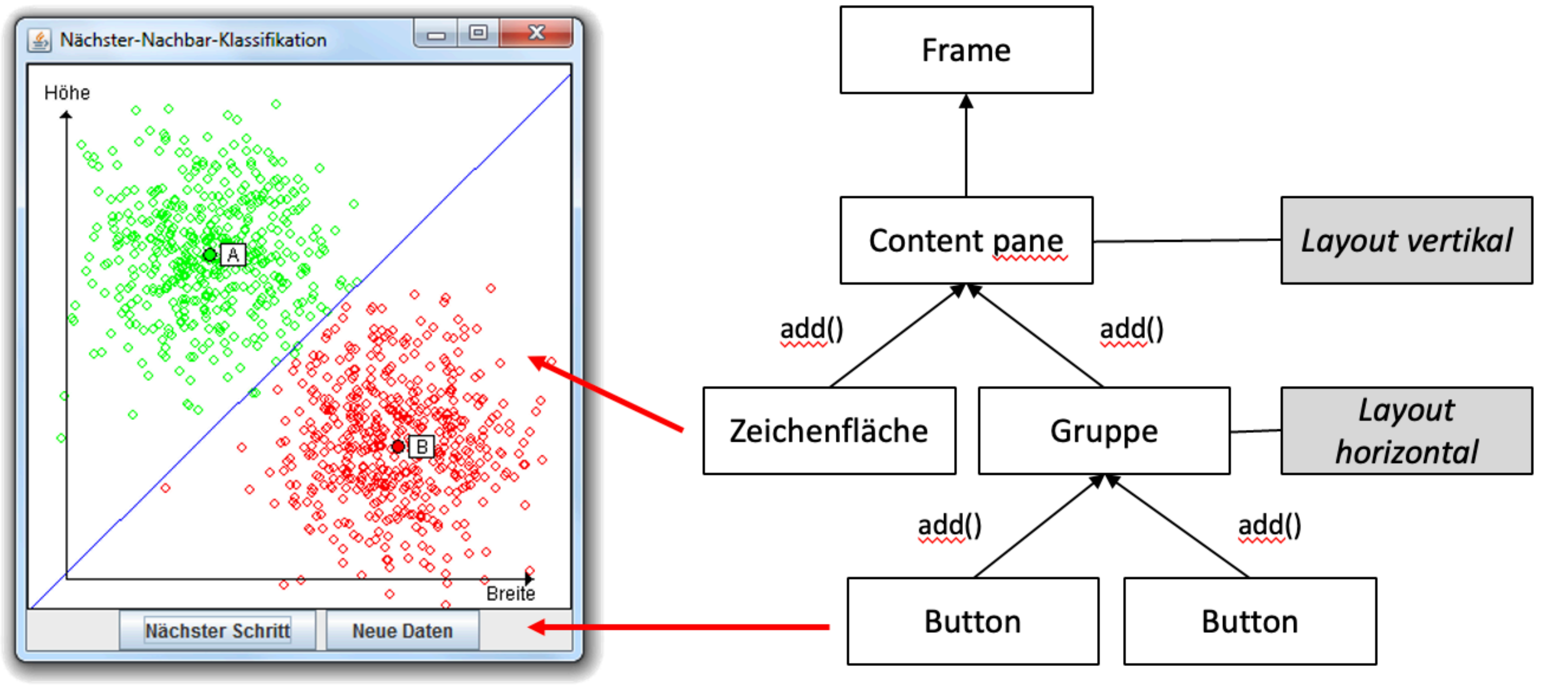


Figure 3: Hierarchy of a window

3. Creating Graphical User Interfaces

3.1 Simple Program

- Executable main() method creates object of the class
- Class creates frame with graphical interface in constructor
- Specify “Close Operation” so that application terminates when window is closed

Java

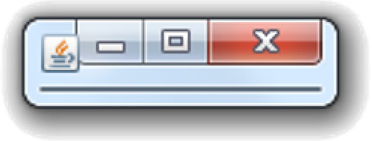
```java
public class HelloWorld {
    public HelloWorld() {
        JFrame frame = new JFrame("GUI example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

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

3.1 Simple Program

Question

- What happens if the “Close Operation” is not set to “Exit on close”?
- Why do you have to explicitly display the window via setVisible(true)?

- And it doesn’t look really nice:
  - The window is too small!
  - The window “sticks” in the upper left corner!



Task 1

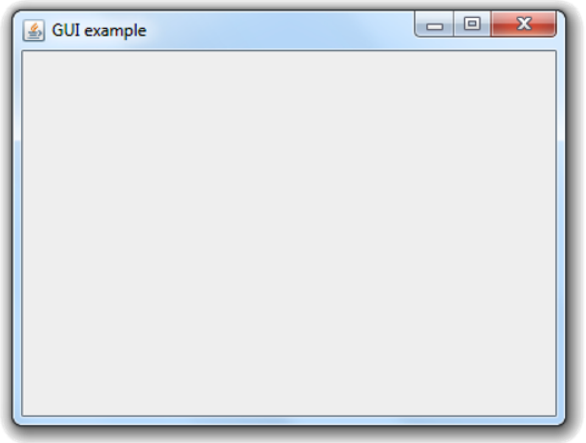
- Enlarge it to 400 x 300 pixels (width x height).
- Place it 50 pixels from the left and top edge respectively.
- Hint: Display the methods of frame.

3.1 Simple Program

- Corrected size and position:

Java

```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();
	}
}
```

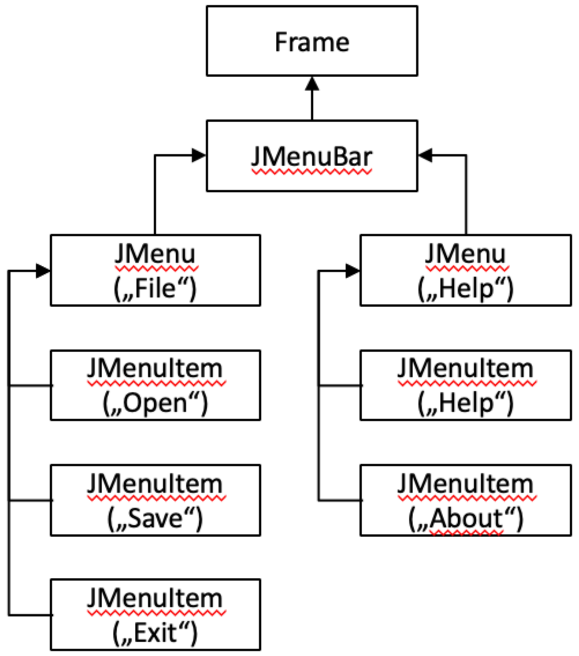


3.1 Simple Program

- Klassen:
  - JMenuBar: Menu bar
  - JMenu: Menu in menu bar (e.g. File, Help)
  - JMenuItem: Entry in a menu (e.g. New, Save as)
- Add the following menus to our program:
  - Menu File with menu items Open, Save and Exit
  - Menu Help with menu items Help and About

🔥 **Tip**

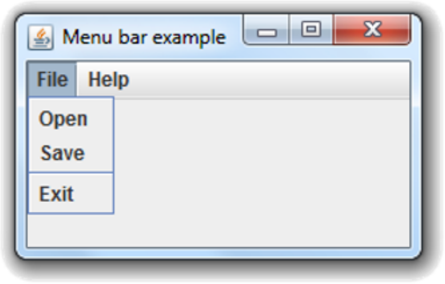
- Elements are usually added via add().
- The menu bar is added via setJMenuBar().

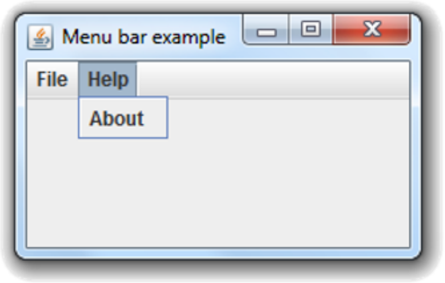


3.1 Simple Program

Java

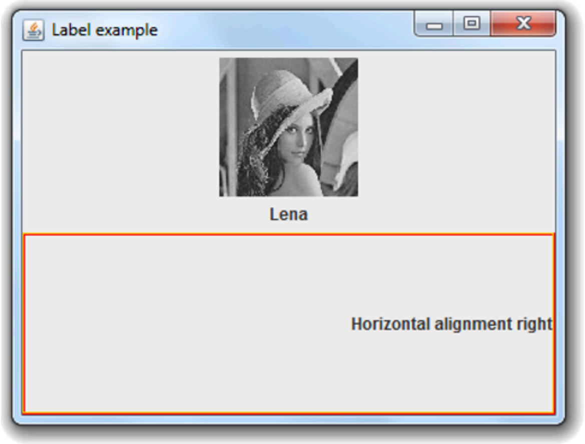
```java
public MenuBar() {
	JFrame frame = new JFrame("Menu bar example");
	// Set frame properties ...

	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 Simple Program

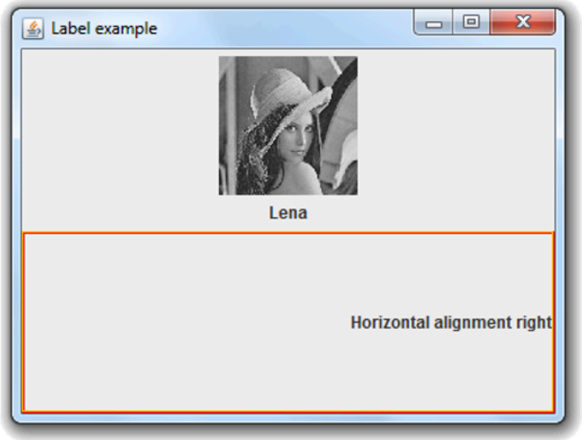
- Class JLabel displays non-editable text
  - Can be aligned horizontally and vertically (e.g. centered)
  - Can draw borders
  - Can also display images
- Let’s create the window shown on the right:
  - Load image via new ImageIcon()
  - Border via BorderFactory.createEtchedBorder()
  - Add label to content pane via add()
  - Layout via frame.setLayout(new GridLayout(2, 1))



3.1 Simple Program

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);
```



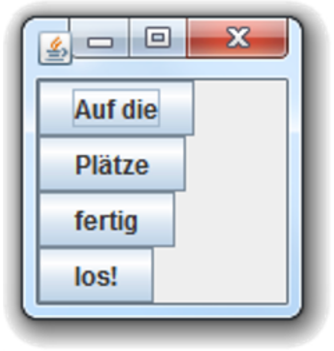
4. Layout

4.1 Layout-Manager

- Define the arrangement of GUI elements
- Various layout managers defined, e.g.:
  - BoxLayout:
    - Elements on top of each other (“vertical”) or next to each other (“horizontal”)
  - GridLayout:
    - Elements placed in uniform grid
    - All cells have the same size
  - FlowLayout:
    - Elements placed in row like horizontal BoxLayout
    - However, line break as soon as a line is “full”

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("Ready"));
contentPane.add(new JButton("Set"));
contentPane.add(new JButton("Go"));
contentPane.add(new JButton("los!"));

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

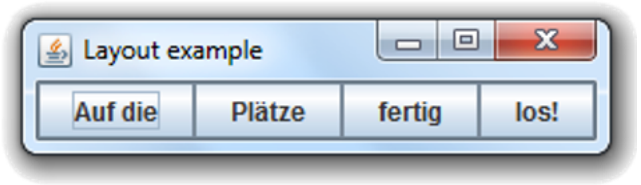


4.1 Layout-Manager

- Horizontal BoxLayout:

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

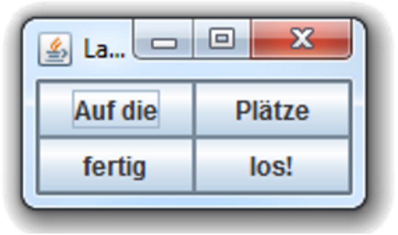
Java



- GridLayout:

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

Java

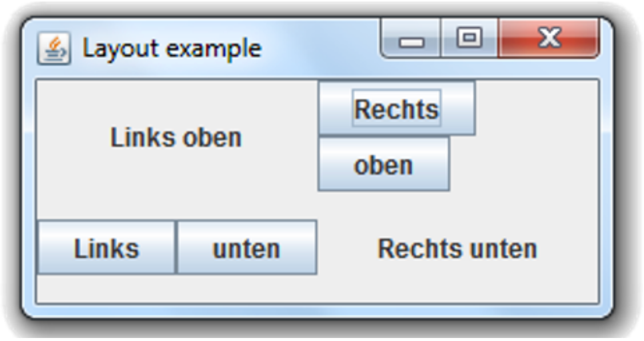


4.1 Layout-Manager

- Elements can be grouped in objects of class JPanel.
- Each JPanel object has its own layout manager.

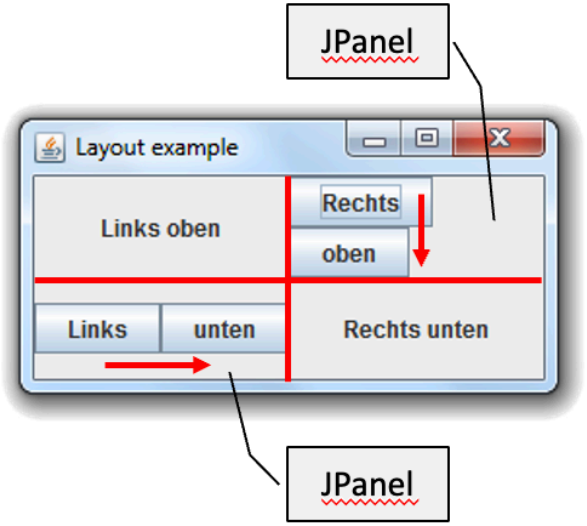
? Question

- Which elements does the shown window contain?
- Via which objects and layout managers are these arranged?



4.1 Layout-Manager

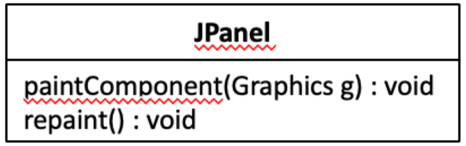
```java
JFrame frame = new JFrame("Layout example");

JPanel panel1 = new JPanel();
panel1.setLayout(new BoxLayout(panel1, BoxLayout.Y_AXIS));
panel1.add(new JButton("Right"));
panel1.add(new JButton("top"));

JPanel panel2 = new JPanel();
panel2.setLayout(new BoxLayout(panel2, BoxLayout.X_AXIS));
panel2.add(new JButton("Left"));
panel2.add(new JButton("bottom"));

Container contentPane = frame.getContentPane();
contentPane.setLayout(new GridLayout(2, 2));
contentPane.add(new JLabel("Top left", JLabel.CENTER));
contentPane.add(panel1);
contentPane.add(panel2);
contentPane.add(new JLabel("Bottom right", JLabel.CENTER));

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



5. Zeichnen

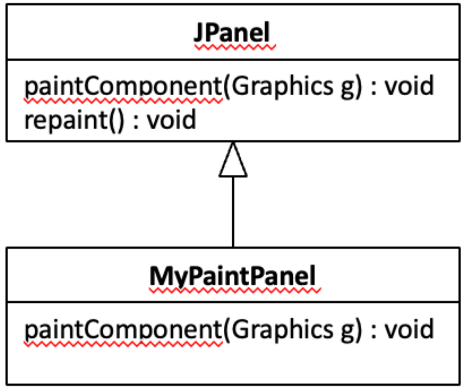
5.1 Zeichnen

- Class JPanel as drawing surface:
  - Can draw freely on panel.
- Drawing method:
  - System executes paintComponent() method for drawing
  - Is automatically called when window changes
  - Method receives parameter of type Graphics (graphics context)
  - Graphics has methods for drawing (texts, lines, rectangles, arcs, …)
- Explicit redrawing:
  - Redrawing can also be initiated via repaint() method.
  - This internally calls paintComponent().



5.1 Zeichnen

- Okay, so there is JPanel with the paintComponent() method.
- What is drawn is what is in paintComponent().
- But how can you add drawing commands to this method?!
- Solution:
  - Derive from JPanel and override paintComponent().
  - This results in: Panel class with freely definable drawing method

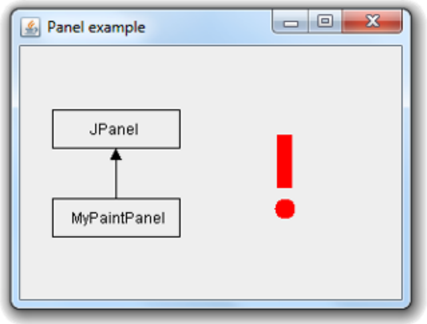

JPanel
paintComponent(Graphics g) : void
repaint() : void
MyPaintPanel
paintComponent(Graphics g) : void

Java

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

5.1 Zeichnen

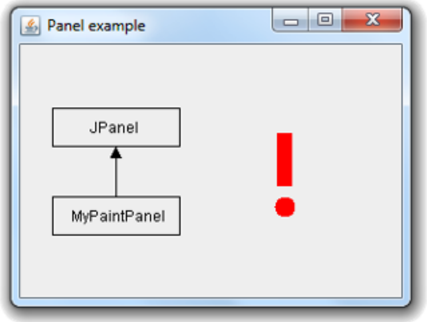
```java
class MyPaintPanel extends JPanel {
    public Dimension getPreferredSize() {
        return new Dimension(300, 200);
    }

    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

- Integration into graphical interface:

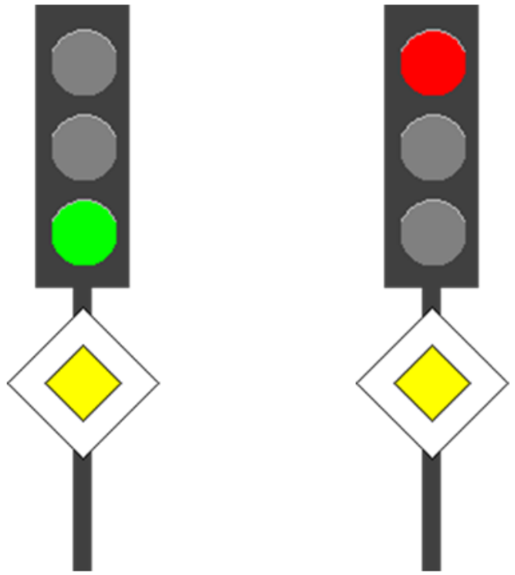
```java
public class PaintPanel {
        public PaintPanel() {
            JFrame frame = new JFrame("Panel example");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setLocation(50, 50);

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

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



5.1 Zeichnen

Task 2

- Discover your artistic streak!
- Create a program that displays a traffic light.



6. Buttons & Events

6.1 Task

- Our goal is the following application:
  - Window with three buttons and one panel
  - Selection of buttons color the panel red, blue or in random color

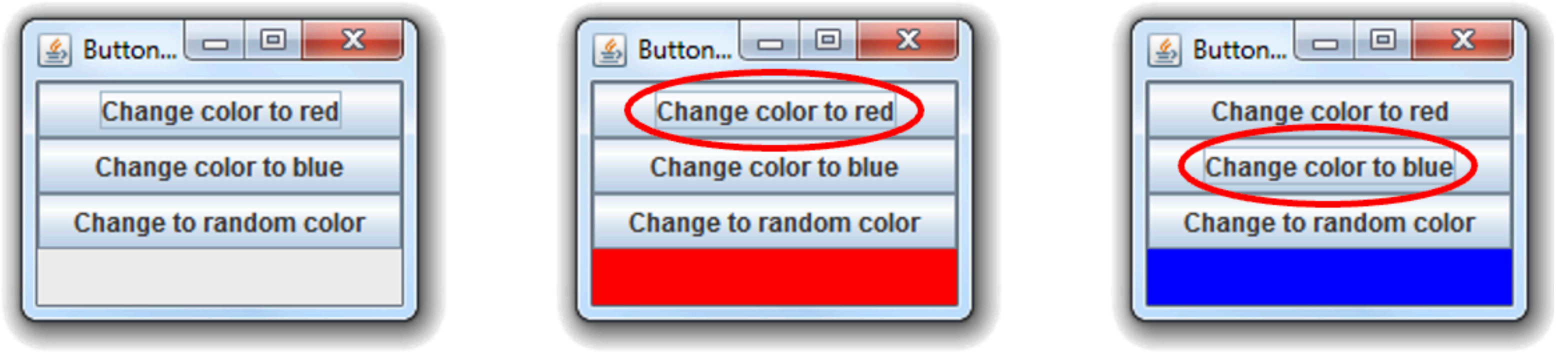


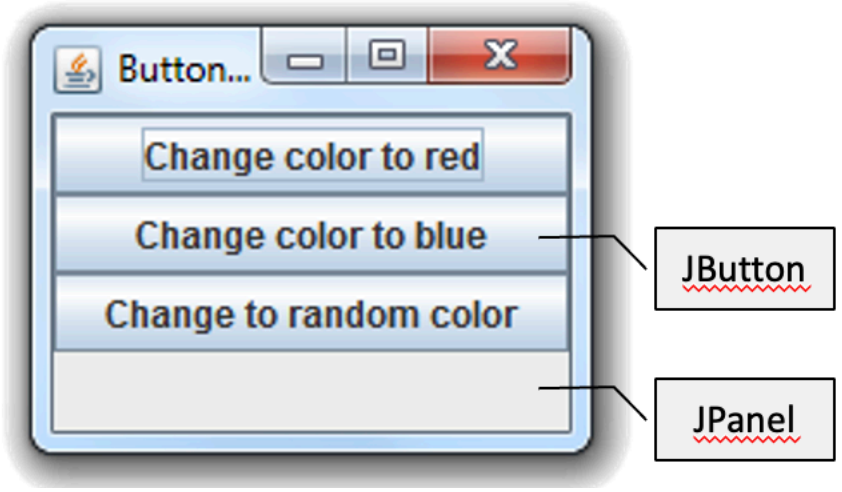
Figure 21: Buttons that change a color

- We need for this:
  - Buttons as elements
  - Possibility to react to pressed button

6.1 Task

Task 3

- First create the GUI with its elements.



6.1 Task

- Creating elements of class JButton:

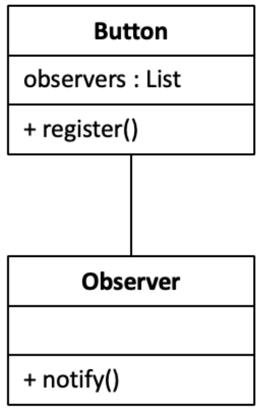
```java
public class ButtonEvent {
    public ButtonEvent() {
        JFrame frame = new JFrame("Button example");
        // Set frame properties ...

        // 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();
    }
}
```

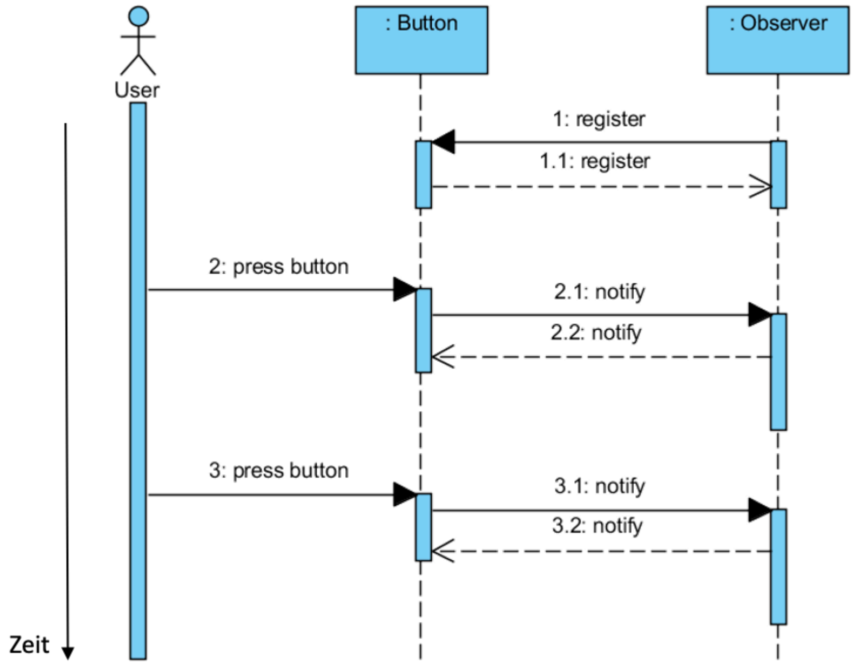
6.1 Task

- But how can we react when a button is pressed?
- Involved objects:
  - Button with state (e.g. “not pressed”, “pressed”)
  - Object that should be notified when the button changes
- Basic approach:
  - Registration:
    - Object “tells the button” that it wants to be notified of changes
    - Button remembers (e.g. in list) which objects should be notified
  - Button is pressed:
    - Button notifies objects in the list that its state has changed

6.1 Task

- Possible implementation:
  - Button: Method register() to add observers to the list
  - Observer: Method notify() that button object calls for notification





6.1 Task

- Buuuuuut:
  - JButton cannot know classes we created.
  - Therefore cannot know if we implemented method notify().
- Solution:
  - Observers implement a defined interface
  - Button doesn’t need to know the observer’s class, only the interface

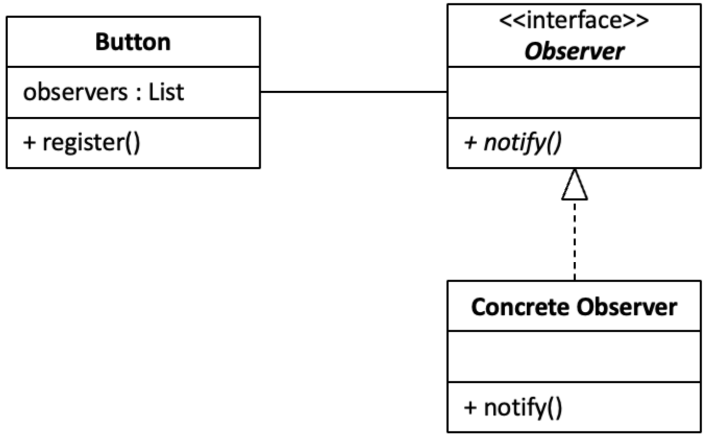


Figure 24: Interface `Observer`

6.1 Task

- Approach is also called Observer pattern
- More than one observer can register.
- In Swing, names of interface and methods chosen differently:

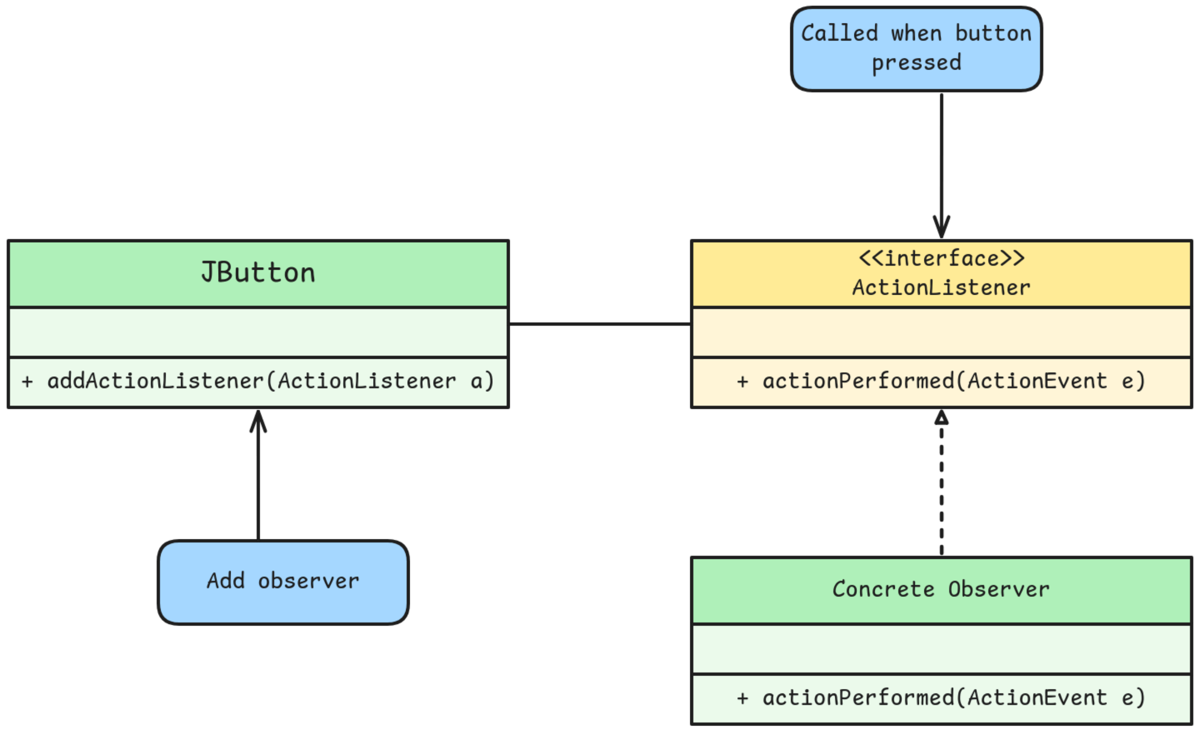
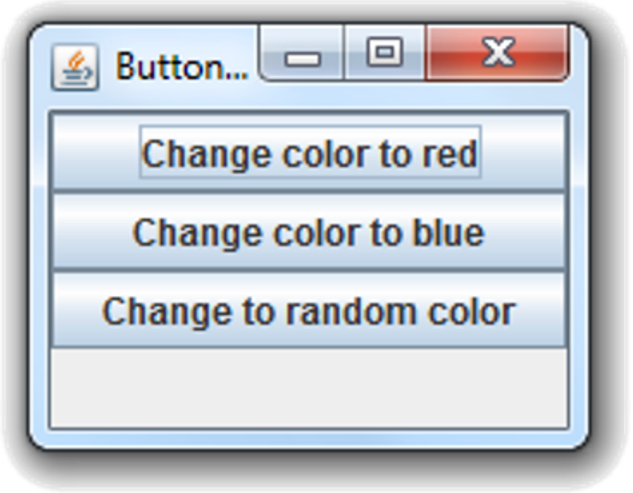

Called when button pressed
JButton
+ addActionListener(ActionListener a)
Add observer
<<interface>> ActionListener
+ actionPerformed(ActionEvent e)
Concrete Observer
+ actionPerformed(ActionEvent e)

Figure 25: Observer pattern

6.1 Task

Task 4

- Executable class implements interface ActionListener
- Object of executable class registers itself with the buttons



6.1 Task

- Excerpt from source code:

Java

```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);
        // ...
    }
}
```

6.1 Task

- Reaction to events (excerpt from source code):
  - Button identified via `getSource()` method of event object

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 Task

- Alternatively (find out in actionPerformed() which button was pressed):
- Connect buttons with a string, e.g.:

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

- Query and use string in `actionPerformed()` method:

```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 Task

- Define new `ActionListener` with `actionPerformed()` method inline

```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();
    }
}
```

6.1 Task

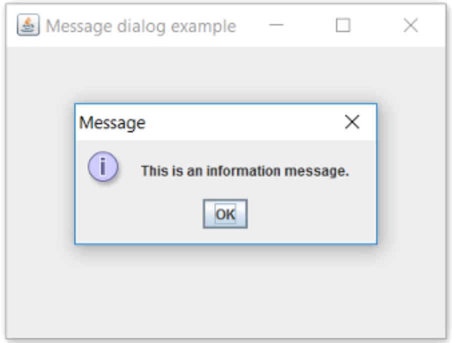
- All Swing components can register the following observers:
  - Component listener: Changes in size, position or visibility
  - Focus listener: Component gains or loses keyboard focus
  - Key listener: Keyboard events (only when component has keyboard focus)
  - Mouse listener: Mouse clicks, pressing, releasing and mouse movements
  - Mouse motion listener: Changes in cursor position over the component
  - Mouse wheel listener: Changes of mouse wheel over the component

7. Simple Dialogs

7.1 Simple Dialogs

- Examples for dialogs via 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);
    }

```



This is an information message.

7.1 Simple Dialogs

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

8. Suggestions

8.1 Ideas for Experimenting

- Some additional GUI elements:
  - Text fields via `JTextField`, `JPasswordField` and `JTextArea`
  - Selection boxes via `JCheckBox`
  - Lists via `JComboBox` and `JList`
  - Tooltips via method `setToolTipText()`
  - File selection via `JFileChooser`

9. License Notice

9.1 Attribution