Kostenloses eBook

LERNEN

swing

Free unaffiliated eBook created from
Stack Overflow contributors.

#swing

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swing

It is an unofficial and free swing ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official swing.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Swing

Bemerkungen

Swing wurde durch JavaFX ersetzt . Oracle empfiehlt im Allgemeinen die Entwicklung neuer Anwendungen mit JavaFX. Trotzdem: Swing wird auf absehbare Zeit in Java unterstützt. JavaFX lässt sich auch gut in Swing integrieren, um den reibungslosen Übergang von Anwendungen zu ermöglichen.

Es wird dringend empfohlen, die meisten Ihrer Swing-Komponenten im Event Dispatch-Thread zu haben. Es ist leicht zu vergessen, Ihr GUI-Setup in einem Aufruf von `invokeLater` zu bündeln. Aus der Java-Dokumentation:

> Der Swing-Ereignisbehandlungscode wird in einem speziellen Thread ausgeführt, der als Event-Dispatch-Thread bezeichnet wird. Der meiste Code, der Swing-Methoden aufruft, wird auch in diesem Thread ausgeführt. Dies ist notwendig, da die meisten Swing-Objektmethoden nicht "threadsicher" sind: Wenn Sie sie von mehreren Threads aus aufrufen, können Thread-Interferenzen oder Speicherkonsistenzfehler auftreten. Einige Swing-Komponentenmethoden werden in der API-Spezifikation als "threadsicher" bezeichnet. Diese können von jedem Thread aus sicher aufgerufen werden. Alle anderen Swing-Komponentenmethoden müssen vom Event-Dispatch-Thread aus aufgerufen werden. Programme, die diese Regel ignorieren, funktionieren meistens korrekt, unterliegen jedoch unvorhersehbaren Fehlern, die schwer zu reproduzieren sind.

Auch, es sei denn aus gutem Grund, *immer* darauf achten , dass Sie angerufen `setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)` oder sonst könnten Sie möglicherweise mit einem Speicherverlust zu tun haben , wenn Sie vergessen , die JVM zu zerstören.

Examples

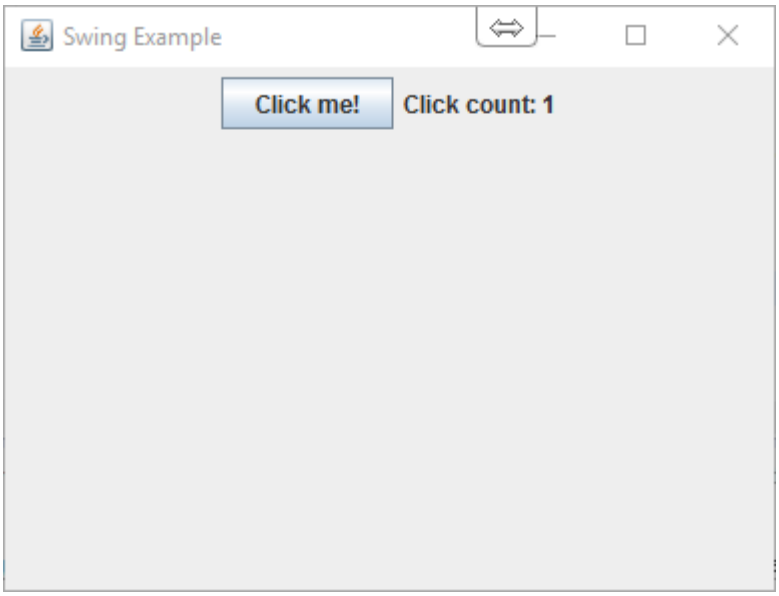
Mit einer Taste inkrementieren

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

/**
 * A very simple Swing example.
 */
public class SwingExample {
    /**
     * The number of times the user has clicked the button.
     */
    private long clickCount;
```

```
    /**
     * The main method: starting point of this application.
     *
     * @param arguments the unused command-line arguments.
     */
    public static void main(final String[] arguments) {
        new SwingExample().run();
    }

    /**
     * Schedule a job for the event-dispatching thread: create and show this
     * application's GUI.
     */
    private void run() {
        SwingUtilities.invokeLater(this::createAndShowGui);
    }

    /**
     * Create the simple GUI for this application and make it visible.
     */
    private void createAndShowGui() {
        // Create the frame and make sure the application exits when the user closes
        // the frame.
        JFrame mainFrame = new JFrame("Counter");
        mainFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // Add a simple button and label.
        JPanel panel = new JPanel();
        JButton button = new JButton("Click me!");
        JLabel label = new JLabel("Click count: " + clickCount);
        panel.add(button);
        panel.add(label);
        mainFrame.getContentPane().add(panel);

        // Add an action listener to the button to increment the count displayed by
        // the label.
        button.addActionListener(actionEvent -> {
            clickCount++;
            label.setText("Click count: " + clickCount);
        });

        // Size the frame.
        mainFrame.setBounds(80, 60, 400, 300);
        //Center on screen
        mainFrame.setLocationRelativeTo(null);
        //Display frame
        mainFrame.setVisible(true);
    }
}
```

Ergebnis

Wie der Button "Klick mich!" gedrückt wird, erhöht sich die Klickanzahl um eins:



"Hallo Welt!" auf Fenstertitel mit Lambda

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Hello World!");
            frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            frame.setSize(200, 100);
            frame.setVisible(true);
        });
    }
}
```

Innerhalb des `main`
In der ersten Zeile wird `SwingUtilities.invokeLater` aufgerufen und ein Lambda-Ausdruck mit einem Block aus code `() -> {...}` wird an ihn übergeben. Dies führt den übergebenen Lambda-Ausdruck im EDT aus, der für Event Dispatch Thread anstelle des Hauptthreads steht. Dies ist notwendig, da im Codeblock des Lambda-Ausdrucks Swing-Komponenten erstellt und aktualisiert werden.

Innerhalb des Codeblocks des Lambda-Ausdrucks:
In der ersten Zeile wird eine neue `JFrame` Instanz namens " `frame` mit dem `new JFrame("Hello World!")` . Dadurch wird eine Fensterinstanz mit "Hello World!" Erstellt. auf dem Titel. Anschließend wird in der zweiten Zeile der `frame` auf `EXIT_ON_CLOSE` konfiguriert. Andernfalls wird das Fenster nur geschlossen, die Ausführung des Programms bleibt jedoch aktiv. Die dritte Zeile konfiguriert die `frame` Instanz mit der `setSize` Methode auf 200 Pixel Breite und 100 Pixel Höhe. Bis jetzt zeigt die Hinrichtung überhaupt nichts. Erst nach dem Aufruf von `setVisible(true)` in der vierten Zeile wird die `frame` Instanz so konfiguriert, dass sie auf dem Bildschirm angezeigt wird.

"Hallo Welt!" auf Fenstertitel mit Kompatibilität

Mit `java.lang.Runnable` machen wir unsere "Hallo Welt!" Beispiel für Java-Benutzer mit Versionen bis zur Version 1.2:

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JFrame frame = new JFrame("Hello World!");
                frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                frame.setSize(200, 100);
                frame.setVisible(true);
            }
        });
    }
}
```

Erste Schritte mit Swing online lesen: https://riptutorial.com/de/swing/topic/2191/erste-schritte-mit-swing

Kapitel 2: Gitterstruktur

Examples

Wie funktioniert GridLayout?

Ein `GridLayout` ist ein Layout-Manager, der Komponenten innerhalb eines Gitters mit gleichen Zellengrößen platziert. Sie können die Anzahl der Zeilen, Spalten, den horizontalen Abstand und den vertikalen Abstand mit den folgenden Methoden festlegen:

- `setRows(int rows)`
- `setColumns(int columns)`
- `setHgap(int hgap)`
- `setVgap(int vgap)`

oder Sie können sie mit den folgenden Konstruktoren setzen:

- `GridLayout(int rows, int columns)`
- `GridLayout(int rows, int columns, int hgap, int vgap)`

Wenn die Anzahl der Zeilen oder Spalten unbekannt ist, können Sie die entsprechende Variable auf `0` . Zum Beispiel:

```
new GridLayout(0, 3)
```

Dies hat zur Folge, dass das `GridLayout` 3 Spalten und so viele Zeilen wie nötig hat.

Das folgende Beispiel `GridLayout` , wie ein `GridLayout` Komponenten mit unterschiedlichen Werten für Zeilen, Spalten, horizontalen Abstand, vertikalen Abstand und Bildschirmgröße `GridLayout` .

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import javax.swing.WindowConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class GridLayoutExample {

    private GridLayout gridLayout;
    private JPanel gridPanel, contentPane;
    private JSpinner rowsSpinner, columnsSpinner, hgapSpinner, vgapSpinner;
```

```
    public void createAndShowGUI() {
        gridLayout = new GridLayout(5, 5, 3, 3);

        gridPanel = new JPanel(gridLayout);

        final ChangeListener rowsColumnsListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setRows((int) rowsSpinner.getValue());
                gridLayout.setColumns((int) columnsSpinner.getValue());
                fillGrid();
            }
        };

        final ChangeListener gapListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setHgap((int) hgapSpinner.getValue());
                gridLayout.setVgap((int) vgapSpinner.getValue());
                gridLayout.layoutContainer(gridPanel);
                contentPane.revalidate();
                contentPane.repaint();
            }
        };

        rowsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getRows(), 1, 10, 1));
        rowsSpinner.addChangeListener(rowsColumnsListener);

        columnsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getColumns(), 1, 10,
1));
        columnsSpinner.addChangeListener(rowsColumnsListener);

        hgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getHgap(), 0, 50, 1));
        hgapSpinner.addChangeListener(gapListener);

        vgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getVgap(), 0, 50, 1));
        vgapSpinner.addChangeListener(gapListener);

        JPanel actionPanel = new JPanel();
        actionPanel.add(new JLabel("Rows:"));
        actionPanel.add(rowsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Columns:"));
        actionPanel.add(columnsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Horizontal gap:"));
        actionPanel.add(hgapSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Vertical gap:"));
        actionPanel.add(vgapSpinner);

        contentPane = new JPanel(new BorderLayout(0, 10));
        contentPane.add(gridPanel);
        contentPane.add(actionPanel, BorderLayout.SOUTH);

        fillGrid();

        JFrame frame = new JFrame("GridLayout Example");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setContentPane(contentPane);
        frame.setSize(640, 480);
```

```
        frame.setLocationByPlatform(true);
        frame.setVisible(true);
    }

    private void fillGrid() {
        gridPanel.removeAll();
        for (int row = 0; row < gridLayout.getRows(); row++) {
            for (int col = 0; col < gridLayout.getColumns(); col++) {
                JLabel label = new JLabel("Row: " + row + " Column: " + col);
                label.setHorizontalAlignment(JLabel.CENTER);
                label.setBorder(BorderFactory.createLineBorder(Color.GRAY));
                gridPanel.add(label);
            }
        }
        contentPane.revalidate();
        contentPane.repaint();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                new GridLayoutExample().createAndShowGUI();
            }
        });
    }

}
```

Gitterstruktur online lesen: https://riptutorial.com/de/swing/topic/2780/gitterstruktur

Kapitel 3: Grafik

Examples

Verwenden der Graphics-Klasse

Intro

Mit der `Graphics` Klasse können Sie auf Java-Komponenten wie ein `Jpanel` . Mit ihr können Sie Strings, Linien, Formen und Bilder zeichnen. Dies geschieht durch *Überschreiben* der `paintComponent(Graphics g)` -Methode der `JComponent` , für die Sie zeichnen, indem Sie das als Argument erhaltene `Graphics` Objekt verwenden, um die Zeichnung `JComponent` :

Klasse `Board`

```
import java.awt.*;
import javax.swing.*;

public class Board extends JPanel{

    public Board() {
        setBackground(Color.WHITE);
    }

    @override
    public Dimension getPreferredSize() {
        return new Dimension(400, 400);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // draws a line diagonally across the screen
        g.drawLine(0, 0, 400, 400);
        // draws a rectangle around "hello there!"
        g.drawRect(140, 180, 115, 25);
    }
}
```

Wrapper-Klasse `DrawingCanvas`

```
import javax.swing.*;

public class DrawingCanvas extends JFrame {

    public DrawingCanvas() {

        Board board = new Board();

        add(board); // adds the Board to our JFrame
```

```
        pack(); // sets JFrame dimension to contain subcomponents

        setResizable(false);
        setTitle("Graphics Test");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        setLocationRelativeTo(null); // centers window on screen
    }

    public static void main(String[] args) {
        DrawingCanvas canvas = new DrawingCanvas();
        canvas.setVisible(true);
    }
}
```

Farben

Um Formen mit unterschiedlichen Farben zu zeichnen, müssen Sie die Farbe des `Graphics` Objekts vor jedem Zeichnungsaufruf mit `setColor` :

```
g.setColor(Color.BLUE);  // draws a blue square
g.fillRect(10, 110, 100, 100);

g.setColor(Color.RED);  // draws a red circle
g.fillOval(10, 10, 100, 100);

g.setColor(Color.GREEN);  // draws a green triangle
int[] xPoints = {0, 200, 100};
int[] yPoints = {100, 100, 280};
g.fillPolygon(xPoints, yPoints, 3);
```

Bilder zeichnen

Bilder können mit der `drawImage` Methode der Klasse `Graphics` auf eine `JComponent` gezeichnet werden:

ein Bild laden

```
BufferedImage img;
try {
    img = ImageIO.read(new File("stackoverflow.jpg"));
} catch (IOException e) {
    throw new RuntimeException("Could not load image", e);
}
```

das bild zeichnen

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    int x = 0;
    int y = 0;

    g.drawImage(img, x, y, this);
}
```

`x` und `y` geben die Position der **oberen linken** Ecke des Bildes an.

Verwenden der Repaint-Methode zum Erstellen einer einfachen Animation

Die MyFrame-Klasse erweitert JFrame und enthält auch die Hauptmethode

```
import javax.swing.JFrame;


public class MyFrame extends JFrame{

    //main method called on startup
    public static void main(String[] args) throws InterruptedException {

        //creates a frame window
        MyFrame frame = new MyFrame();

        //very basic game loop where the graphics are re-rendered
        while(true){
            frame.getPanel().repaint();

            //The program waits a while before rerendering
            Thread.sleep(12);
        }
    }

    //the MyPanel is the other class and it extends JPanel
    private MyPanel panel;

    //constructor that sets some basic staring values
    public MyFrame(){
        this.setSize(500, 500);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creates the MyPanel with paramaters of x=0 and y=0
        panel = new MyPanel(0,0);
        //adds the panel (which is a JComponent because it extends JPanel)
        //into the frame
        this.add(panel);
        //shows the frame window
        this.setVisible(true);
    }

    //gets the panel
    public MyPanel getPanel(){
        return panel;
    }

}
```

Die MyPanel-Klasse, die JPanel erweitert und über die paintComponent-Methode verfügt

```
import java.awt.Graphics;
import javax.swing.JPanel;


public class MyPanel extends JPanel{

    //two int variables to store the x and y coordinate
    private int x;
    private int y;

    //construcor of the MyPanel class
    public MyPanel(int x, int y){
        this.x = x;
        this.y = y;
    }

    /*the method that deals with the graphics
        this method is called when the component is first loaded,
         when the component is resized and when the repaint() method is
        called for this component
    */
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);

        //changes the x and y varible values
        x++;
        y++;

        //draws a rectangle at the x and y values
        g.fillRect(x, y, 50, 50);
    }

}
```

Grafik online lesen: https://riptutorial.com/de/swing/topic/5153/grafik

Kapitel 4: GridBag-Layout

Syntax

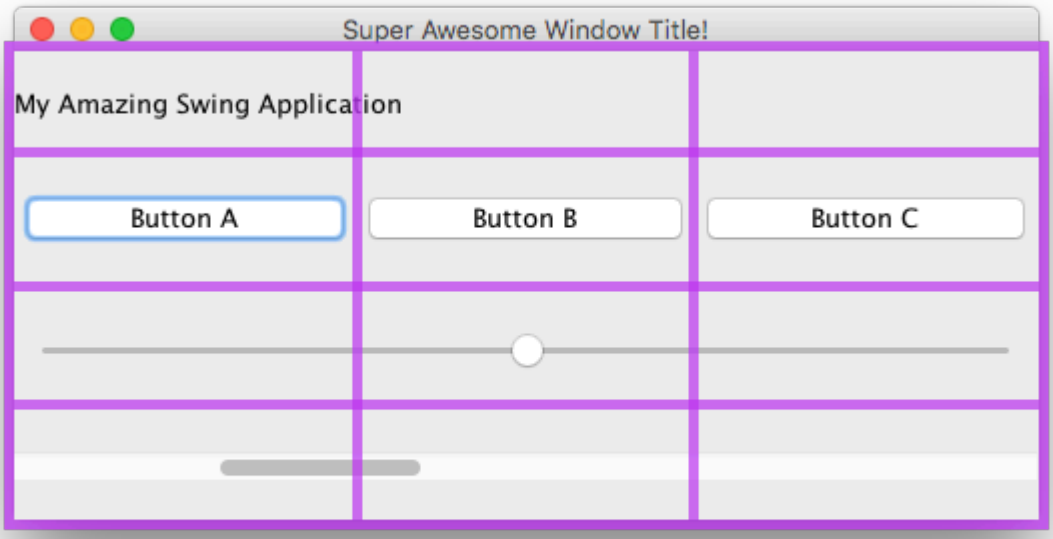
- frame.setLayout (new GridBagLayout ()); // Setze GridBagLayout als Frame
- pane.setLayout (new GridBagLayout ()); // Setze GridBagLayout für Panel
- JPanel-Fenster = neues JPanel (neues GridBagLayout ()); // Setze GridBagLayout für Panel
- GridBagConstraints c = new GridBagConstraints () // Initialisiert eine GridBagConstraint

Examples

Wie funktioniert GridBagLayout?

Layouts werden immer dann verwendet, wenn Ihre Komponenten nicht nur nebeneinander angezeigt werden sollen. Das `GridBagLayout` ist nützlich, da es Ihr Fenster in Zeilen und Spalten unterteilt und Sie entscheiden, in welche Zeile und Spalte Komponenten eingefügt werden sollen und wie viele Zeilen und Spalten die Komponente groß ist.

Nehmen wir dieses Fenster als Beispiel. Gitterlinien wurden markiert, um das Layout anzuzeigen.



Hier habe ich 6 Komponenten erstellt, die mit einem GridBagLayout angelegt wurden.

Komponente	Position	Größe
`JLabel` : "Meine erstaunliche Swing-Anwendung"	`0, 0`	`3, 1`
`JButton` : "Button A"	`0, 1`	`1, 1`

Komponente	Position	Größe
`JButton` : "Button B"	`1, 1`	`1, 1`
`JButton` : "Button C"	`2, 1`	`1, 1`
`JSlider`	`0, 2`	`3, 1`
`JScrollBar`	`0, 3`	`3, 1`

Beachten Sie, dass sich Position `0, 0` oben links befindet: Die Werte für `x` (Spalte) nehmen von links nach rechts zu, die Werte für `y` (Zeile) steigen von oben nach unten.

Um mit dem Auslegen von Komponenten in einem `GridBagLayout` zu beginnen, legen Sie zunächst das Layout Ihres `JFrame` oder Inhaltsbereichs fest.

```
frame.setLayout(new GridBagLayout());
//OR
pane.setLayout(new GridBagLayout());
//OR
JPanel pane = new JPanel(new GridBagLayout()); //Add the layout when creating your content
pane
```

Beachten Sie, dass Sie niemals die Größe des Rasters definieren. Dies geschieht automatisch, wenn Sie Ihre Komponenten hinzufügen.

Anschließend müssen Sie ein `GridBagConstraints` Objekt erstellen.

```
GridBagConstraints c = new GridBagConstraints();
```

Um sicherzustellen, dass Ihre Komponenten die Größe des Fensters ausfüllen, möchten Sie möglicherweise die Gewichtung aller Komponenten auf 1 setzen. Mit der Gewichtung wird festgelegt, wie der Abstand zwischen Spalten und Zeilen aufgeteilt wird.

```
c.weightx = 1;
c.weighty = 1;
```

Möglicherweise möchten Sie auch sicherstellen, dass Komponenten so viel horizontalen Platz wie möglich beanspruchen.

```
c.fill = GridBagConstraints.HORIZONTAL;
```

Sie können auch andere Fülloptionen einstellen, wenn Sie möchten.

```
GridBagConstraints.NONE //Don't fill components at all
GridBagConstraints.HORIZONTAL //Fill components horizontally
GridBagConstraints.VERTICAL //Fill components vertically
GridBagConstraints.BOTH //Fill components horizontally and vertically
```

Beim Erstellen von Komponenten müssen Sie festlegen, wo sich das Raster befinden soll und wie viele Raster-Kacheln es verwenden soll. Um beispielsweise eine Schaltfläche in der 3. Zeile in der

2. Spalte zu platzieren und einen Raster von 5 x 5 zu beanspruchen, gehen Sie wie folgt vor. Beachten Sie, dass das Raster bei `0, 0` und nicht bei `1, 1` beginnt.

```
JButton button = new JButton("Fancy Button!");
c.gridx = 2;
c.gridy = 1;
c.gridwidth = 5;
c.gridheight = 5;
pane.add(buttonA, c);
```

Denken Sie beim Hinzufügen von Komponenten an Ihr Fenster daran, die Einschränkungen als Parameter zu übergeben. Dies ist in der letzten Zeile des obigen Codebeispiels zu sehen.

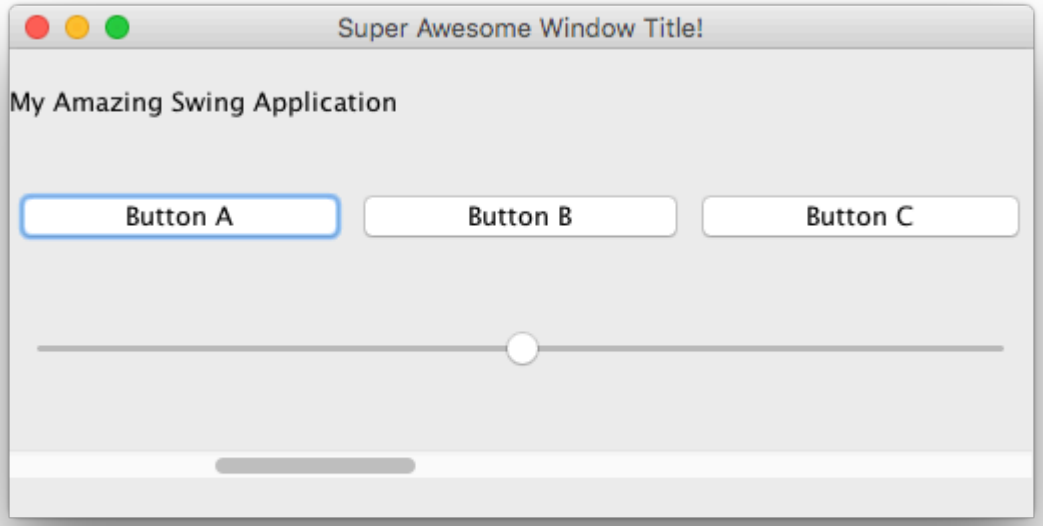
Sie können dieselben `GridBagConstraints` für jede Komponente wiederverwenden. `GridBagConstraints` sie nach dem Hinzufügen einer Komponente ändern, wird die zuvor hinzugefügte Komponente nicht geändert.

Beispiel

Hier ist der Code für das Beispiel am Anfang dieses Abschnitts.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(new GridBagLayout()); //Create a pane to house all content, and give
it a GridBagLayout
frame.setContentPane(pane);

GridBagConstraints c = new GridBagConstraints();
c.weightx = 1;
c.weighty = 1;
c.fill = GridBagConstraints.HORIZONTAL;

JLabel headerLabel = new JLabel("My Amazing Swing Application");
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 0;
pane.add(headerLabel, c);

JButton buttonA = new JButton("Button A");
c.gridx = 0;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonA, c);

JButton buttonB = new JButton("Button B");
c.gridx = 1;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonB, c);

JButton buttonC = new JButton("Button C");
```

```
c.gridx = 2;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonC, c);

JSlider slider = new JSlider(0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 2;
pane.add(slider, c);

JScrollBar scrollBar = new JScrollBar(JScrollBar.HORIZONTAL, 20, 20, 0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 3;
pane.add(scrollBar, c);

frame.setVisible(true); //Show the window
```



GridBag-Layout online lesen: https://riptutorial.com/de/swing/topic/3698/gridbag-layout

Kapitel 5: Grundlagen

Examples

Eine UI-Aufgabe um einen bestimmten Zeitraum verzögern

Alle Swing-bezogenen Operationen finden in einem dedizierten Thread (dem EDT- **E** vent **D** ispatch **T-** Patch) statt. Wenn dieser Thread blockiert wird, reagiert die Benutzeroberfläche nicht mehr.

Wenn Sie eine Operation verzögern möchten, können Sie `Thread.sleep` nicht verwenden. Verwenden `javax.swing.Timer` stattdessen einen `javax.swing.Timer` . Beispielsweise wird der folgende `Timer` den Text eines `JLabel`

```
  int delay = 2000;//specify the delay for the timer
  Timer timer = new Timer( delay, e -> {
    //The following code will be executed once the delay is reached
    String revertedText = new StringBuilder( label.getText() ).reverse().toString();
    label.setText( revertedText );
  } );
  timer.setRepeats( false );//make sure the timer only runs once
```

Ein vollständig lauffähiges Beispiel, das diesen `Timer` ist unten angegeben: Die Benutzeroberfläche enthält eine Schaltfläche und eine Beschriftung. Durch Drücken der Taste wird der Text des Etiketts nach einer Verzögerung von 2 Sekunden umgekehrt

```
import javax.swing.*;
import java.awt.*;

public final class DelayedExecutionExample {

  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }

  private static void showUI(){
    JFrame frame = new JFrame( "Delayed execution example" );

    JLabel label = new JLabel( "Hello world" );
    JButton button = new JButton( "Reverse text with delay" );
    button.addActionListener( event -> {
      button.setEnabled( false );
      //Instead of directly updating the label, we use a timer
      //This allows to introduce a delay, while keeping the EDT free
      int delay = 2000;
      Timer timer = new Timer( delay, e -> {
        String revertedText = new StringBuilder( label.getText() ).reverse().toString();
        label.setText( revertedText );
        button.setEnabled( true );
      } );
      timer.setRepeats( false );//make sure the timer only runs once
      timer.start();
    } );
```

```
    frame.add( label, BorderLayout.CENTER );
    frame.add( button, BorderLayout.SOUTH );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Wiederholen Sie eine UI-Aufgabe in einem festen Intervall

Das Aktualisieren des Status einer Swing-Komponente muss im Event Dispatch Thread (EDT) erfolgen. Der `javax.swing.Timer` löst seinen `ActionListener` im EDT aus und ist daher eine gute Wahl für Swing-Vorgänge.

Im folgenden Beispiel wird der Text eines `JLabel` alle zwei Sekunden aktualisiert:

```
//Use a timer to update the label at a fixed interval
int delay = 2000;
Timer timer = new Timer( delay, e -> {
  String revertedText = new StringBuilder( label.getText() ).reverse().toString();
  label.setText( revertedText );
} );
timer.start();
```

Ein vollständig lauffähiges Beispiel, das diesen `Timer` ist unten aufgeführt: Die Benutzeroberfläche enthält ein Label, und der Text des Labels wird alle zwei Sekunden zurückgesetzt.

```
import javax.swing.*;
import java.awt.*;

public final class RepeatTaskFixedIntervalExample {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated task example" );
    JLabel label = new JLabel( "Hello world" );

    //Use a timer to update the label at a fixed interval
    int delay = 2000;
    Timer timer = new Timer( delay, e -> {
      String revertedText = new StringBuilder( label.getText() ).reverse().toString();
      label.setText( revertedText );
    } );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Eine UI-Task eine festgelegte Anzahl von Malen ausführen

Im `ActionListener` , der an einen `javax.swing.Timer` , können Sie verfolgen, wie oft der `Timer` den `ActionListener` ausgeführt `ActionListener` . Sobald die erforderliche Anzahl von Malen erreicht ist, können Sie die Verwendung `Timer#stop()` Methode , um die stoppen `Timer` .

```
Timer timer = new Timer( delay, new ActionListener() {
  private int counter = 0;
  @Override
  public void actionPerformed( ActionEvent e ) {
    counter++;//keep track of the number of times the Timer executed
    label.setText( counter + "" );
    if ( counter == 5 ){
      ( ( Timer ) e.getSource() ).stop();
    }
  }
});
```

Ein vollständig lauffähiges Beispiel, das diesen `Timer` wird unten angegeben: Es zeigt eine Benutzeroberfläche, bei der der Text des Labels von null bis fünf zählt. Sobald fünf erreicht ist, wird der `Timer` angehalten.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public final class RepeatFixedNumberOfTimes {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated fixed number of times example" );
    JLabel label = new JLabel( "0" );

    int delay = 2000;
    Timer timer = new Timer( delay, new ActionListener() {
      private int counter = 0;
      @Override
      public void actionPerformed( ActionEvent e ) {
        counter++;//keep track of the number of times the Timer executed
        label.setText( counter + "" );
        if ( counter == 5 ){
          //stop the Timer when we reach 5
          ( ( Timer ) e.getSource() ).stop();
        }
      }
    });
    timer.setInitialDelay( delay );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Erstellen Sie Ihren ersten JFrame

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class FrameCreator {

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame();
            //JFrames will not display without size being set
            frame.setSize(500, 500);

            JLabel label = new JLabel("Hello World");
            frame.add(label);

            frame.setVisible(true);
        });
    }

}
```

Wie Sie möglicherweise feststellen, wenn Sie diesen Code ausführen, befindet sich das Etikett an einer sehr schlechten Stelle. Dies lässt sich mit der `add` Methode nur schwer ändern. Um dynamischere und flexiblere Platzierungen zu ermöglichen, checken Sie die Swing Layout Manager aus .

JFrame-Unterklasse erstellen

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    private static CustomFrame statFrame;

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        JLabel label = new JLabel(labelText);
        add(label);

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }

    public void addLabel(String labelText) {
        JLabel label = new JLabel(labelText);
        add(label);
        this.validate();
```

```
    }

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            statFrame = frame;
            frame.setVisible(true);
        });

        try {
            Thread.sleep(5000);
        } catch (InterruptedException ex) {
            //Handle error
        }

        SwingUtilities.invokeLater(() -> statFrame.addLabel("Oh, hello world too."));
    }

}
```

Weitere Informationen zu FlowLayout finden Sie hier .

Einem Ereignis zuhören

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        //Add a button
        JButton btn = new JButton("Hello button");
        //And a textbox
        JTextField field = new JTextField("Name");
        field.setSize(150, 50);
        //This next block of code executes whenever the button is clicked.
        btn.addActionListener((evt) -> {
            JLabel helloLbl = new JLabel("Hello " + field.getText());
            add(helloLbl);
            validate();
        });
        add(btn);
```

```
        add(field);
    }

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            frame.setVisible(true);
        });
    }

}
```

Erstellen Sie ein Popup-Fenster "Bitte warten ..."

Dieser Code kann zu jedem Ereignis hinzugefügt werden, z. B. Listener, Schaltflächen usw. Ein blockierender `JDialog` wird `JDialog` und bleibt bis zum Abschluss des Vorgangs bestehen.

```
final JDialog loading = new JDialog(parentComponent);
JPanel p1 = new JPanel(new BorderLayout());
p1.add(new JLabel("Please wait..."), BorderLayout.CENTER);
loading.setUndecorated(true);
loading.getContentPane().add(p1);
loading.pack();
loading.setLocationRelativeTo(parentComponent);
loading.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
loading.setModal(true);

SwingWorker<String, Void> worker = new SwingWorker<String, Void>() {
    @Override
    protected String doInBackground() throws InterruptedException
        /** Execute some operation */
    }
    @Override
    protected void done() {
        loading.dispose();
    }
};
worker.execute(); //here the process thread initiates
loading.setVisible(true);
try {
    worker.get(); //here the parent thread waits for completion
} catch (Exception e1) {
    e1.printStackTrace();
}
```

JButtons hinzufügen (Hello World Pt.2)

Vorausgesetzt, Sie haben erfolgreich einen JFrame erstellt und Swing wurde importiert ...

Sie können Swing vollständig importieren

```
import javax.Swing.*;
```

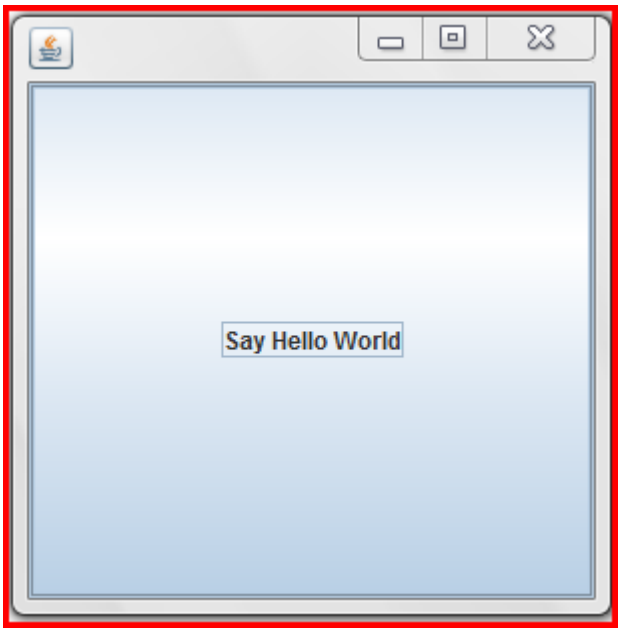
oder Sie können die zu verwendenden Swing-Komponenten / Rahmen importieren

```
import javax.Swing.Jframe;
import javax.Swing.JButton;
```

Nun zum Hinzufügen der J-Taste ...

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

    JFrame frame = new JFrame(); //creates the frame
    frame.setSize(300, 300);
    frame.setVisible(true);

    ////////////////////////ADDING BUTTON BELOW//////////////////////////////
    JButton B = new JButton("Say Hello World");
    B.addMouseListener(new MouseAdapter() {

        public void mouseReleased(MouseEvent arg0) {
            System.out.println("Hello World");
        }

    });
    B.setBounds(0, 0,frame.getHeight(), frame.getWidth());
    B.setVisible(true);
    frame.add(B);
    ////////////////////////////////////////////////////////////////////////
}
```

Wenn Sie diesen Code ausführen / kompilieren, sollten Sie so etwas bekommen ...



Wenn Sie auf die Schaltfläche klicken, sollte "Hello World" ebenfalls in Ihrer Konsole angezeigt werden.

Grundlagen online lesen: https://riptutorial.com/de/swing/topic/5415/grundlagen

Kapitel 6: JList

Examples

Ändern Sie die ausgewählten Elemente in einer JList

Gegeben eine `JList` wie

```
JList myList = new JList(items);
```

die ausgewählten Elemente in der Liste kann durch die modifiziert werden `ListSelectionModel` des `JList` :

```
ListSelectionModel sm = myList.getSelectionModel();
sm.clearSelection();                        // clears the selection
sm.setSelectionInterval(index, index);    // Sets a selection interval
                                          // (single element, in this case)
```

Alternativ bietet `JList` auch einige praktische Methoden, um die ausgewählten Indizes direkt zu bearbeiten:

```
myList.setSelectionIndex(index);            // sets one selected index
                                            // could be used to define the Default Selection

myList.setSelectedIndices(arrayOfIndexes);  // sets all indexes contained in
                                            // the array as selected
```

JList online lesen: https://riptutorial.com/de/swing/topic/5413/jlist

Kapitel 7: Layoutverwaltung

Examples

Rahmenlayout

```
import static java.awt.BorderLayout.*;
import javax.swing.*;
import java.awt.BorderLayout;

JPanel root = new JPanel(new BorderLayout());

root.add(new JButton("East"), EAST);
root.add(new JButton("West"), WEST);
root.add(new JButton("North"), NORTH);
root.add(new JButton("South"), SOUTH);
root.add(new JButton("Center"), CENTER);

JFrame frame = new JFrame();
frame.setContentPane(root);
frame.pack();
frame.setVisible(true);
```

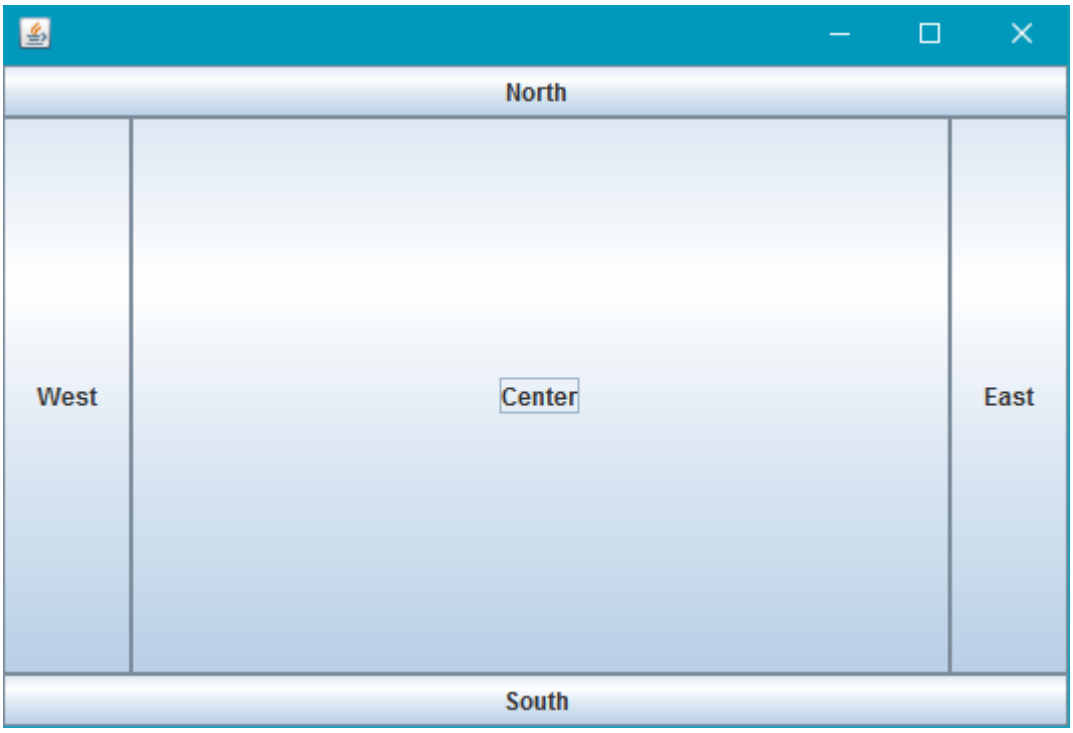
Das Randlayout ist einer der einfachsten Layoutmanager. `JPanel` einen Layout-Manager zu verwenden, können Sie den Manager eines `JPanel` .

Border Layout-Slots folgen den folgenden Regeln:

- Norden und Süden: bevorzugte Höhe
- Osten und Westen: bevorzugte Breite
- Mitte: maximal verbleibender Speicherplatz

In `BorderLayout` Slots auch leer sein. Der Layout-Manager wird Leerstellen automatisch ausgleichen und bei Bedarf die Größe ändern.

So sieht dieses Beispiel aus:



Flusslayout

```
import javax.swing.*;
import java.awt.FlowLayout;

public class FlowExample {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JPanel panel = new JPanel();
                panel.setLayout(new FlowLayout());

                panel.add(new JButton("One"));
                panel.add(new JButton("Two"));
                panel.add(new JButton("Three"));
                panel.add(new JButton("Four"));
                panel.add(new JButton("Five"));

                JFrame frame = new JFrame();
                frame.setContentPane(Panel);
                frame.pack();
                frame.setVisible(true);
            }
        });
    }
}
```

Das Flow-Layout ist der einfachste Layout-Manager, den Swing zu bieten hat. Das Flow-Layout versucht, alles in einer Zeile zu platzieren. Wenn das Layout die Breite überläuft, wird die Zeile umbrochen. Die Reihenfolge wird durch die Reihenfolge angegeben, in der Sie Ihrem Panel Komponenten hinzufügen.

Screenshots:

Gitterstruktur

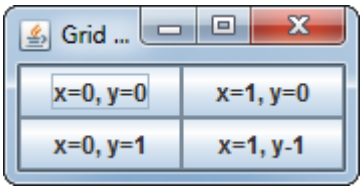
Mit dem `GridLayout` können Sie Komponenten in Form eines Gitters anordnen.

Sie passieren die Anzahl der Zeilen und Spalten Sie das Raster auf denen haben wollen `GridLayout` ,s Konstruktor, zum Beispiel `new GridLayout(3, 2)` ein erstellen `GridLayout` mit drei Zeilen und 2 Spalten.

Beim Hinzufügen von Komponenten zu einem Container mit dem `GridLayout` werden die Komponenten Zeile für Zeile von links nach rechts hinzugefügt:

```
import javax.swing.*;
import java.awt.GridLayout;

public class Example {
    public static void main(String[] args){
        SwingUtilities.invokeLater(Example::createAndShowJFrame);
    }

    private static void createAndShowJFrame(){
        JFrame jFrame = new JFrame("Grid Layout Example");

        // Create layout and add buttons to show restraints
        JPanel jPanel = new JPanel(new GridLayout(2, 2));
        jPanel.add(new JButton("x=0, y=0"));
        jPanel.add(new JButton("x=1, y=0"));
        jPanel.add(new JButton("x=0, y=1"));
        jPanel.add(new JButton("x=1, y-1"));

        jFrame.setContentPane(jPanel);
        jFrame.pack();
        jFrame.setLocationRelativeTo(null);
        jFrame.setVisible(true);
    }
}
```

Dadurch wird ein `JFrame` erstellt und `JFrame` , der wie `JFrame` aussieht:



Eine ausführlichere Beschreibung ist verfügbar: GridLayout

Layoutverwaltung online lesen: https://riptutorial.com/de/swing/topic/5417/layoutverwaltung

Kapitel 8: MigLayout

Examples

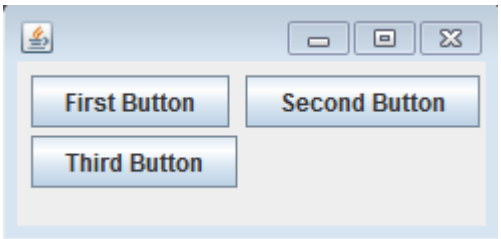
Elemente einwickeln

In diesem Beispiel wird gezeigt, wie insgesamt 3 Schaltflächen platziert werden, wobei sich 2 Schaltflächen in der ersten Reihe befinden. Dann erfolgt ein Umbruch, sodass sich die letzte Schaltfläche in einer neuen Zeile befindet.

Die Einschränkungen sind einfache Zeichenfolgen, in diesem Fall "wrap" beim Platzieren der Komponente.

```
public class ShowMigLayout {

    // Create the elements
    private final JFrame demo = new JFrame();
    private final JPanel panel = new JPanel();
    private final JButton button1 = new JButton("First Button");
    private final JButton button2 = new JButton("Second Button");
    private final JButton button3 = new JButton("Third Button");

    public static void main(String[] args) {
        ShowMigLayout showMigLayout = new ShowMigLayout();
        SwingUtilities.invokeLater(showMigLayout::createAndShowGui);
    }

    public void createAndShowGui() {
        // Set the position and the size of the frame
        demo.setBounds(400, 400, 250, 120);

        // Tell the panel to use the MigLayout as layout manager
        panel.setLayout(new MigLayout());

        panel.add(button1);
        // Notice the wrapping
        panel.add(button2, "wrap");
        panel.add(button3);

        demo.add(panel);
        demo.setVisible(true);
    }
}
```

Ausgabe:



MigLayout online lesen: https://riptutorial.com/de/swing/topic/2966/miglayout

Kapitel 9: MVP-Muster

Examples

Einfaches MVP-Beispiel

Um ein einfaches Beispiel für die Verwendung des MVP-Musters zu veranschaulichen, betrachten Sie den folgenden Code, der eine einfache Benutzeroberfläche mit nur einer Schaltfläche und einem Label erstellt. Wenn Sie auf die Schaltfläche klicken, wird das Label mit der Häufigkeit aktualisiert, mit der die Schaltfläche angeklickt wurde.

Wir haben 5 Klassen:

- Modell - Der POJO, der den Status beibehält (M in MVP)
- Ansicht - Die Klasse mit UI-Code (V in MVP)
- ViewListener - Schnittstelle, die Methoden zum Antworten auf Aktionen in der Ansicht bereitstellt
- Presenter - Reagiert auf Eingaben und aktualisiert die Ansicht (P in MVP)
- Anwendung - Die "Hauptklasse" zum Zusammenführen und Starten der App

Eine minimale „Modell“ Klasse , die nur eine einzige unterhält `count` variabel.

```
/**
 * A minimal class to maintain some state
 */
public class Model {
    private int count = 0;

    public void addOneToCount() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

Eine minimale Benutzeroberfläche, um die Zuhörer zu benachrichtigen:

```
/**
 * Provides methods to notify on user interaction
 */
public interface ViewListener {
    public void onButtonClicked();
}
```

Die Ansichtsklasse erstellt alle Elemente der Benutzeroberfläche. Die Ansicht und *nur* die Ansicht sollten auf Elemente der Benutzeroberfläche verweisen (dh keine Schaltflächen, Textfelder usw. im Präsentator oder in anderen Klassen).

```
/**
 * Provides the UI elements
 */

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

public class View {
    // A list of listeners subscribed to this view
    private final ArrayList<ViewListener> listeners;
    private final JLabel label;

    public View() {
        final JFrame frame = new JFrame();
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setLayout(new GridLayout());

        final JButton button = new JButton("Hello, world!");

        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                notifyListenersOnButtonClicked();
            }
        });
        frame.add(button);

        label = new JLabel();
        frame.add(label);

        this.listeners = new ArrayList<ViewListener>();

        frame.setVisible(true);
    }

    // Iterate through the list, notifying each listner individualy
    private void notifyListenersOnButtonClicked() {
        for (final ViewListener listener : listeners) {
            listener.onButtonClicked();
        }
    }

    // Subscribe a listener
    public void addListener(final ViewListener listener) {
        listeners.add(listener);
    }

    public void setLabelText(final String text) {
        label.setText(text);
    }
}
```

Die Benachrichtigungslogik kann auch in Java8 folgendermaßen codiert werden:

```
        ...
        final Button button = new Button("Hello, world!");
        // In order to do so, our interface must be changed to accept the event parametre
        button.addActionListener((event) -> {
            notifyListeners(ViewListener::onButtonClicked, event);
            // Example of calling methodThatTakesALong, would be the same as callying:
            // notifyListeners((listener, long)->listener.methodThatTakesALong(long), 10L)
            notifyListeners(ViewListener::methodThatTakesALong, 10L);
        });
        frame.add(button);
        ...

/**
 * Iterates through the subscribed listeneres notifying each listener individually.
 * Note: the {@literal '<T>' in private <T> void} is a Bounded Type Parametre.
 *
 * @param <T>      Any Reference Type (basically a class).
 *
 * @param consumer A method with two parameters and no return,
 *                 the 1st parametre is a ViewListner,
 *                 the 2nd parametre is value of type T.
 *
 * @param data     The value used as parametre for the second argument of the
 *                 method described by the parametre consumer.
 */
private <T> void notifyListeners(final BiConsumer<ViewListener, T> consumer, final T data) {
    // Iterate through the list, notifying each listener, java8 style
    listeners.forEach((listener) -> {

        // Calls the funcion described by the object consumer.
        consumer.accept(listener, data);

        // When this method is called using ViewListener::onButtonClicked
        // the line: consumer.accept(listener,data); can be read as:
        // void accept(ViewListener listener, ActionEvent data) {
        //     listener.onButtonClicked(data);
        // }

    });
}
```

Die Schnittstelle muss umgestaltet werden, um das ActionEvent als Parameter zu übernehmen:

```
public interface ViewListener {
    public void onButtonClicked(ActionEvent evt);
    // Example of methodThatTakesALong signature
    public void methodThatTakesALong(long );
}
```

Hier wird nur eine Benachrichtigungsmethode benötigt, die aktuelle Listener-Methode und ihre Parameter werden als Parameter übergeben. Falls erforderlich, kann dies auch für etwas weniger raffiniertes als die eigentliche Ereignisbehandlung verwendet werden. Dies funktioniert alles, solange es eine Methode in der Benutzeroberfläche gibt, z.

```
        notifyListeners(ViewListener::methodThatTakesALong, -1L);
```

Der Moderator kann die Ansicht übernehmen und sich selbst als Zuhörer hinzufügen. Wenn Sie in der Ansicht auf die Schaltfläche klicken, werden alle Zuhörer (einschließlich des Präsentators) in der Ansicht benachrichtigt. Nachdem der Präsentator nun benachrichtigt wurde, kann er geeignete Maßnahmen ergreifen, um das Modell (dh den Status der Anwendung) zu aktualisieren, und anschließend die Ansicht entsprechend aktualisieren.

```
/**
 * Responsible to responding to user interaction and updating the view
 */
public class Presenter implements ViewListener {
    private final View view;
    private final Model model;

    public Presenter(final View view, final Model model) {
        this.view = view;
        view.addListener(this);
        this.model = model;
    }

    @Override
    public void onButtonClicked() {
        // Update the model (ie. the state of the application)
        model.addOneToCount();
        // Update the view
        view.setLabelText(String.valueOf(model.getCount()));
    }
}
```

Um alles zusammenzusetzen, kann die Ansicht erstellt und in den Moderator eingefügt werden. Ebenso kann ein Ausgangsmodell erstellt und eingefügt werden. Beide *können* zwar im Presenter erstellt werden, das Einfügen in den Konstruktor ermöglicht jedoch wesentlich einfachere Tests.

```
public class Application {
    public Application() {
        final View view = new View();
        final Model model = new Model();
        new Presenter(view, model);
    }

    public static void main(String... args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new Application();
            }
        });
    }
}
```

MVP-Muster online lesen: https://riptutorial.com/de/swing/topic/5154/mvp-muster

Kapitel 10: StyledDocument

Syntax

- doc.insertString (Index, Text, Attribute); // Attribute sollten ein AttributeSet sein

Examples

DefaultStyledDocument erstellen

```
try {
    StyledDocument doc = new DefaultStyledDocument();
    doc.insertString(0, "This is the beginning text", null);
    doc.insertString(doc.getLength(), "\nInserting new line at end of doc", null);
    MutableAttributeSet attrs = new SimpleAttributeSet();
    StyleConstants.setBold(attrs, true);
    doc.insertString(5, "This is bold text after 'this'", attrs);
} catch (BadLocationException ex) {
    //handle error
}
```

DefaultStyledDocuments sind wahrscheinlich die am häufigsten verwendeten Ressourcen. Sie können direkt erstellt werden und die abstrakte Klasse `StyledDocument` subclass

Hinzufügen von StyledDocument zu JTextPane

```
try {
        JTextPane pane = new JTextPane();
        StyledDocument doc = new DefaultStyledDocument();
        doc.insertString(0, "Some text", null);
        pane.setDocument(doc); //Technically takes any subclass of Document
    } catch (BadLocationException ex) {
        //handle error
    }
```

Das JTextPane kann dann zu einem beliebigen Swing-GUI-Formular hinzugefügt werden.

DefaultStyledDocument wird kopiert

`StyledDocuments` Allgemeinen keinen Klon und müssen sie auf andere Weise kopieren, wenn dies erforderlich ist.

```
try {
        //Initialization
        DefaultStyledDocument sourceDoc = new DefaultStyledDocument();
        DefaultStyledDocument destDoc = new DefaultStyledDocument();
        MutableAttributeSet bold = new SimpleAttributeSet();
        StyleConstants.setBold(bold, true);
        MutableAttributeSet italic = new SimpleAttributeSet();
```

```
        StyleConstants.setItalic(italic, true);
        sourceDoc.insertString(0, "Some bold text. ", bold);
        sourceDoc.insertString(sourceDoc.getLength(), "Some italic text", italic);

        //This does the actual copying
        String text = sourceDoc.getText(0, sourceDoc.getLength()); //This copies text, but
loses formatting.
        for (int i = 0; i < text.length(); i++) {
            Element e = destDoc.getCharacterElement(i); //A Elment describes a particular part
of a document, in this case a character
            AttributeSet attr = e.getAttributes(); //Gets the attributes for the character
            destDoc.insertString(destDoc.getLength(), text.substring(i, i+1), attr); //Gets
the single character and sets its attributes from the element
        }
    } catch (BadLocationException ex) {
        //handle error
    }
```

Serialisieren eines DefaultStyledDocument an RTF

Mit der AdvancedRTFEditorKit- Bibliothek können Sie ein `DefaultStyledDocument` zu einer RTF-Zeichenfolge serialisieren.

```
try {
   DefaultStyledDocument writeDoc = new DefaultStyledDocument();
   writeDoc.insertString(0, "Test string", null);

   AdvancedRTFEditorKit kit = new AdvancedRTFEditorKit();
   //Other writers, such as a FileWriter, may be used
   //OutputStreams are also an option
   Writer writer = new StringWriter();
   //You can write just a portion of the document by modifying the start
   //and end indexes
   kit.write(writer, writeDoc, 0, writeDoc.getLength());
   //This is the RTF String
   String rtfDoc = writer.toString();

   //As above this may be a different kind of reader or an InputStream
   StringReader reader = new StringReader(rtfDoc);
   //AdvancedRTFDocument extends DefaultStyledDocument and can generally
   //be used wherever DefaultStyledDocument can be.
   //However for reading, AdvancedRTFDocument must be used
   DefaultStyledDocument readDoc = new AdvancedRTFDocument();
   //You can insert at different values by changing the "0"
   kit.read(reader, readDoc, 0);
   //readDoc is now the same as writeDoc
} catch (BadLocationException | IOException ex) {
    //Handle exception
    ex.printStackTrace();
}
```

StyledDocument online lesen: https://riptutorial.com/de/swing/topic/5416/styleddocument

Kapitel 11: Swing Workers und der EDT

Syntax

- öffentliche abstrakte Klasse SwingWorker <T, V>
- T - Der von diesem SwingWorker-Typ doInBackground und get-Methoden zurückgegebene Ergebnistyp.
- V - der Typ, der für die Ausführung von Zwischenergebnissen der Veröffentlichungs- und Verarbeitungsverfahren dieses SwingWorker verwendet wird.
- T doInBackground () - Die abstrakte Funktion, die überschrieben werden muss. Rückgabetyp ist T.

Examples

Haupt- und Ereignisausgabe-Thread

Wie jedes andere Java-Programm beginnt jedes Swing-Programm mit einer Hauptmethode. Die Hauptmethode wird vom Hauptthread initiiert. Swing-Komponenten müssen jedoch im Event-Dispatch-Thread (oder kurz: EDT) erstellt und aktualisiert werden. Um die Dynamik zwischen dem Hauptthread und dem EDT zu veranschaulichen, werfen Sie einen Blick auf diese Hello World! Beispiel.

Der Haupt-Thread wird nur verwendet, um die Erstellung des Fensters an den EDT zu delegieren. Wenn der EDT noch nicht initiiert ist, `SwingUtilities.invokeLater` der erste Aufruf von `SwingUtilities.invokeLater` die erforderliche Infrastruktur für die Verarbeitung von Swing-Komponenten ein. Darüber hinaus bleibt der EDT im Hintergrund aktiv. Der Haupt-Thread wird direkt nach dem Einrichten des EDT-Setups sterben, der EDT bleibt jedoch aktiv, bis der Benutzer das Programm beendet. Dies kann erreicht werden, indem Sie das `JFrame` in der sichtbaren `JFrame` Instanz `JFrame` . Dadurch wird der EDT heruntergefahren und der Programmprozess wird vollständig ablaufen.

Ermitteln Sie die ersten N geraden Zahlen und zeigen Sie die Ergebnisse in einer JTextArea an, in der Berechnungen im Hintergrund ausgeführt werden.

```
import java.awt.EventQueue;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JTextArea;
```

```
import javax.swing.SwingWorker;

class PrimeNumbersTask extends SwingWorker<List<Integer>, Integer> {
    private final int numbersToFind;

    private final JTextArea textArea;

    PrimeNumbersTask(JTextArea textArea, int numbersToFind) {
        this.numbersToFind = numbersToFind;
        this.textArea = textArea;
    }

    @Override
    public List<Integer> doInBackground() {
        final List<Integer> result = new ArrayList<>();
        boolean interrupted = false;
        for (int i = 0; !interrupted && (i < numbersToFind); i += 2) {
            interrupted = doIntenseComputing();
            result.add(i);
            publish(i); // sends data to process function
        }
        return result;
    }

    private boolean doIntenseComputing() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            return true;
        }
        return false;
    }

    @Override
    protected void process(List<Integer> chunks) {
        for (int number : chunks) {
            // the process method will be called on the EDT
            // thus UI elementes may be updated in here
            textArea.append(number + "\n");
        }
    }
}

public class SwingWorkerExample extends JFrame {
    private JTextArea textArea;

    public SwingWorkerExample() {
        super("Java SwingWorker Example");
        init();
    }

    private void init() {
        setSize(400, 400);
        setLayout(new GridLayout(1, 1));
        textArea = new JTextArea();
        add(textArea);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
                System.exit(0);
```

```
            }
        });
    }

    public static void main(String args[]) throws Exception {

        SwingWorkerExample ui = new SwingWorkerExample();
        EventQueue.invokeLater(() -> {
            ui.setVisible(true);
        });

        int n = 100;
        PrimeNumbersTask task = new PrimeNumbersTask(ui.textArea, n);
        task.execute(); // run async worker which will do long running task on a
        // different thread
        System.out.println(task.get());
    }
}
```

Swing Workers und der EDT online lesen: https://riptutorial.com/de/swing/topic/3431/swing-workers-und-der-edt

Kapitel 12: Timer in JFrame

Examples

Timer in JFrame

Angenommen, Sie haben eine Schaltfläche in Ihrem Java-Programm, die eine Zeit herunterzählt. Hier ist der Code für den 10-Minuten-Timer.

```
private final static long REFRESH_LIST_PERIOD=10 * 60 * 1000; //10 minutes

Timer timer = new Timer(1000, new ActionListener() {

   @Override
   public void actionPerformed(ActionEvent e) {
    if (cnt > 0) {
     cnt = cnt - 1000;
     btnCounter.setText("Remained (" + format.format(new Date(cnt)) + ")");
    } else {
     cnt = REFRESH_LIST_PERIOD;
     //TODO
    }

   }
  });

  timer.start();
```

Timer in JFrame online lesen: https://riptutorial.com/de/swing/topic/6745/timer-in-jframe

Kapitel 13: Verwenden von Look and Feel

Examples

Verwendung von System L & F

Swing unterstützt viele native L & Fs.
Sie können eine einfach installieren, ohne eine bestimmte L & F-Klasse zu fordern:

```
public class SystemLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
                {
                    // Installing native L&F as a current application L&F
                    // We do not know what exactly L&F class is, it is provided by the
UIManager
                    UIManager.setLookAndFeel ( UIManager.getSystemLookAndFeelClassName () );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some natively-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
```

```
                content.add ( new JButton ( "Native-looking button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Dies sind die nativen L & Fs, die JDK unterstützt (OS -> L & F):

OS	L & F Name	L & F-Klasse
Solaris, Linux mit GTK +	GTK +	com.sun.java.swing.plaf.gtk.GTKLookAndFeel
Andere Solaris, Linux	Motiv	com.sun.java.swing.plaf.motif.MotifLookAndFeel
Klassisches Windows	Windows	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows XP	Windows XP	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows Vista	Windows Vista	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Macintosh	Macintosh	com.apple.laf.AquaLookAndFeel *
IBM UNIX	IBM	javax.swing.plaf.synth.SynthLookAndFeel *
HP UX	HP	javax.swing.plaf.synth.SynthLookAndFeel *

* Diese L & Fs werden vom Systemanbieter bereitgestellt, und der tatsächliche Name der L & F-Klasse kann variieren

Benutzerdefiniertes L & F verwenden

```
public class CustomLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
```

```
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
                {
                    // Installing custom L&F as a current application L&F
                    UIManager.setLookAndFeel ( "javax.swing.plaf.metal.MetalLookAndFeel" );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some pretty-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
                content.add ( new JButton ( "Metal button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Eine riesige Liste verfügbarer Swing-L & F-Dateien finden Sie im Thema hier: Java Look and Feel (L & F)
Beachten Sie, dass einige dieser L & Fs an diesem Punkt möglicherweise veraltet sind.

Verwenden von Look and Feel online lesen:
https://riptutorial.com/de/swing/topic/3627/verwenden-von-look-and-feel

Kapitel 14: Verwenden von Swing für grafische Benutzeroberflächen

Bemerkungen

Beenden der Anwendung beim Schließen des Fensters

Es ist leicht zu vergessen, die Anwendung zu beenden, wenn das Fenster geschlossen wird. Denken Sie daran, die folgende Zeile hinzuzufügen.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the JFrame is closed
```

Examples

Leeres Fenster erstellen (JFrame)

JFrame erstellen

Ein Fenster zu erstellen ist einfach. Sie müssen nur einen `JFrame` erstellen.

```
JFrame frame = new JFrame();
```

Das Fenster betiteln

Sie können Ihrem Fenster einen Titel geben. Sie können dies tun, indem Sie beim Erstellen des `JFrame` einen String `JFrame` oder `frame.setTitle(String title)` aufrufen.

```
JFrame frame = new JFrame("Super Awesome Window Title!");
//OR
frame.setTitle("Super Awesome Window Title!");
```

Fenstergröße einstellen

Das Fenster ist so klein wie möglich, wenn es erstellt wurde. Um es zu vergrößern, können Sie seine Größe explizit einstellen:

```
frame.setSize(512, 256);
```

Oder Sie können die Frame-Größe selbst anhand der `pack()` -Methode basierend auf der Größe des Inhalts `pack()` .

```
frame.pack();
```

Die `setSize()` und `pack()` sich gegenseitig aus, verwenden Sie also die eine oder die andere.

Was ist bei Window Close zu tun?

Beachten Sie, dass die Anwendung **nicht** beendet wird, wenn das Fenster geschlossen wurde. Sie können die Anwendung beenden, nachdem das Fenster geschlossen wurde, indem Sie den `JFrame` dazu auffordern.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

Alternativ können Sie dem Fenster mitteilen, dass es beim Schließen etwas anderes tun soll.

```
WindowConstants.DISPOSE_ON_CLOSE //Get rid of the window
WindowConstants.EXIT_ON_CLOSE //Quit the application
WindowConstants.DO_NOTHING_ON_CLOSE //Don't even close the window
WindowConstants.HIDE_ON_CLOSE //Hides the window - This is the default action
```

Erstellen eines Inhaltsbereichs

Ein optionaler Schritt ist das Erstellen eines Inhaltsbereichs für Ihr Fenster. Dies ist nicht erforderlich, aber wenn Sie dies möchten, erstellen Sie ein `JPanel` und rufen Sie `frame.setContentPane(Component component)` .

```
JPanel pane = new JPanel();
frame.setContentPane(pane);
```

Das Fenster anzeigen

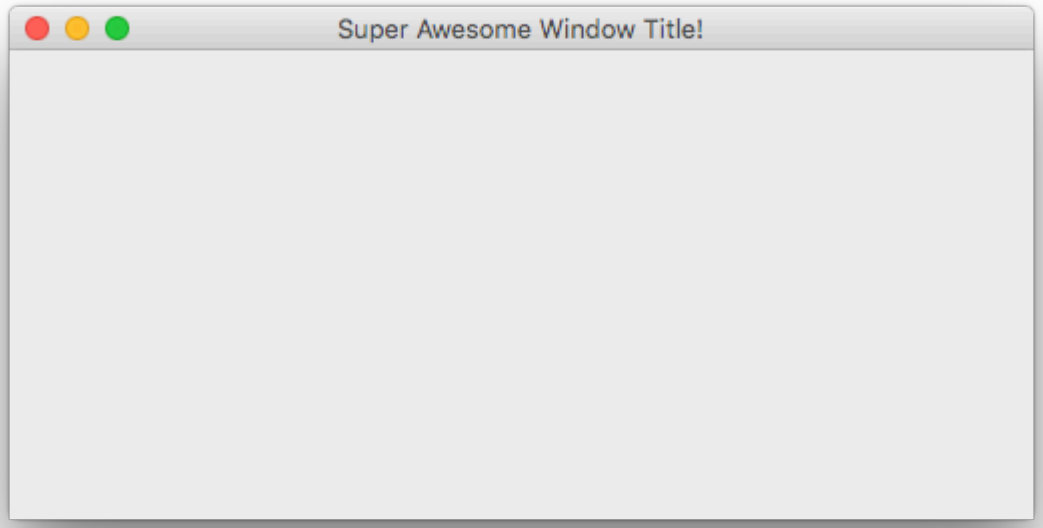
Nach dem Erstellen möchten Sie Ihre Komponenten erstellen und dann das Fenster anzeigen. Die Anzeige des Fensters erfolgt als solches.

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

Beispiel

Für diejenigen von Ihnen, die gerne kopieren und einfügen, hier ein Beispielcode.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

frame.setVisible(true); //Show the window
```



Komponenten hinzufügen

Eine Komponente ist eine Art Benutzeroberflächenelement, z. B. eine Schaltfläche oder ein Textfeld.

Eine Komponente erstellen

Das Erstellen von Komponenten ist nahezu identisch mit dem Erstellen eines Fensters. Anstatt einen `JFrame` erstellen, erstellen Sie diese Komponente. Um beispielsweise eine `JButton` zu erstellen, führen Sie folgende `JButton` aus.

```
JButton button = new JButton();
```

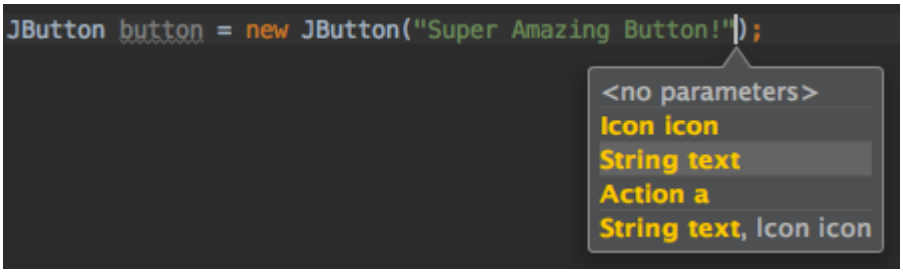
Viele Komponenten können beim Erstellen Parameter übergeben werden. Beispielsweise kann einer Schaltfläche Text angezeigt werden, der angezeigt werden soll.

```
JButton button = new JButton("Super Amazing Button!");
```

Wenn Sie keine Schaltfläche erstellen möchten, finden Sie in einem anderen Beispiel auf dieser Seite eine Liste mit häufig verwendeten Komponenten.

Die Parameter, die an sie übergeben werden können, variieren von Komponente zu Komponente. Eine gute Möglichkeit zu überprüfen, was sie akzeptieren können, ist, sich die Parameter in Ihrer IDE anzusehen (falls Sie eines verwenden). Die Standardverknüpfungen sind unten aufgeführt.

- IntelliJ IDEA - Windows / Linux: `CTRL + P`
- IntelliJ IDEA - OS X / macOS: `CMD + P`
- Eclipse: `CTRL + SHIFT + Space`
- NetBeans: `CTRL + P`



Anzeige der Komponente

Nachdem eine Komponente erstellt wurde, legen Sie normalerweise deren Parameter fest. Danach müssen Sie es irgendwo `JFrame` , beispielsweise in Ihrem `JFrame` oder in Ihrem Inhaltsbereich, falls Sie einen erstellt haben.

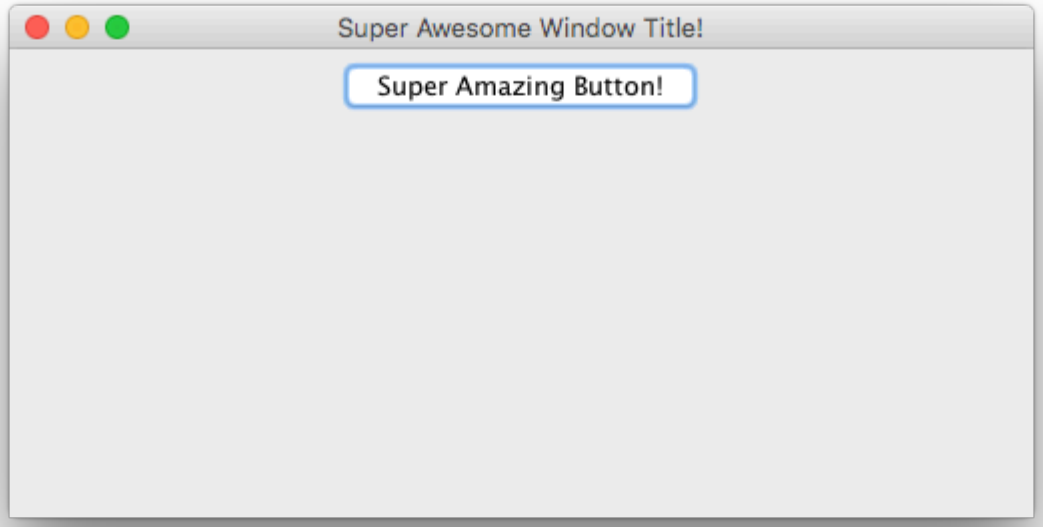
```
frame.add(button); //Add to your JFrame
//OR
pane.add(button); //Add to your content pane
//OR
myComponent.add(button); //Add to whatever
```

Beispiel

Hier ein Beispiel zum Erstellen eines Fensters, Festlegen eines Inhaltsbereichs und Hinzufügen einer Schaltfläche zum Fenster.

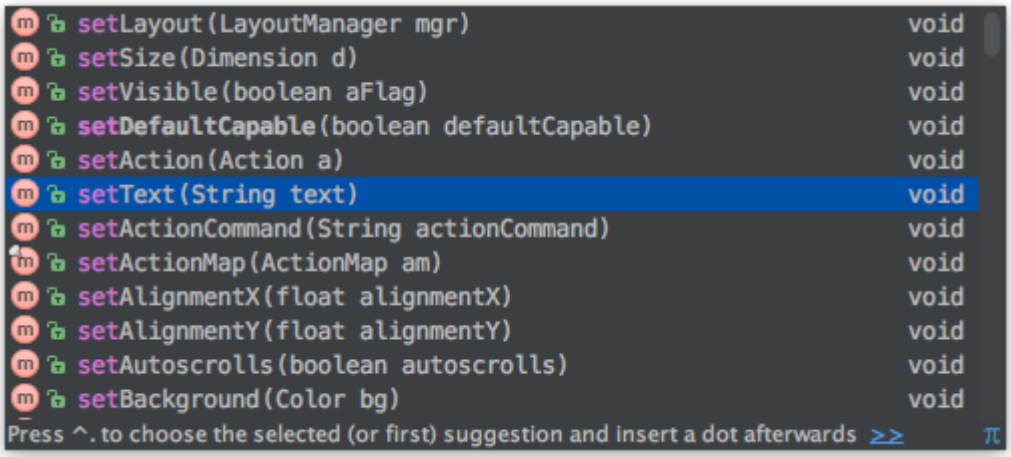
```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

JButton button = new JButton("Super Amazing Button!"); //Create the button
```

```
pane.add(button); //Add the button to the content pane

frame.setVisible(true); //Show the window
```



Parameter für Komponenten einstellen

Komponenten haben verschiedene Parameter, die für sie eingestellt werden können. Sie sind von Komponente zu Komponente unterschiedlich. Um zu ermitteln, welche Parameter für Komponenten festgelegt werden können, sollten Sie mit der Eingabe von `componentName.set` und die automatische Vervollständigung Ihrer IDE (wenn Sie eine IDE verwenden) vorschlagen. Die Standardverknüpfung in vielen IDEs lautet `CTRL + Space` Leertaste, wenn sie nicht automatisch angezeigt wird.



Gemeinsame Parameter, die von allen Komponenten gemeinsam genutzt werden

Beschreibung	Methode
Legt die kleinste Größe für die Komponente fest (nur wenn der Layoutmanager die minimumSize-Eigenschaft berücksichtigt).	`setMinimumSize(Dimension minimumSize)`
Legt die größte Größe der Komponente fest (nur wenn der Layoutmanager die maximumSize-Eigenschaft berücksichtigt).	`setMaximumSize(Dimension maximumSize)`
Legt die perferred-Größe der Komponente fest (nur, wenn der Layout-Manager die Eigenschaft PreferredSize berücksichtigt).	`setPreferredSize(Dimension preferredSize)`
Zeigt oder verbirgt die Komponente	`setVisible(boolean aFlag)`
Legt fest, ob die Komponente auf Benutzereingaben reagieren soll	`setEnabled(boolean enabled)`
Legt die Schriftart für den Text fest	`setFont(Font font)`
Legt den Text der QuickInfo fest	`setToolTipText(String text)`
Legt die Hintergrundfarbe der Komponente fest	`setBackground(Color bg)`
Legt die Vordergrundfarbe (Schriftfarbe) der Komponente fest	`setForeground(Color bg)`

Gemeinsame Parameter in anderen Komponenten

Gemeinsame Komponenten	Beschreibung	Methode
`JLabel`, `JButton`, `JCheckBox`, `JRadioButton`, `JToggleButton`, `JMenu`, `JMenuItem`, `JTextArea`, `JTextField`	Legt den angezeigten Text fest	`setText(String text)`
`JProgressBar`, `JScrollBar`, `JSlider`, `JSpinner`	Legt einen numerischen Wert zwischen den minimalen und maximalen Werten der Komponente fest	`setValue(int n)`
`JProgressBar`, `JScrollBar`, `JSlider`, `JSpinner`	Legt den kleinsten möglichen Wert fest, den die `value` Eigenschaft annehmen kann	`setMinimum(int n)`
`JProgressBar`, `JScrollBar`, `JSlider`, `JSpinner`	Legt den größtmöglichen Wert fest, den die `value` Eigenschaft annehmen kann	`setMaxmimum(int n)`

Gemeinsame Komponenten	Beschreibung	Methode
`JCheckBox` , `JToggleBox`	Legt fest, ob der Wert wahr oder falsch ist (z. B .: Soll ein Kontrollkästchen aktiviert werden?)	`setSelected(boolean b)`

Gemeinsame Komponenten

Beschreibung	Klasse
Taste	`JButton`
Ankreuzfeld	`JCheckBox`
Dropdown-Menü / Kombinationsfeld	`JComboBox`
Etikette	`JLabel`
Liste	`JList`
Menüleiste	`JMenuBar`
Menü in einer Menüleiste	`JMenu`
Element in einem Menü	`JMenuItem`
Panel	`JPanel`
Fortschrittsanzeige	`JProgressBar`
Radio knopf	`JRadioButton`
Scrollleiste	`JScrollBar`
Schieberegler	`JSlider`
Spinner / Zahlenauswahl	`JSpinner`
Tabelle	`JTable`
Baum	`JTree`
Textbereich / mehrzeiliges Textfeld	`JTextArea`
Textfeld	`JTextField`
Werkzeugleiste	`JToolBar`

Interaktive Benutzeroberflächen erstellen

Einen Knopf dort zu haben ist alles gut und gut, aber was ist der Punkt, wenn das Klicken nichts macht? `ActionListener` werden verwendet, um Ihrer Schaltfläche oder anderen Komponente mitzuteilen, dass sie bei Aktivierung etwas tut.

Das Hinzufügen von `ActionListener` erfolgt als solches.

```
buttonA.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //Code goes here...
        System.out.println("You clicked the button!");
    }
});
```

Oder, wenn Sie Java 8 oder höher verwenden ...

```
buttonA.addActionListener(e -> {
    //Code
    System.out.println("You clicked the button!");
});
```

Beispiel (Java 8 und höher)

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create a pane to house all content
frame.setContentPane(pane);

JButton button = new JButton("Click me - I know you want to.");
button.addActionListener(e -> {
    //Code goes here
    System.out.println("You clicked me! Ouch.");
});
pane.add(buttonA);

frame.setVisible(true); //Show the window
```

Komponentenlayout organisieren

Das Hinzufügen von Komponenten nacheinander führt zu einer schwer zu verwendenden Benutzeroberfläche, da sich alle Komponenten **irgendwo befinden** . Die Komponenten sind von oben nach unten angeordnet, wobei jede Komponente in einer separaten "Reihe" angeordnet ist.

Um dem abzuhelfen und Ihnen als Entwickler die Möglichkeit zu geben, Komponenten einfach zu gestalten, bietet Swing `LayoutManager` .

Diese LayoutManager werden ausführlicher in Einführung in Layout Manager sowie in den

separaten Layout Manager-Themen behandelt:

- Gitterstruktur
- GridBag-Layout

Verwenden von Swing für grafische Benutzeroberflächen online lesen:
https://riptutorial.com/de/swing/topic/2982/verwenden-von-swing-fur-grafische-benutzeroberflachen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Swing	Community, Freek de Bruijn, Petter Friberg, Vogel612, XavCo7
2	Gitterstruktur	Lukas Rotter, user6653173
3	Grafik	Adel Khial, Ashlyn Campbell, Squidward
4	GridBag-Layout	CraftedCart, Enwired, mayha, Vogel612
5	Grundlagen	DarkV1, DonyorM, elias, Robin, Squidward
6	JList	Andreas Fester, Squidward, user6653173
7	Layoutverwaltung	explv, J Atkin, mayha, pietrocalzini, recke96, Squidward, XavCo7
8	MigLayout	hamena314, keuleJ
9	MVP-Muster	avojak, ehzawad, Leonardo Pina, sjngm, Squidward
10	StyledDocument	DonyorM, Squidward
11	Swing Workers und der EDT	dpr, isaias-b, rahul tyagi
12	Timer in JFrame	SSD
13	Verwenden von Look and Feel	Mikle Garin
14	Verwenden von Swing für grafische Benutzeroberflächen	CraftedCart, mayha, Michael, Vogel612, Winter