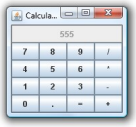


Einführung

Einfache Applikationen zu denen auch dieser Java Taschenrechner gehört, lassen sich heut zu Tage relativ bequem und schnell programmieren. Die entsprechenden API's (**A**pplication **P**rogr amming
I
nterface) stellen jedem Programmierer alle notwendigen Funktionen zur Verfügung welche es unter anderem erlauben Fenster und Dialogfelder auf dem Bildschirm zu zeichnen. Dazu enthält die API Definitionen zur Interaktion der einzelnen Softwarekomponenten untereinander. Versucht wird eine Abstraktion zwischen einer High-Level und einer Low-Level Programmiersprache zu erreichen mit Hilfe derer Low-Level Prozesse, wie das Bilden einer Matrix zur Darstellung von Text schnell realisiert werden können.

Hintergrund

In Java existieren drei wesentliche API's zur Erstellung von grafischen Benutzerschnittstellen. Zum einen das AWT (**A**bstract **W**indow **T**oolkit), zum anderen Swing und zuletzt das SWT (**S**
tandard
W
idget
T
oolkit). Die AWT Klassenbibliotheken von Sun lieferten die ersten Frameworks und GUI Routinen für Java Programmierer. Zudem war AWT das erste Entwicklungssystem für Benutzerschnittstellen in den
Java Foundation Classes
(JFC). Jedoch zeigten sich schnell einige konzeptionelle Fehler beim Design von AWT. Die geringe Abstraktion über die unterstehende native Benutzerschnittstelle führte dazu das es zu Darstellungsproblemen auf verschiedenen Betriebssytemen kam. Während z.B. auf Windows das Programm erwartungsgemäß aussah und funktionierte, konnte es auf Linux ganz anders sein und umgekehrt. In den 90'ern machten sich einige Programmierer einen Spaß daraus indem Sie meinten das Motto von Sun wäre "write once, test everywhere", also "einmal schreiben, überall testen". Dies aber widersprach der Philosophie der Plattformunabhängigkeit von Java.

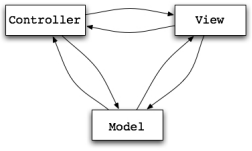
Zwischen 1997 und 98 führte Sun Swing ein, welches komplett in Java geschrieben war und mehr Möglichkeiten offenbarte. In der zweiten JDK Version wurden die AWT Widgets (engl.

Steuerelemente) größtenteils von denen aus dem Swing Toolkit abgelöst. Swing vermeidet die Probleme von AWT indem es seine eigenen Widgets zeichnet. Das bedeutet das Swing-Komponenten direkt von Java gerendert werden und nicht von nativen Betriebssystemkomponenten abhängig sind. Dadurch funktionieren alle Swing-Komponenten auf allen Plattformen, unabhängig davon ob die Plattform eine entsprechende Komponente zur Verfügung stellt oder nicht.
Mittlerweile sprechen viele weitere Gründe für die Nutzung von Swing gegenüber AWT. Darunter die Performance, Vollständigkeit der Klassenbibliotheken und die Entwicklung der Bibliotheken.

Das Problem der optischen Darstellung von Java Programmen mittels Swing, welches darin resultiert das Java Programme nicht so aussehen wie native Programme führte dazu, das IBM 2001 für die Entwicklungsumgebung Eclipse SWT entwickelte. Dabei wurde versucht die Probleme von AWT zu vermeiden indem versucht wird native Widgets durch möglichst dünne Wrapper einzubinden. Das Standard Widget Toolkit Framework greift mittels JNI auf die nativen Komponenten zu. Wenn eine Komponente auf der Plattform nicht verfügbar ist, emuliert SWT diese Komponente. Mittlerweile gibt es verschiedene Meinungen zu den beiden API's. Welches Konzept letztenendes das Bessere ist, lässt sich genauso wenig beantworten, wie die Frage nach dem besten Betriebssystem.

Konzepte von Swing

Ein wichtiger Aspekt der modernen Softwareentwicklung ist das sogenannte *Model-View-Controller* Schema, ein bedeutendes Architekturmuster welches die wesentlichen Aufgabengebiete in separate und miteinander in Wechselwirkung stehende Komponenten unterteilt. Nämlich das Modell selbst, die Ansicht und den Controller. Während sich der Controller um die Interaktion mit dem Benutzer kümmert, liefert das Modell z.B. Daten und teilt der Ansicht eventuelle Änderungen im Abauf mit. Die Ansicht kümmert sich um die Darstellung der Daten.



Über diese *internen* Prozesse in unseren Applikationen brauchen wir im Allgemeinen nicht nachzudenken. Wesentlicher Punkt ist, das jede Benutzerfläche eine Wrapper-Klasse besitzt, welche das Modell und die Ansicht speichert und bei Bedarf Abfragen und Antworten ausführt. Zum Beispiel beim Ausfüllen eines Textfeldes. Bei Swing wurde versucht dieses Konzept eins zu eins umzusetzen.

Erste Schritte

Nachdem wir nun ein wenig über die drei Java API's und die Implementation des Model-View-Controller Konzeptes in Swing gelernt haben, wollen wir uns der Programmierung unseres Taschenechners zuwenden. Wir benutzen dazu die kostenlose integrierte Entwicklungsumgebung (IDE) NetBeans 6.0 in der englischen Version. Selbstverständlich erhalten sie NetBeans auch in deutsch. Alternativ dazu kann auch die bekannte Entwicklungsumgebung Eclipse SDK oder ein schlichter Editor verwendet werden. Zunächst starten wir unsere Entwicklungsumgebung und generieren mit Hilfe des Assistenten ein neues Java Projekt mit dem Namen *Calculator*. Das Tutorial bezieht sich auf das JDK in der Version 1.6.

Vielleicht kennen Sie Formular-Editoren zur Entwicklung der Benutzeroberfläche, wie den Resource Editor in Visual Studio mit welchem per Drag & Drop z.B. Interface Dialogs erzeugt werden können. Das JDK kennt Formular-Editoren als solches nicht. Es gibt zwar in den großen IDE's auch Layout-Werkzeuge, wie den in NetBeans 6.0 neu eingeführten Matisse GUI-Builder, die Funktionen in dieser Richtung unterstützen, aber in der Regel wird in Java der entsprechende Code separat geschrieben. Mit der Entwicklung von Netbeans hat sich der GUI-Builder allerdings sehr verbessert, so dass mittlerweile auch oft zum GUI-Builder gegriffen wird. Auch wenn Sie zum GUI-Builder greifen ist es wichtig zu wissen, was der vom GUI-Builder automatisch erzeugte Code bewirkt.

Lassen Sie uns nun manuell eine neue Java-Klasse generieren. Diese nennen wir entsprechend ebenfalls *Calculator*. Mit Hilfe des Package Explorers können wir nun unsere Projektdateien überblicken. Öffnen sie die Datei Calculator.java falls diese noch nicht zu sehen sein sollte. Dies erreichen sie indem sie im Package Explorer die Baumstruktur öffnen und die Datei doppelklicken. Die Entwicklungsumgebung hat bereits das Grundgerüst für unseren Taschenrechner erzeugt. Dieses sieht wie folgt aus:

[code xml:lang="java"]public class Calculator { /** * @param args the command line arguments */ public static void main(String[] args) { // TODO code application logic here } }[/code]

Frame erzeugen

In Java werden Fenster die nicht Teil eines übergeordneten Fensters sind Frames genannt. Die Programmierung dieser Frames, also im Allgemeinen von Benutzeroberflächen in Java, geschieht wie schon erwähnt größtenteils über die Swing API. Anhand eines vorangestellten **J**, wie z.B. JButton oder JFrame, kann man Swing Klassen übrigens von AWT-Klassen unterscheiden.

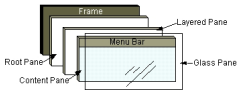
Wir erzeugen nun zunächst einmal einen leeren Frame welches das Basisfenster unseres Taschenrechners werden soll. Dazu importieren wir die Swing-Klassen mittels import javax.swing.* und erzeugen eine von JFrame abgeleitete Subklasse namens **CalculatorFrame** in welcher wir unter anderem den Titel unserer Titelleiste definieren.

[code xml:lang="java"]import javax.swing.*; public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Frame sichtbar machen frame.setVisible(true); } } /** * Leeren Frame ohne Größe erzeugen. */ class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); // Name der Titelleiste } }[/code]

In unserer Subklasse CalculatorFrame haben wir ein CalculatorFrame-Objekt erzeugt mit welchem wir über die Methode setTitle() den Titelnamen angeben. Desweiteren definieren wir in main() was passieren soll wenn das Frame vom Benutzer geschlossen wird. In diesem Fall soll sich unser Programm schließen. Dies erreichen wir über die Anweisung frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE). Als letztes müssen wir das noch unsichtbare Frame "sichtbar" machen, indem wir die Methode setVisible() mit true aufrufen lassen. Fertig! Nun haben wir das Grundgerüst unseres Taschenrechners, nämlich unser Frame programmiert. Unser Frame besitzt zwar noch keine Dimension bzw. die Größe 0 x 0 Pixel, aber das soll uns zu dem Zeitpunkt nicht allzu sehr stören. Probehalber können Sie dem Frame aber eine bestimmte Fenstergröße mittels setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT) zuweisen.

Panel erzeugen

Mit der Klasse JFrame haben wir unserem Taschenrechner einen Rahmen gegeben, einen sogenannten Container. In Java dienen diese Container im Allgemeinen dazu Komponenten wie Teile der Benutzeroberfläche "aufzunehmen". Für Ausgaben oder Menüleisten werden allerdings nicht Frames benutzt sondern die so genannten **Panel**. Panels werden durch die Klasse **JPanel** implementiert und sind selbst Container und Oberflächen. Diese Panels werden in das Frame eingefügt. Das Frame selbst besteht eigentlich aus mehreren übereinander liegenden Layern, nämlich dem Root Pane, dem Layered Pane, dem Glass Pane und dem Content Pane.



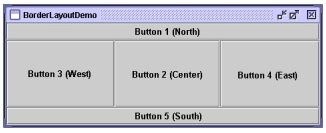
Für uns entscheidend ist das Content Pane in welches wir zunächst einmal ein Panel einfügen. Dazu erzeugen wir wiederum mittels Vererbung eine Klasse mit dem Namen CalculatorPanel. Auch diese Klasse stellt eine von der Superklasse JPanel abgeleitete Subklasse dar.

[code xml:lang="java"]import javax.swing.*; public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); } } class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); CalculatorPanel panel = new CalculatorPanel(); add(panel); } } /** * Panel Grundfläche für die Rechnertasten und Ergebnisanzeige * anlegen */ class CalculatorPanel extends JPanel { public CalculatorPanel() { panel = new JPanel(); } private JPanel panel; }[/code]

Der Aufruf von add(Panel) erfolgt seit dem JDK 1.5 automatisch auf dem Content Pane. In früheren Versionen müssen sie dies mittels getContentPane().add(Panel) realisieren.

Layout zeichnen

Für den Taschenrechner benötigen wir ein sogenanntes Layout . Das Layout bestimmt, wie alle Komponenten in einem Frame angeordnet werden, also wie unser Taschenrechner später aussehen wird. Der Matisse GUI-Builder verwendet einen eigenen Layoutmanager (GroupLayout), der seit Version 6 Teil der Java Platform Standard Edition ist, und kann das Layout in das null-Layout konvertieren. Sofern Sie für das Formular kein explizites Layout programmieren, werden alle Komponenten (Buttons, Textfelder, Labels, etc.) zentriert und gleichmäßig im Panel angeordnet. Dies geschieht mit Hilfe des
Flow-Layout-Managers
, welcher während des Programmablaufs dynamisch die Komponenten anordnet. Java besitzt mehrere Layout-Manager, darunter auch den
Border-Layout-Manager
. Der Border-Layout-Manager ermöglicht die feste Platzierung einzelner Komponenten im Content Pane des JFrame's.



Unser Taschenrechner soll ein homogenes Layout erhalten, in welchem die Zellen mit den Buttons stehts gleichmäßig groß sein sollen. Dies erreichen wir mit Hilfe des Grid-Layout's welches die Komponenten rasterförmig, wie in einer Tabelle anordnet. Dazu rufen wir im Konstruktor des Grid-Layout-Objekts die Anzahl der Zeilen und Spalten auf.

[code xml:lang="java"]import java.awt.*; import javax.swing.*; public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); } } class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); CalculatorPanel panel = new CalculatorPanel(); add(panel); } } class CalculatorPanel extends JPanel { public CalculatorPanel() { setLayout(new BorderLayout()); // Layout-Manager festlegen panel = new JPanel(); // Angabe der

Anzahl von Spalten und Zeilen im Konstruktor panel.setLayout(new GridLayout(4, 4)); // Panel Komponenten im Centerblock platzieren add(panel, BorderLayout.CENTER); } private JPanel panel; }[/code]

Buttons generieren

Die wesentlichen Interface-Komponenten unseres Taschenrechners sind die Schaltflächen, auch Buttons genannt. Wir möchten den Taschenrechner über diese Buttons bedienen und somit Eingaben tätigen. Zusätzlich möchten wir ein Display erzeugen welches und das Ergebnis unserer Rechnungen anzeigt. Die Implementierung von Buttons in Swing geschieht über die Klasse JButton . Wir schreiben uns nun eine Methode **private void addButton(String label)** welche es erlaubt Buttons in das Panel einzufügen. Wir werden aber vorerst nur die numerischen Tasten implementieren. Warum, erfahren wir im nächsten Abschnitt.

[code xml:lang="java"]import java.awt.*; import javax.swing.*; public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); } } class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); CalculatorPanel panel = new CalculatorPanel(); add(panel); // Größe des Fensters an seine Komponenten anpassen pack(); } } class CalculatorPanel extends JPanel { public CalculatorPanel() { setLayout(new BorderLayout()); // Display hinzufügen display = new JButton("0"); display.setEnabled(false); // Display oben positionieren add(display, BorderLayout.NORTH); panel = new JPanel(); panel.setLayout(new GridLayout(4, 4)); // Generierung der numerischen Taschenrechner-Tasten addButton("7"); addButton("8"); addButton("9"); addButton("4"); addButton("5"); addButton("6"); addButton("1"); addButton("2"); addButton("3"); addButton("0"); addButton("."); add(panel, BorderLayout.CENTER); } /** * Taste in die mittlere Grundfläche einfügen. * @param label Beschriftung der Taste */ private void addButton(String label) { JButton button = new JButton(label); panel.add(button); } private JButton display; private JPanel panel; }[/code]

Zusätzlich passen wir die Größe des Taschenrechnerfensters so an, das es sich an die vorhandenen Subkomponenten, also an die Buttons und an das Display anpasst. Dies erreichen wir mit der Funktion void pack().

Ereignisbehandlung

Wir haben nun den Taschenrechner mit Tasten und einem Display ausgestattet. Jedoch sind diese Komponenten derzeit noch funktionslos und bei Betätigung geschieht relativ unspektakuläres, nämlich nichts. Deshalb brauchen wir eine Ereignisbehandlung. Das Betriebssystem überwacht ständig die Umgebung auf Ereignisse, die z.B. durch Mausklicks oder Tastenbetätigungen hervorgerufen werden. Anschließend werden diese Ereignisse dem

Programm "mitgeteilt" und von diesem entsprechend verarbeitet. Java kapselt Ereignisse in so genannte Ereignisobjekte. Sobald ein Ereignis, wie z.B. ein Mausklick auftritt, sendet die Ereignisquelle Ereignisobjekte an alle registrierten Empfänger. Anschließend verwenden die Empfängerobjekte die Informationen in diesem Ereignisobjekt um entsprechende Reaktionen auszulösen.

Bei unseren Tasten ist ein Ereignis ein Klick auf die Schaltfläche. Das heißt sobald der Benutzer auf eine Taste klickt wird ein Ereignis, ein so genanntes *ActionEvent* ausgelöst. Daraufhin erfolgt eine Benachrichtigung an das listener-Objekt. Dazu ist es erforderlich in die Klasse, zu der das Empfängerobjekt (listener) gehört, eine passende Schnittstelle zu implementieren. Um die Schnittstelle zu implementieren, muss der Empfänger über eine Methode namens *actionPerformed*
verfügen, die unser ActionEvent-Objekt als Parameter übernimmt. Sobald der Benutzer auf eine Schaltfläche klickt, erzeugt das JButton-Objekt ein ActionEvent-Objekt, ruft listener.actionPerformed(event) auf und übergibt ein entsprechendes ActionEvent-Objekt.

Für die numerischen Tasten des Taschenrechners möchten wir ein Aktionsbefehl im Falle eines Klicks auslösen. In diesem Fall soll ein entsprechender String übergeben werden. Dazu implementiert die Klasse **DefaultButtonModel** in Swing eine Schnittstelle mit Methoden. Einen Überblick über diese Klasse und alle ihre Methoden erhält man auf der offiziellen Java Sun Seite unter DefaultButtonModel . Für uns ist nur eine Methode der Schnittstelle ButtonModel interessant und diese sehen wir uns etwas näher an.

Wert	Methode
String	getActionCommand - Die Methode gibt einen Aktionsbefehl (action command) für das entsprech

Wir werden nun die ActionListener Schnittstelle implementieren und Methoden includieren. Dazu verwenden wir eine neue Klasse mit dem Namen *InsertAction* und lassen diese das Interface *ActionListener*
implementieren. Anschließend erweitern wir den Code der Klasse um die Methode ActionPerformed welche mittels getActionCommand() das Ereignis des Schaltflächenklicks abfängt.

Da sich der Aktionsbefehl der numerischen Tasten von den Operationstasten, wie +, -, * usw. unterscheidet, erzeugen wir zunächst nur für die numerischen Tasten eine neue Klasse. Deshalb haben wir im vorigen Abschnitt vorerst die Operationstasten ausgelassen.

[code xml:lang="java"]import java.awt.*; import javax.swing.*; // Stellt die Schnittstelle und

Klassen für die // Ereignisbehandlung welche durch AWT Komponenten // erzeugt wurden bereit import java.awt.event.*; public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); } } class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); CalculatorPanel panel = new CalculatorPanel(); add(panel); pack(); } } class CalculatorPanel extends JPanel { public CalculatorPanel() { setLayout(new BorderLayout()); // Startflag setzen start = true; display = new JButton("0"); display.setEnabled(false); add(display, BorderLayout.NORTH); // ActionListener insert erzeugen ActionListener insert = new InsertAction(); panel = new JPanel(); panel.setLayout(new GridLayout(4, 4)); // Die Aktion insert als zweiten Parameter hinzufügen addButton("7", insert); addButton("8", insert); addButton("9", insert); addButton("4", insert); addButton("5", insert); addButton("6", insert); addButton("1", insert); addButton("2", insert); addButton("3", insert); addButton("0", insert); addButton(".", insert); add(panel, BorderLayout.CENTER); } /** * @param label Beschriftung der Taste * @param listener der Ereignisempfänger für die Taste */ private void addButton(String label, ActionListener listener) { JButton button = new JButton(label); // Ereignisempfänger für die Schaltfläche button.addActionListener(listener); panel.add(button); } /** * Diese Aktion fügt den String der Tastenaktion an das Ende * des Anzeigetextes an. */ private class InsertAction implements ActionListener { public void actionPerformed(ActionEvent event) { String input = event.getActionCommand(); // Beim Start keinen Text anzeigen // Das Start Flag anschließend auf false setzen if (start) { display.setText(""); start = false; } // Text anzeigen display.setText(display.getText() + input); } } private JButton display; private JPanel panel; private boolean start; }[/code]

Implementierung der Operationstasten

Wir haben nun die numerischen Schaltflächen auf unserem Taschenrechner erzeugt und eine zugehörige Ereignisbehandlung implementiert. Jeder Mausklick auf eine Schaltfläche hängt einen String, also eine Zahl im Display an. Nun fehlen uns allerdings noch die Operationstasten, die es uns erlauben diese Zahlen z.B. zu addieren, multiplizieren und das Ergebnis anschließend auf dem Display auszugeben. Dazu benötigen wir eine neue Ereignisbehandlung. Wir nennen diese dem Sinn entsprechend *CommandAction*, also Kommandoaktion. Die Vorgehensweise ist dieselbe wie schon bei der Ereignisbehandlung der numerischen Tasten mit dem kleinen Unterschied das wir vorerst noch keinen Code in die Methode actionPerformed() schreiben.

[code xml:lang="java"]import java.awt.*; import javax.swing.*; import java.awt.event.*; public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); } } class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); CalculatorPanel panel = new CalculatorPanel(); add(panel); pack(); } } class CalculatorPanel extends JPanel {

public CalculatorPanel() { setLayout(new BorderLayout()); start = true; display = new JButton("0"); display.setEnabled(false); add(display, BorderLayout.NORTH); ActionListener insert = new InsertAction(); // ActionListener command erzeugen ActionListener command = new CommandAction(); panel = new JPanel(); panel.setLayout(new GridLayout(4, 4)); addButton("7", insert); addButton("8", insert); addButton("9", insert); addButton("/", command); addButton("4", insert); addButton("5", insert); addButton("6", insert); addButton("*", command); addButton("1", insert); addButton("2", insert); addButton("3", insert); addButton("-", command); addButton("0", insert); addButton(".", insert); addButton("=", command); addButton("+", command); add(panel, BorderLayout.CENTER); } private void addButton(String label, ActionListener listener) { JButton button = new JButton(label); button.addActionListener(listener); panel.add(button); } private class InsertAction implements ActionListener { public void actionPerformed(ActionEvent event) { String input = event.getActionCommand(); if (start) { display.setText(""); start = false; } display.setText(display.getText() + input); } } private class CommandAction implements ActionListener { public void actionPerformed(ActionEvent event) { } } private JButton display; private JPanel panel; private boolean start; }[/code]

Rechenoperationen programmieren

Unser Taschenrechner ist nun fast fertig. Es fehlt nur noch der Code der die Rechenoperationen ausführt. Dazu implementieren wir die Methode **public void calculate(double x)** und fügen die entsprechende Berechnung in die Ereignisbehandlung *CommandAction* ein. Negative Zahlen bekommen den Präfix "-", sofern es sich um die erste Operation handelt.

[code xml:lang="java"]import java.awt.*; import java.awt.event.*; import javax.swing.*; /** * Erstellt einen kleinen Taschenrechner. * * @version 1.00 2005-09-03 * @author GAGA * @link http://www.codeplanet.eu/ */ public class Calculator { public static void main(String[] args) { CalculatorFrame frame = new CalculatorFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true); } } class CalculatorFrame extends JFrame { public CalculatorFrame() { setTitle("Calculator"); CalculatorPanel panel = new CalculatorPanel(); add(panel); pack(); } } class CalculatorPanel extends JPanel { public CalculatorPanel() { setLayout(new BorderLayout()); result = 0; lastCommand = "="; start = true; display = new JButton("0"); display.setEnabled(false); add(display, BorderLayout.NORTH); ActionListener insert = new InsertAction(); ActionListener command = new CommandAction(); panel = new JPanel(); panel.setLayout(new GridLayout(4, 4)); addButton("7", insert); addButton("8", insert); addButton("9", insert); addButton("/", command); addButton("4", insert); addButton("5", insert); addButton("6", insert); addButton("*", command); addButton("1", insert); addButton("2", insert); addButton("3", insert); addButton("-", command); addButton("0", insert); addButton(".", insert); addButton("=", command); addButton("+", command); add(panel, BorderLayout.CENTER); } private void addButton(String label, ActionListener listener) { JButton button = new JButton(label);

button.addActionListener(listener); panel.add(button); } private class InsertAction implements ActionListener { public void actionPerformed(ActionEvent event) { String input = event.getActionCommand(); if (start) { display.setText(""); start = false; } display.setText(display.getText() + input); } } /** * Diese Aktion führt den mit der Taste verbundenen * Befehl aus. */ private class CommandAction implements ActionListener { public void actionPerformed(ActionEvent event) { String command = event.getActionCommand(); // Füge den Präfix "-" an den String an wenn // es sich um den ersten Befehl handelt (negative Zahl) if (start) { if (command.equals("-")) { display.setText(command); start = false; } else lastCommand = command; } else { // Berechnung ausführen calculate(Double.parseDouble(display.getText())); lastCommand = command; start = true; } } } /** * Führt die anstehenden Berechnungen aus. * @param x der mit dem vorherigen Ergebnis zu berechnende Wert */ public void calculate(double x) { if (lastCommand.equals("+")) result += x; else if (lastCommand.equals("-")) result -= x; else if (lastCommand.equals("*")) result *= x; else if (lastCommand.equals("/")) result /= x; else if (lastCommand.equals("=")) result = x; display.setText("" + result); } private JButton display; private JPanel panel; private double result; private String lastCommand; private boolean start; }[/code]

Geschafft! Wahrscheinlich haben sie sich den Aufwand etwas größer vorgestellt. Aber wie sie gesehen haben stellt uns Swing und AWT alle notwendigen Klassen und Methoden die wir brauchen zur Verfügung. Sie können den bestehenden Code natürlich um weitere Funktionen, wie eine Taste für die Kreiszahl PI oder eine zur Berechnung der Fakultät, erweitern.

Abschließend finden Sie das zum Tutorial gehörende Programm, sowie den Source Code als ZIP-Paket gepackt zum Download. Das war's, viel Spass beim nachprogrammieren!