

Inhaltsverzeichnis

1	AWT/Swing	2
2	Aufbau einer grafischen Benutzeroberfläche.....	2
3	Vorgehensweise bei der Erstellung.....	2
4	Layout-Manager.....	4
5	Ereignisbehandlung (Event Handling).....	8
5.1	Grundlagen	8
5.2	Listener.....	8
5.2.1	Als "Innere Klasse"	11
5.2.2	Als direkte Implementierung des/der Interfaces durch die GUI Klasse selbst	14
5.2.3	Als anonyme Klasse.....	15
6	Grafische Oberflächen erstellen mit dem Window Builder von Eclipse	16
6.1	Vorgehensweise beim Arbeiten mit dem GUI-Designer:	16
6.2	Installation des plugins "WindowBuilder/SwingDesigner" auf dem eigenen PC.....	17

1 AWT/Swing

In Java gibt es verschiedene Möglichkeiten, um grafische Benutzeroberflächen zu erstellen:

- AWT (Abstract Windowing Toolkit) und
- Swing (seit JDK 1.2)
- JavaFX (neueste GUI Bibliothek; nicht in JDK enthalten)

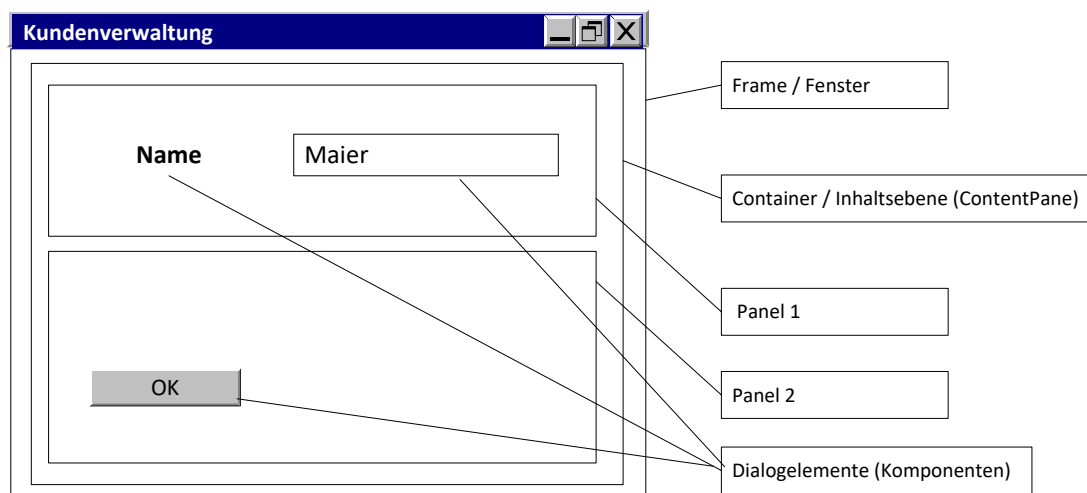
Eigenschaften des AWT:

- Das AWT ist die ursprüngliche Form.
- kennt relative wenige Oberflächenelemente, z.B. Button, CheckBox, Choice, Label, List
- das Aussehen der Oberfläche ist plattformspezifisch, d.h. sie sieht unter Windows anders aus als auf anderen Plattformen.

Eigenschaften von Swing:

- Aussehen der Oberfläche auf allen Plattformen gleich
- mehr Oberflächenelemente, z.B. JSlider, JToolTip, JProgressBar, JTree, Jtable
- einheitliche Namensgebung der Komponenten (beginnen alle mit J, alle Teilwörter; werden großgeschrieben)

2 Aufbau einer grafischen Benutzeroberfläche



3 Vorgehensweise bei der Erstellung

1. Neue GUI-Klasse erzeugen.
Im Konstruktor der GUI-Klasse die Oberflächenelemente erzeugen und in der hierarchischen Struktur einfügen:
2. Zuerst ein `JFrame` Objekt erzeugen und die Inhaltsebene referenzieren.
3. Panels (= "Pinnwände") erzeugen, auf denen die Oberflächenelemente angebracht werden
`JPanel pnlButtons = new JPanel();`
4. die benötigten Oberflächenelemente definieren:
z.B. `JButton btnSpeichern = new JButton();`
5. jedem Panel einen Layout-Manager zuordnen:
z.B. `pnlButtons.setLayout(new LayoutManager());`
6. die Oberflächenelemente mit den jeweiligen Panels verknüpfen:
`pnlButtons.add(btnSpeichern);`
die Reihenfolge des Hinzufügens entscheidet über die Anordnung.

Beispiel:

```
// Importieren der benötigten Pakete:
import javax.swing.*;
import java.awt.*;

public class Bsp1KundeGUI
{
    private JFrame frmKundeGUI; // JFrame ermöglicht die GUI
    private JTextField txtName; // Dialogelemente die in der ganzen Klasse
                                // benötigt werden, als Attribute definieren

    public Bsp1KundeGUI()
    {
        this.frmKundeGUI = new JFrame("Beispiel 1: Kunde GUI");
        // Wenn das Fenster geschlossen wird, soll auch das Programm enden
        this.frmKundeGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // in Swing muss die Inhaltsebene explizit ermittelt werden
        Container contentPane = frmKundeGUI.getContentPane();

        // Deklaration der benötigten Panels
        JPanel pnlButton = new JPanel();
        JPanel pnlText = new JPanel();

        /*
         * Hinzufügen der Oberflächenelemente Buttons, Labels, Textfelder
         */
        JButton btnSpeichern = new JButton("Speichern");
        JLabel lblName = new JLabel("Name");
        txtName = new JTextField("Maier");

        // die Layout-Manager für die jeweiligen Panels festlegen
        contentPane.setLayout(new GridLayout(0, 1));
        pnlButton.setLayout(new FlowLayout());
        pnlText.setLayout(new FlowLayout());

        // Oberflächenelemente den jeweiligen Panels zuordnen
        // (die Reihenfolge entscheidet über die Anordnung)
        pnlButton.add(btnSpeichern);
        pnlText.add(lblName);
        pnlText.add(txtName);

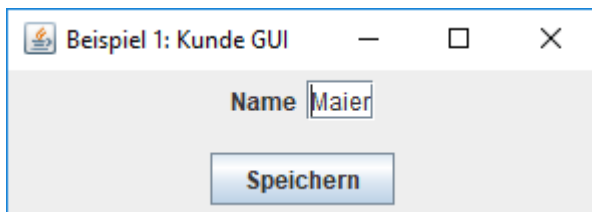
        // verschachtelte Panels zuordnen
        contentPane.add(pnlText);
        contentPane.add(pnlButton);
    }
}
```

```

public static void main(String[] args)
{
    // ein Objekt der neuen GUI-Klasse erzeugen
    Bsp1KundeGUI fenster = new Bsp1KundeGUI();
    // Darstellung des Fensters in der minimalen Grösse
    fenster.frmKundeGUI.pack();

    // Darstellung des Fensters auf dem Bildschirm
    fenster.frmKundeGUI.setVisible(true);
}
}

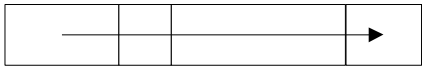
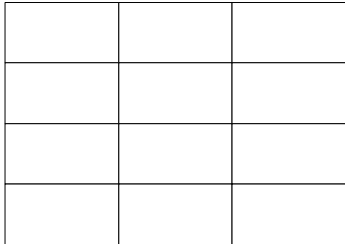
```

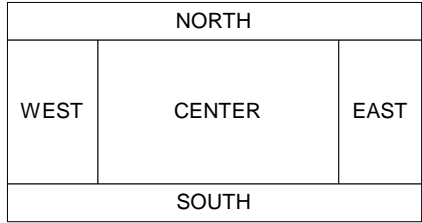
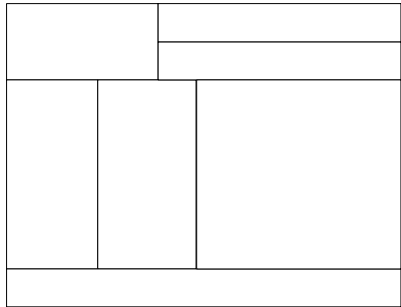
Ergebnis:**4 Layout-Manager**

In vielen Grafikoberflächen werden die Dialogelemente pixel-genau mit ihren Koordinaten auf der Oberfläche angeordnet. Das hat Nachteile beim Vergrößern oder Verkleinern des Fensters oder bei der Anzeige auf unterschiedlichen Plattformen.

In Java ordnet ein Layout-Manager die Dialogelemente auf der Oberfläche an und sorgt automatisch für eine entsprechende Verschiebung der Elemente bei Veränderung der Fenstergröße.

Java kennt verschiedene Layout-Manager:

Manager	Anordnung der Dialogelemente	
NullLayout	feste Vorgabe der Positionen und Größen keine automatische Anordnung	
FlowLayout	nebeneinander in einer Zeile wenn Zeile voll, nächste Zeile	
GridLayout	in Tabellenform Anzahl der Zeilen und Spalten wird beim Erstellen angegeben	

Manager	Anordnung der Dialogelemente	
BorderLayout	in 4 Randbereichen und in der Mitte	
CardLayout	mehrere Unterdialoge in einem Fenster	
GridBagLayout	komplexe Anordnung	

Layout-Manager können auch ineinander geschachtelt werden.

Bevor ein Layout-Manager verwendet werden kann muss er einem Panel zugeordnet werden. Dazu wird ein Objekt des gewünschten Layout-Managers erzeugt und dieses einem Panel mit `pnlPanelname.setLayout (new FlowLayout())` zugeordnet.

Anschließend können die Dialogelemente dem Panel zugewiesen werden. Dabei entscheidet (bei `FlowLayout` und `GridLayout`) die Reihenfolge der Zuweisung über die Anordnung im Panel.

Beispiel:

```
public class Bsp2bLayoutMgmt
{
    private JFrame frmLayoutMgmt;

    public Bsp2LayoutMgmt()
    {
        frmLayoutMgmt = new JFrame("Bsp. 2 Layout Manager");
        // Wenn das Fenster geschlossen wird, soll auch das Programm enden
        this.frmLayoutMgmt.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

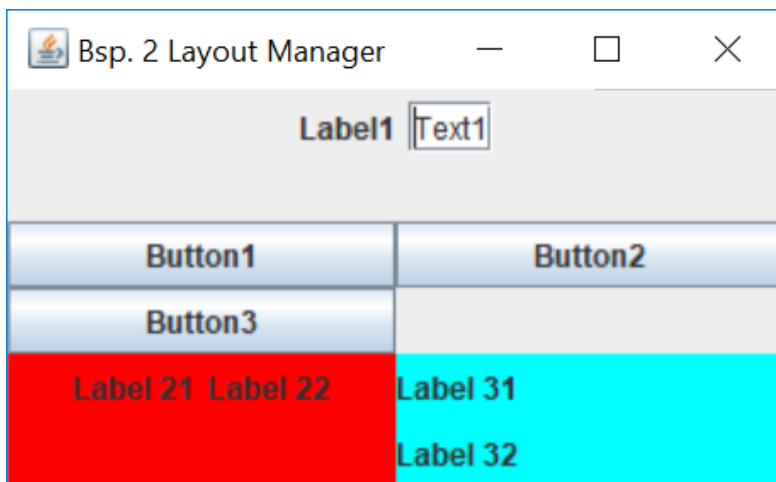
        Container c = frmLayoutMgmt.getContentPane();
        JPanel p1 = new JPanel();
        JPanel p2 = new JPanel();
        JPanel p3 = new JPanel();
        JPanel p31 = new JPanel();
        JPanel p32 = new JPanel();

        // Auf den ContentPane und die Panels werden zur Demonstration
        // verschiedene Layout Manager platziert
        c.setLayout(new GridLayout(0, 1)); // Spaltenanzahl: 1, die Zeilen
            // ergeben sich automatisch (da die Zeilenangabe 0 ist)
        p1.setLayout(new FlowLayout());
        p2.setLayout(new GridLayout(2, 3)); // Zeilenanzahl:2, die Spalten
            // ergeben sich automatisch → Spaltenangabe wird ignoriert
        p3.setLayout(new GridLayout(1, 2));
        p31.setLayout(new FlowLayout());
        p32.setLayout(new GridLayout(2, 0)); // Zeilen: 2, die Spalten
            // ergeben sich automatisch

        p1.add(new JLabel("Label1"));
        p1.add(new JTextField("Text1"));
        p2.add(new JButton("Button1"));
        p2.add(new JButton("Button2"));
        p2.add(new JButton("Button3"));
        p31.add(new JLabel("Label 311"));
        p31.add(new JLabel("Label 312"));
        p31.setBackground(Color.red);
        p32.add(new JLabel("Label 321"));
        p32.add(new JLabel("Label 322"));
        p32.setBackground(Color.cyan);
        p3.add(p31);
        p3.add(p32);

        c.add(p1);
        c.add(p2);
        c.add(p3);
    }
}
```

```
public static void main(String[] args)
{
    Bsp2LayoutMgmt s = new Bsp2bLayoutMgmt();
    // setLocation(x, y) legt die Position des Fensters fest
    s.frmLayoutMgmt.setLocation(100, 200);
    // setSize(x, y) legt die Größe des Fensters fest
    s.frmLayoutMgmt.setSize(400, 200);
    s.frmLayoutMgmt.setVisible(true);
}
}
```

**Hinweise:**

- Beim `GridLayout` gibt man beim Konstruktor sinnvollerweise entweder nur die gewünschte Zeilenzahl oder die gewünschte Spaltenzahl an. Der jeweils andere Wert wird auf 0 initialisiert. Ist die Zeilenzahl vorgegeben, so ergibt sich die Spaltenanzahl automatisch aus der Anzahl der Komponenten die im Grid platziert wird. Werden beide Werte im Konstruktor angegeben, so wird die Spaltenanzahl ignoriert.
- Die Ausrichtung der Komponenten im `FlowLayout` ist standardmäßig zentriert (`FlowLayout.CENTER`). Dies kann durch einen Konstruktor-Parameter verändert werden, oder mit Hilfe einer `set`-Methode.

5 Ereignisbehandlung (Event Handling)

5.1 Grundlagen

Nach der Anzeige der Oberfläche wartet ein Programm mit einer grafischen Benutzeroberfläche normalerweise auf Eingaben des Benutzers (= Ereignis / Event).

Im Gegensatz zu einem Programm mit einer Terminal-Oberfläche kann der Benutzer selbst entscheiden, welche Komponente er als nächstes benutzen möchte. Daher muss eine grafische Benutzeroberfläche jederzeit damit rechnen, dass der Benutzer

- mit der Maus auf einen Button klickt,
- in ein Textfeld von der Tastatur einen Text eingibt,
- aus einer Listbox einen Eintrag auswählt,
- einen Menüpunkt anwählt
- den Mauszeiger einfach nur über den Bildschirm bewegt
- einen Rollbalken hin und her schiebt
- etc.

Diese Benutzeraktionen lösen Ereignisse aus, die vom jeweiligen Betriebssystem an das Programm weitergegeben werden.

U.a. gibt es folgende **Ereignisarten** (Auszug):

ActionEvent	Klick auf Buttons
MouseEvent	Aktionen mit der Maus
KeyEvent	Aktionen auf Tastenbetätigung der Tastatur
TextEvent	Aktionen, wenn Inhalt eines Textfeldes verändert wurde
AdjustmentEvent	Verschieben eines Schiebereglers
FocusEvent	eine Komponente wird aktiviert, z.B. Cursor wird in ein Textfeld gesetzt
ItemEvent	ein Menüpunkt wird ausgewählt
WindowEvent	ein Fenster wird verändert (vergrößert, verkleinert, geschlossen, ...)

Jede Oberflächenkomponente, die ein Ereignis auslösen kann, wird als **Ereignisquelle** bezeichnet.

5.2 Listener

Damit eine Oberflächenkomponente vom Betriebssystem über Ereignisse informiert werden kann, die der Benutzer ausgelöst hat, muss die Oberflächenkomponente auf die Ereignisse, für die sie sich interessiert (z.B. Maus-Klicks) achten, d.h. sie muss einen geeigneten **Ereignis-„Abhörer“** oder –**Listener** aktivieren:

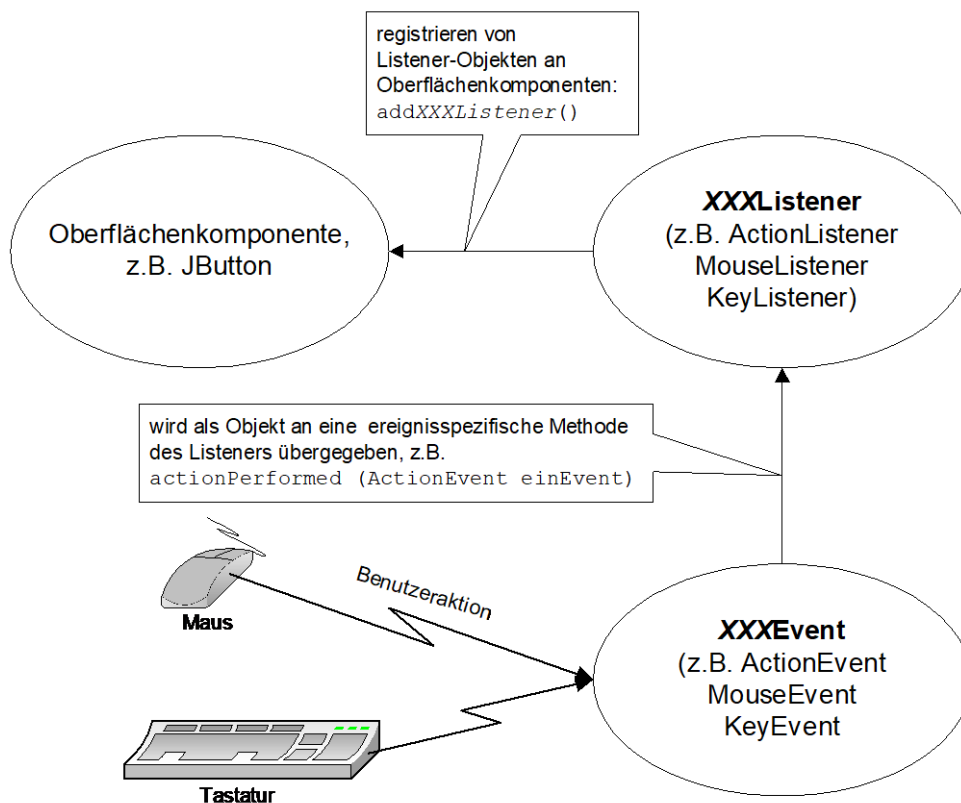
Zu den jeweiligen Ereignisarten gibt es folgende Listener:

Listener	Ereignisquellen (z.B.)	Methoden, die beim Auftreten eines Events aufgerufen werden
ActionListener	JButton, JList, JMenuItem, JTextField	actionPerformed eine Aktion wurde ausgeführt

MouseListener	alle Komponenten	mouseClicked	Maustaste wurde gedrückt und wieder losgelassen
		mouseEntered	Mauszeiger betritt die Komponente
		mouseExited	Mauszeiger verlässt die Komponente
		mousePressed	Maustaste wurde gedrückt
		mouseReleased	Maustaste wurde wieder losgelassen
MouseMotionListener	alle Komponenten	mouseDragged	Maustaste wurde gedrückt und gezogen
		mouseMoved	Mauszeiger wurde bewegt und nicht gedrückt
KeyListener	alle Komponenten	keyPressed	Taste wurde gedrückt
		keyReleased	Taste wurde losgelassen
		keyTyped	Taste wurde gedrückt und wieder losgelassen
TextListener	JTextField, JTextArea	textValueChanged	Der Text wurde verändert
AdjustmentListener	JScrollBar	adjustmentValueChanged	Der Wert wurde verändert
FocusListener	alle Komponenten	focusGained	Komponente erhält den Focus
		focusLost	Komponente verliert den Focus
ItemListener	JCheckBox, JChoice, JList	itemStateChanged	Zustand hat sich verändert
WindowListener	JFrame	windowActivated	das Fenster wurde aktiviert
		windowClosed	das Fenster wurde geschlossen
		windowClosing	das Fenster wird geschlossen
		windowDeactivated	das Fenster wurde deaktiviert
		windowDeiconified	das Fenster wurde wieder hergestellt
		windowIconified	das Fenster wurde auf Symbolgröße verkleinert
		windowOpened	das Fenster wurde geöffnet

Ein Listener ist ein Interface, das die Ausprogrammierung der aufgeführten Methoden von der implementierenden Klasse fordert. Diese Methoden enthalten den Programmcode, der ausgeführt werden soll, wenn ein bestimmtes Ereignis eingetreten ist. Sie werden auch als Ereignismethoden oder „Callback-Methode“ bezeichnet.

Ein Listener wird von der betreffenden Oberflächenkomponente mit der Methode `addListener` registriert (zugeordnet).



Vorgehensweise bei der Implementierung:

1. Überlegen, von welcher Ereignisquelle (Oberflächenkomponente) Ereignisse (Events) abgehört werden sollen (z.B. JButton btnOK)
2. Überlegen, welche Events abgehört werden sollen (Maus (MouseEvent), Tastatur (KeyEvent) ...)
3. an dieser Komponente einen entsprechenden Listener registrieren (z.B. btnOK.addMouseListener(ListenerObjekt))
4. Programmierung des Listeneres. Dazu stehen im Wesentlichen zwei Möglichkeiten zur Verfügung:

Möglichkeiten zur Programmierung des Listeners

5.2.1 Als "Innere Klasse"

Alternative 1: Wir schreiben eine eigene innere Klasse, die das gewünschte Listener-Interface *implementiert*, also z.B. `MouseListener`, `KeyListener`, `ActionListener` oder `WindowListener`. Somit müssen alle Methoden, die das Interface fordert, implementiert werden:

```
class MausLauscher implements MouseListener
{
    public void mouseClicked(MouseEvent e)
    {
        lblMeldung.setText("Mausklick");
    }
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
}
```

Alternative 2: Wir schreiben eine eigene innere Klasse, die eine entsprechende **Adapterklasse** ableitet, die ihrerseits das gewünschte Listener-Interface implementiert hat, z.B. `MouseAdapter`, die bereits im jdk vorhanden ist:

```
class MausLauscher extends MouseAdapter
{
    public void mouseClicked(MouseEvent event)
    {
        lblMeldung.setText("Mausklick");
    }
}
```

Vorteil: es müssen nur die Methoden überschrieben werden, die auch tatsächlich benutzt werden. Alle anderen brauchen nicht implementiert werden (, da sie ja bereits von der Adapterklasse implementiert wurden).

Weitere vorhandene Adapterklassen sind `KeyAdapter`, `MouseAdapter`, `FocusAdapter` oder `WindowAdapter`.

Alternative 1 oder 2 werden als "innere Klasse" innerhalb der GUI-Klasse programmiert. Dadurch ergibt sich der Vorteil, dass die innere Klasse direkt auf die Attribute der GUI-Klasse zugreifen kann:

Beispiel (Innere Klasse die eine Adapterklasse ableitet):

```
/* das Beispiel von oben wird erweitert um ein Meldungslabel,
 * das in ein eigenes Meldungspanel gelegt wird.
 * Ausserdem wird eine innere Klasse für den Maus-Listener
 * angelegt, von der ein Objekt an den Speicherbutton geknüpft wird. */
public class Bsp3bKundeGUI
{
    private JFrame frmKundeGUI;
    private JTextField txtName;
    // Zusätzliches Label
    private JLabel lblMeldung;

    public Bsp3bKundeGUI()
    {
        this.frmKundeGUI = new JFrame("Beispiel 3b: Kunde GUI");
        // Wenn das Fenster geschlossen wird, soll auch das Programm enden
        this.frmKundeGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // in Swing muss die Inhaltsebene explizit ermittelt werden
        Container contentPane = frmKundeGUI.getContentPane();

        // Deklaration der benötigten Panels
        JPanel pnlText = new JPanel();
        JPanel pnlButton = new JPanel();
        JPanel pnlMeldung = new JPanel(); // neu in v2

        /*
         * Hinzufügen der Oberflächenelemente Buttons, Labels, Textfelder
         */
        JButton btnSpeichern = new JButton("Speichern");
        JLabel lblName = new JLabel("Name");
        txtName = new JTextField("Maier");
        lblMeldung = new JLabel(""); // neu in v2
        lblMeldung.setForeground(Color.RED);

        // die Layout-Manager für die jeweiligen Panels festlegen
        contentPane.setLayout(new GridLayout(0, 1));
        pnlText.setLayout(new FlowLayout());
        pnlButton.setLayout(new FlowLayout());
        pnlMeldung.setLayout(new FlowLayout());

        // Oberflächenelemente den jeweiligen Panels zuordnen
        // (die Reihenfolge entscheidet über die Anordnung)
        pnlText.add(lblName);
        pnlText.add(txtName);
        pnlButton.add(btnSpeichern);
        pnlMeldung.add(lblMeldung);

        // verschachtelte Panels zuordnen
```

```
        contentPane.add(pnlText);
        contentPane.add(pnlButton);
        contentPane.add(pnlMeldung);

        // anmelden der Listener für die Buttons ...
        btnSpeichern.addMouseListener(new MausLauscher());
    }

    // "INNERE KLASSE" für den Maus-Listener als

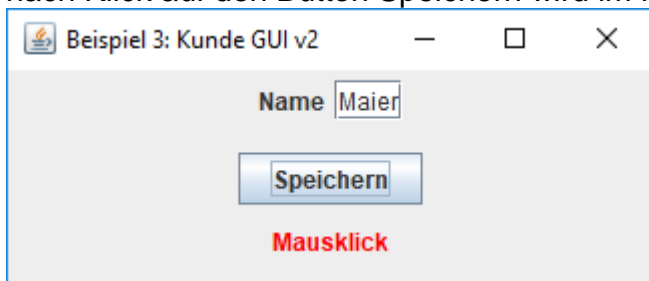
    class MausLauscher extends MouseAdapter
    {
        public void mouseClicked(MouseEvent event)
        {
            // an dieser Stelle werden alle Aktionen programmiert,
            // die nach einem Klick auf den Button ausgeführt werden sollen
            lblMeldung.setText("Mausklick");
        }
    }

    public static void main(String[] args)
    {
        // ein Objekt der neuen GUI-Klasse erzeugen
        Bsp3bKundeGUI fenster = new Bsp3bKundeGUI();
        // Darstellung des Fensters in der optimalen Grösse
        fenster.frmKundeGUI.pack();

        // Darstellung des Fensters auf dem Bildschirm
        fenster.frmKundeGUI.setVisible(true);
    }
}
```

Ergebnis:

nach Klick auf den Button Speichern wird im MeldungsLabel der Text „Mausklick“ angezeigt:



5.2.2 Als direkte Implementierung des/der Interfaces durch die GUI Klasse selbst

Bei dieser Implementierungsvariante wird das entsprechende Interface direkt von der GUI-Klasse implementiert, es gibt keine innere Klasse:

```
public class Bsp4aKundeGUI implements MouseListener
{
    private JFrame frmKundeGUI;
    private JTextField txtName;
    private JLabel lblMeldung;

    public Bsp4aKundeGUI()
    {
        // ***** hier ändert sich nichts! (s.o.) *****

        // ***** hier gibt es eine Änderung! *****
        // anmelden des Listener für den Button ...
        // "ich" bin selbst ein MouseListener, daher "this":
        btnSpeichern.addMouseListener(this);
    }

    // *** Die GUI Klasse muss die Interface Methoden implementieren! ***

    @Override
    public void mouseClicked(MouseEvent e)
    {
        // an dieser Stelle werden alle Aktionen programmiert,
        // die nach einem Klick auf den Button ausgeführt werden sollen
        lblMeldung.setText("Mausklick");
    }

    @Override
    public void mousePressed(MouseEvent e){}
    @Override
    public void mouseReleased(MouseEvent e){}
    @Override
    public void mouseEntered(MouseEvent e){}
    @Override
    public void mouseExited(MouseEvent e) {}

    public static void main(String[] args)
    {
        // ein Objekt der neuen GUI-Klasse erzeugen
        Bsp4aKundeGUI fenster = new Bsp4aKundeGUI();
        // Darstellung des Fensters in der optimalen Grösse
        fenster.frmKundeGUI.pack();
        // Darstellung des Fensters auf dem Bildschirm
        fenster.frmKundeGUI.setVisible(true);
    }
}
```

5.2.3 Als anonyme Klasse

Die Implementierung als anonyme Klasse bietet den Vorteil, dass keine Klasse definiert und kein Interface implementiert werden muss.

Der erforderliche Code wird dort programmiert, wo der Listener instanziiert wird, also beim Registrieren an der Komponente.

Diese Variante bietet sich vor allem an, wenn sehr wenig Code in den ListenerMethoden zu implementieren ist.

```
public class Bsp4bKundeGUI
{
    private JFrame frmKundeGUI;
    private JTextField txtName;
    private JLabel lblMeldung;

    public Bsp4bKundeGUI()
    {
        // ***** hier ändert sich nichts! (s.o.) *****

        // ***** hier gibt es eine Änderung! *****
        // anmelden des Listener für den Button ...
        // als anonyme Klasse:

        btnSpeichern.addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent e)
            {
                lblMeldung.setText("Mausklick ");
            }
        });
    }

    public static void main(String[] args)
    {
        // ein Objekt der neuen GUI-Klasse erzeugen
        Bsp4bKundeGUI fenster = new Bsp4bKundeGUI();
        // Darstellung des Fensters in der optimalen Grösse
        fenster.frmKundeGUI.pack();
        // Darstellung des Fensters auf dem Bildschirm
        fenster.frmKundeGUI.setVisible(true);
    }
}
```

6 Grafische Oberflächen erstellen mit dem Window Builder von Eclipse

Es gibt verschiedene Tools um grafische Oberflächen mit Java produktiver und einfacher zu erstellen. Eclipse bietet hierfür den Window Builder (und weitere Alternativen).

6.1 Vorgehensweise beim Arbeiten mit dem GUI-Designer:

- **Neues Package oder neues Java-Projekt anlegen**
- **neue GUI-Klasse erstellen mit:**
File → new → other → WindowBuilder → SwingDesigner → gewünschten Fenstertyp auswählen, z.B. Application Window → Klassenname vergeben

danach sehen Sie den generierten Code (Registerkarte Source).

- **Registerkarte 'Design' auswählen → Palette mit GUI-Komponenten wird angezeigt**
-

Bemerkungen:

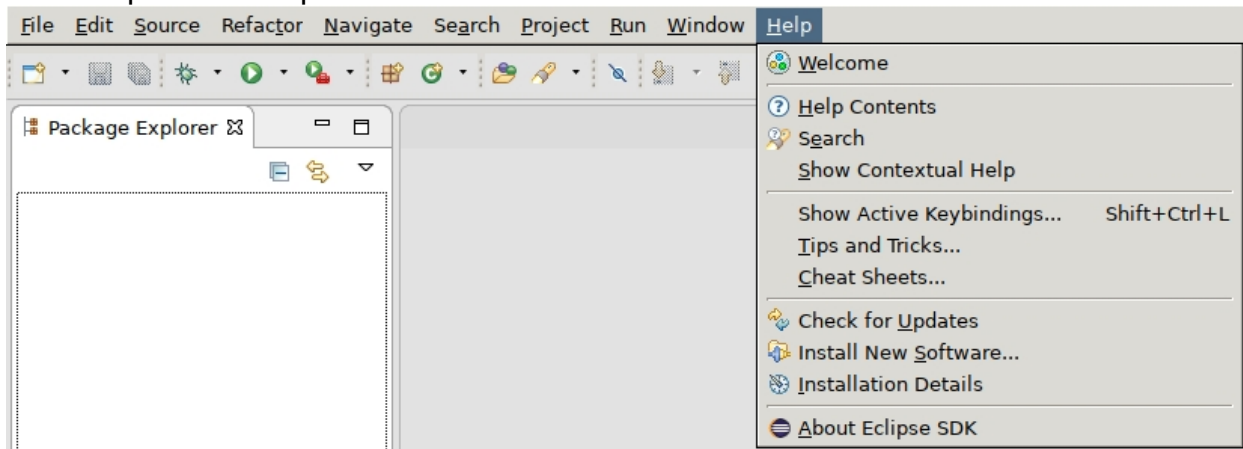
- falls das Fenster mit dieser Palette versehentlich geschlossen wird, kann es neu angezeigt werden mit: Window → show view → other → WindowBuilder → Palette
 - Ein Doppelklick auf die .java-Datei schafft mehr Platz und besseren Überblick, weil dann nur noch die für das Design erforderlichen Fenster zu sehen sind.
 - falls die Design-Ansicht versehentlich geschlossen wurde: Klick mit rechter Maustaste auf die entsprechende Datei im 'package explorer' → open with → WindowBuilder Editor
- **gewünschtes Layout** (für Container) anklicken → 'contentPane' im Komponentenbaum unter dem entsprechenden Container anklicken und im 'properties'-Fenster bei Layout bzw. Constraints weitere Eigenschaften einstellen.
 - **Basiskomponente** (bzw. weitere Containerkomponente) **aufnehmen:**
entsprechende Komponente in Palette anklicken → aufzunehmenden Container anklicken.
Alle Eigenschaften einer (gerade markierten) GUI-Komponente werden im "properties"-Fenster angezeigt und können dort angepasst werden.
 - **"eventhandling"** für eine GUI-Komponente implementieren:
Klick mit rechter Maustaste auf die Komponente → im Kontextmenü auswählen: add event handler →
gewünschtes event auswählen, z.B. action für ActionListener → actionPerformed

Vorteil beim Arbeiten mit einem UI-Designer:

Bei Änderungen im Layout ändert sich der Source-Code – und umgekehrt: **"reverse engineering"**

6.2 Installation des plugins "WindowBuilder/SwingDesigner" auf dem eigenen PC

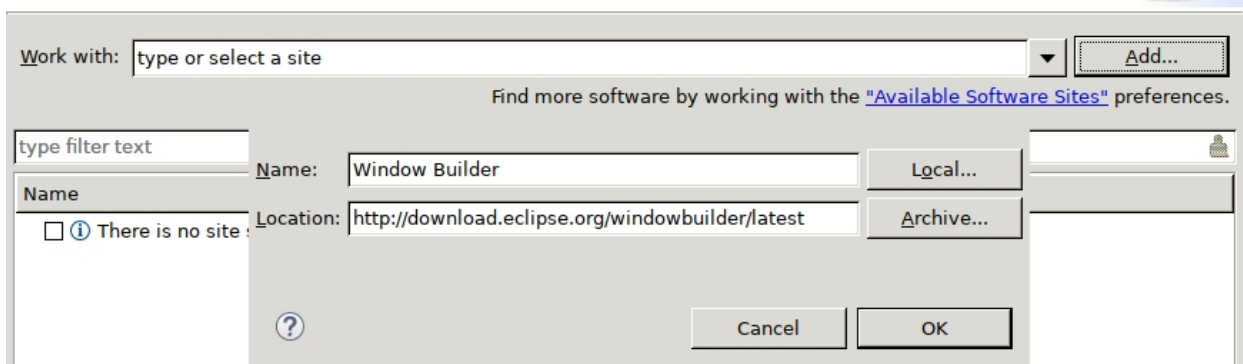
1. In Eclipse auf "Help" und dann auf "Install New Software":



2. Im erscheinenden Fenster auf "Add" klicken, bei "Name" "Window Builder" und bei "Location" die URL "http://download.eclipse.org/windowbuilder/latest" auswählen bzw. eingeben:

Available Software

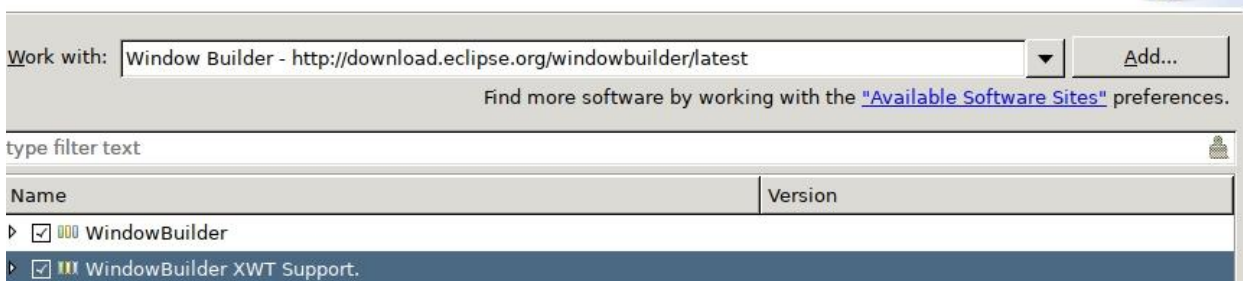
Select a site or enter the location of a site.



3. Eine Weile warten (zuerst kommt "Pending") und dann die Software auswählen:
Window Builder (alles auswählen); Swing Designer (alles auswählen)

Available Software

Check the items that you wish to install.



4. Danach unten auf "Next" klicken, nochmal auf "Next", dann die Lizenz akzeptieren und auf "Finish".
5. Die Software installiert; wenn die Installation beendet ist, bekommt man die Aufforderung Eclipse neu zu starten.