In dieser Starthilfe

- *Um was geht's?*
  - Grundstruktur einer SWT Applikation
- *Grundlegende Widgets*
  - Label, Text, Button, Listen, Combo, Composite
- *Events*
  - Selektion, Key, Mouse, Text, Focus & Traverse
- *Zusammenfassung*

Eclipse's SWT Basic Widgets

1.1. Um was geht's?

Im Folgenden wollen wir uns das Standard Widget Toolkit (SWT) genauer ansehen. Ich gehe davon aus, dass Sie SWT installiert und konfiguriert haben und das SWT Hello World Programm bereits getestet haben.

1.2. Grundstruktur einer SWT Applikation

Jede SWT Widget Applikation benötigt folgende Packages:

```
org.eclipse.swt.*;
org.eclipse.swt.widgets.*;
```

Dann können wir mit dem Aufbau der Applikation beginnen. Widgets verwenden Shells und Displays als Container für die Widgets:

```
Display display = new Display();
Shell shell = new Shell(display);
```

Alle GUI Komponenten sind im Display Objekt enthalten. Ein Display ist selber nicht sichtbar, lediglich die darin enthaltenen Komponenten.

- Pro Applikation wird typischerweise ein einziger Display kreiert.

Eine Shell ist ein Applikationsfenster.

- Pro Applikation können mehrere Shells existieren
- Shells können Top Level (am Display angehängt) oder verschachtelt sein (an einer Shell angehängt).

SWT Widgets kann man wie Swing Widgets initialisieren und modifizieren:

- `shell.setSize(100,100)`

Damit Shells auf dem Bildschirm sichtbar sind, müssen diese geöffnet werden und in einem Event Loop muss dauernd überprüft werden, ob die Shell noch aktiv ist:

```
shell.open();
while(!shell.isDisposed()){
      if(!display.readAndDispatch())
      display.sleep();
}
display.dispose();
```

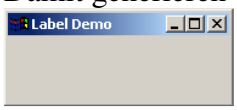
1.3. Grundlegende Widgets

Nun besprechen wir die grundlegenden Widgets, eines nach dem andern. Layout Manager vermeiden wir am Anfang, weil dadurch der Code nur komplexer würde.

Starten wir mit einem leeren Rahmen:

```
public class LabelDemo {

	public static Display labelDisplay;
	public static boolean internalCall = false;

	public static void main(String[] args) {
		internalCall = true;
		labelDisplay = new Display();
		LabelDemo ld = new LabelDemo();
		ld.runDemo(labelDisplay);
	}

	public void runDemo(Display display) {
		labelDisplay = display;
		Shell shell = new Shell(display);
		shell.setSize(300,300);
		shell.setText("Label Demo");
		shell.open();

		while(!shell.isDisposed()){
		if(!display.readAndDispatch())
			display.sleep();
		}
		if (internalCall) display.dispose();
	}
}
```

Damit generieren wir folgendes Fenster:



1.3.1. Label

Als erstes wollen wir lediglich ein Label auf eine Shell. Ein Label besteht aus Zeichen, welche vom Benutzer nicht modifiziert werden können.

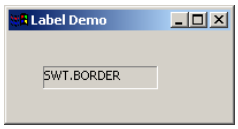
Programmfragment:

- `Label label_1 = new Label(shell, SWT.BORDER)`

Labels gibt es in verschiedenen Stylen:

- `BORDER, CENTER, LEFT, RIGHT, WRAP, SEPARATOR`

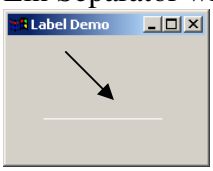
Wenn wir die obige Methode einfach durch den folgenden Code ergänzen:



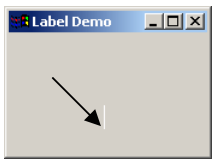

```
Label borderLabel = new Label(shell, SWT.BORDER);
borderLabel.setText("SWT.BORDER");
borderLabel.setSize(100,20);
borderLabel.setLocation(30,30);
```

erhalten wir die obige Ausgabe / Anzeige.

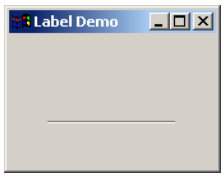
Ein Separator wird als Trennlinie angezeigt:




```
Label sep = new Label(shell, SWT.SEPARATOR| SWT.HORIZONTAL |
                                                SWT.SHADOW_IN);
sep.setBounds(30,60,100,20);
```




```
Label sep = new Label(shell, SWT.SEPARATOR| SWT.VERTICAL |
                                                SWT.SHADOW_IN);
sep.setBounds(30,60,100,20);
```

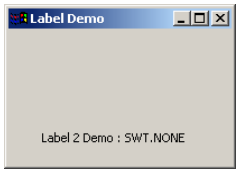



```
Label sep = new Label(shell, SWT.SEPARATOR| SWT.HORIZONTAL
                                        | SWT.SHADOW_OUT);
sep.setBounds(30,60,100,20);
```

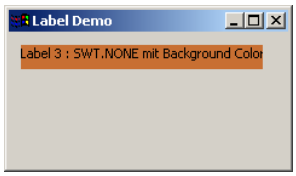
Ein Separator besitzt also verschiedene Styles:

- HORIZONTAL, VERTICAL, SHADOW_IN, SHADOW_OUT und SHADOW_NONE

Kommen wir zurück zu den einfachen Labels in verschiedenen Varianten.




```
Label label2 = new Label(shell, SWT.NONE);
label2.setText("Label 2 Demo : SWT.NONE");
label2.setSize(150,20);
label2.setLocation(30,90);
```




```
Label label3 = new Label(shell, SWT.NONE);
label3.setSize(100,20);
label3.setLocation(10,10);
label3.setBackground(new Color(display,200,111,50));
label3.setText("Label 3 : SWT.NONE mit Background
Color");
```

Und hier das gesamte Beispiel:

```
package basicsWidgets;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;
/**
* @author jjoller
 *
 * Label Demo zeigt, wie man das Label Widget in SWT
 * verwenden kann.
 * SWT = Standard Widget Toolkit von Eclipse
 */
public class LabelDemo {

	public static Display labelDisplay;
	public static boolean internalCall = false;
	public static void main(String[] args) {
		internalCall = true;
		labelDisplay = new Display();
		LabelDemo ld = new LabelDemo();
		ld.runDemo(labelDisplay);
	}
	public void runDemo(Display display) {
		labelDisplay = display;
		Shell shell = new Shell(display);
		shell.setSize(300,300);
		shell.setText("Label Demo");

		Label borderLabel = new Label(shell, SWT.BORDER);
		borderLabel.setText("Label 1 : SWT.BORDER");
		borderLabel.setSize(150,20);
		borderLabel.setLocation(30,30);

		Label sep = new Label(shell, SWT.SEPARATOR| SWT.HORIZONTAL |
		                             SWT.SHADOW_OUT);
		sep.setBounds(30,60,100,20);

		Label label2 = new Label(shell, SWT.NONE);
		label2.setText("Label 2 : SWT.NONE");
		label2.setSize(150,20);
		label2.setLocation(30,90);

		Label sep2 = new Label(shell, SWT.SEPARATOR | SWT.HORIZONTAL);
		sep2.setBounds(30,120,100,20);

		Label label3 = new Label(shell, SWT.NONE);
		label3.setSize(200,20);
		label3.setLocation(30,150);
		label3.setBackground(new Color(display,200,111,50));
		label3.setText("Label 3 : SWT.NONE mit Background Color");

		shell.open();

		while(!shell.isDisposed()){
		if(!display.readAndDispatch())
			display.sleep();
		}
		if (internalCall) display.dispose();
	}
}
```

1.3.2. Text

Ein Text Widget enthält Text, der normalerweise durch den Benutzer definiert wird / wurde.

- Text können Sie im Programm oder durch den Benutzer setzen und im Programm lesen.
- Text kann unlesbar gemacht werden (Passwort)
- Text kann nur lesbar sein (Anzeige)

Wir gehen wieder von einem fixen Rahmenprogramm aus und testen die unterschiedlichen Text-Optionen:


```
Text text1 = new Text(shell, SWT.BORDER);
text1.setText("Texteingabe Feld");
text1.setBounds(10,10,200,20);
text1.setTextLimit(30);
```

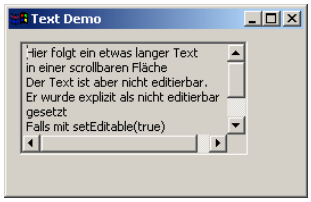
Auch beim Text Widget werden unterschiedliche Styles unterstützt:

- BORDER,
- H_SCROLL,
- V_SCROLL,
- MULTI,
- SINGLE,
- READ_ONLY und
- WRAP

Im obigen Beispiel wurde die maximale Anzahl Zeichen auf 30 festgelegt.
Falls Sie die Eingabe vor Unbefugten verstecken wollen, können Sie als Echo auf Ihre Eingabe ein fixes Zeichen festlegen. Dieses wird dann anstelle Ihrer Eingabezeichen sichtbar:


```
Text text2 = new Text(shell, SWT.NONE);
text2.setEchoChar('*');
text2.setBounds(10,10,200,20);
text2.setText("Passwort");
```

Falls der Text umfangreicher wird, benötigen Sie Scoll-Bars auf den Seiten des Textfeldes bzw. der Textfläche:




```
Text text3 = new Text(shell,
SWT.BORDER|SWT.H_SCROLL|SWT.V_SCROLL);
text3.setBounds(10,10,200,100);
text3.setEditable(false);
String langerText = "..";
text3.setText(langerText);
```

Der Text enthält “\n” für Zeilenumbrüche (new Line). Da er auf nicht editierbar

Den Text in einem Textfeld können Sie über abfragen:

```
// Abfrage des Textes im Textfeld (ohne Listener)
String strImText = text1.getText();
// Einfügen von Zusatztext
text1.insert("Hallo");
strImText = text1.getText();
```

Zur Vervollständigung, hier noch der komplette Programmcode:

```
public class TextDemo {

	public static Display myDisplay;
	public static boolean internalCall = false;

	public static void main(String[] args) {
		internalCall = true;
		myDisplay = new Display();
		TextDemo td = new TextDemo();
		td.runDemo(myDisplay);
	}

	public void runDemo(Display display) {
		myDisplay = display;
		Shell shell = new Shell(display);
		shell.setSize(300, 300);
		shell.setText("Text Demo");

		Text text1 = new Text(shell, SWT.BORDER);
		text1.setText("Texteingabe Feld");
		text1.setBounds(10, 10, 200, 20);
		text1.setTextLimit(30);

		// Abfrage des Textes im Textfeld (ohne Listener)
		String strImText = text1.getText();
		System.out.println("[TextLabel]Text = " + strImText);
		text1.insert("Hallo");
		strImText = text1.getText();
		System.out.println("[TextLabel]Text = " + strImText);

		Text text2 = new Text(shell, SWT.NONE);
		text2.setEchoChar('*');
		text2.setBounds(10, 50, 200, 20);
		text2.setText("Passwort");
		Text text3 = new Text(shell,
				SWT.BORDER|SWT.H_SCROLL|SWT.V_SCROLL);
		text3.setBounds(10,10,200,100);
		text3.setEditable(false);
		String langerText = "Hier ...\n"+"...";
		text3.setText(langerText);

		shell.open();

		while (!shell.isDisposed()) {
			if (!display.readAndDispatch())
				display.sleep();
		}
		if (internalCall)
			display.dispose();
	}
}
```

1.3.3. Button

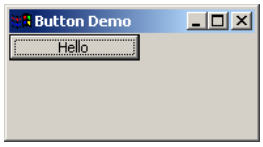
Ein Button ist ein Widget, welches vom Benutzer aktiviert wird, um irgend eine Verarbeitung auszulösen. Das grundsätzliche Konstrukt ist der Button; Buttons sind

- PUSH
- CHECK
- RADIO
- TOGGLE
- ARROW

Styles sind:

- FLAT
- BORDER
- LEFT
- RIGHT
- CENTER

Schauen wir uns einige Beispiele an.
Als erstes bauen wir uns einen HelloButton:




```
Button button1 = new Button(shell,SWT.PUSH);
button1.setText("Hello");
button1.setLocation(0,0);
button1.setSize(100,20);
```

Jetzt möchten Sie noch feststellen, ob der Button angeklickt wurde. Dazu benötigen Sie einen Listener, wobei Sie das package `org.eclipse.swt.events.*` importieren müssen.

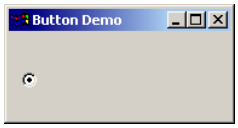
```
button1.addSelectionListener(new SelectionAdapter() {
      // anonyme Klasse
      // SelectionAdapter Methode
      public void widgetSelected(SelectionEvent e) {
            System.out.println("Button wurde geklickt");
      }
});
```

Wir fügen einen Selection Listener hinzu und definieren dessen Aktionen in einer anonymen Klasse, die wir gleich dahinter definieren (`new SelectionAdapter() {//anonyme Klasse…}`).

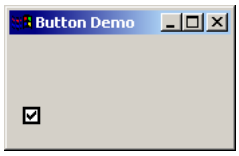
Mit zusätztlichen set-Methoden können Sie beispielsweise den Hintergrund setzen (Bild, Farbe, usw.)

Der einfachste Event Handler für Buttons ist der Event Handler. Dieser wird aktiviert, falls ein Button selektiert oder gedrückt wird.

Nun wollen wir noch kurz einige Varianten des Buttons zeigen:

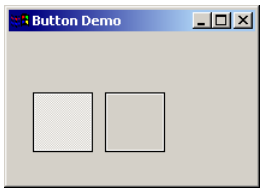



```
Button button2 = new Button(shell, SWT.RADIO);
button2.setSize(20,20);
button2.setLocation(10,30);
```




```
Button button3 = new Button(shell, SWT.CHECK|SWT.FLAT);
button3.setSize(20,20);
button3.setLocation(10,50);
```

Zwei identische Buttons, einmal geklicked, einmal neutral:



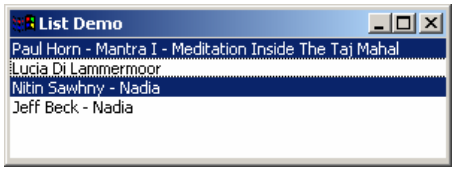

```
Button button4 = new Button(shell, SWT.FLAT|SWT.TOGGLE);
button4.setSize(50,50);
button4.setLocation(20,50);
Button button5 = new Button(shell, SWT.FLAT|SWT.TOGGLE);
button5.setSize(50,50);
button5.setLocation(80,50);
```

```
package basicsWidgets;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;
public class ButtonDemo {

	public static Display myDisplay;
	public static boolean internalCall = false;

	public static void main(String[] args) {
		internalCall = true;
		myDisplay = new Display();
		ButtonDemo bd = new ButtonDemo();
		bd.runDemo(myDisplay);
	}

	public void runDemo(Display display) {
		myDisplay = display;
		Shell shell = new Shell(display);
		shell.setSize(300,300);
		shell.setText("Button Demo");

		Button button1 = new Button(shell,SWT.PUSH);
		button1.setText("Hello");
		button1.setLocation(10,10);
		button1.setSize(100,20);
		button1.addSelectionListener(new SelectionAdapter() {
			public void widgetSelected(SelectionEvent e) {
				System.out.println("Button wurde geklickt");
			}
		});
…
		shell.open();

		while(!shell.isDisposed()){
		if(!display.readAndDispatch())
			display.sleep();
		}
		if (internalCall) display.dispose();
	}
}
```

1.3.4. Listen

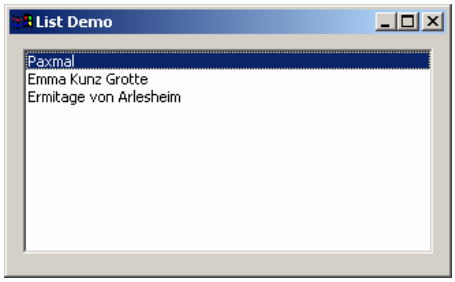
Eine Liste ist ein Widget, welches eine Sammling von Items enthält. Der Benutzer kann eines dieser Items auswählen. Schauen wir uns einige einfache Beispiele an. Der Rahmen ist der selbe, wie bei den obigen Programmen.

Als erstes bauen wir eine Liste mit mehreren gleichzeitig selektierbaren Texten:




```
List list1 = new List(shell,
SWT.MULTI|SWT.H_SCROLL);
list1.setItems(new String[] {"Paul Horn -
Mantra I - Meditation Inside The Taj
Mahal","Lucia Di Lammermoor","Nitin Sawhny -
Nadia"});
list1.add("Jeff Beck - Nadia");
list1.setBounds(0,0,360,80);
```

Beim zweiten Beispiel setzen wir einen andern Style, so dass lediglich nur noch ein Item ausgewählt werden kann. Bei Auswählen wir ein Maus Event ausgelöst und der Inhalt des selektierten Items ausgegeben:




```
final List list2 = new List(shell,
SWT.SINGLE|SWT.BORDER);
list2.setItems(new String[] {"Paxmal",
            "Emma Kunz Grotte",
            "Ermitage von Arlesheim"});
list2.setBounds(10,10,300,150);
list2.addMouseListener(new MouseAdapter() {
public void mouseDown(MouseEvent e) {
 System.out.println(list2.getSelection()[0] +
                    " wurde ausgewählt");
 }
 public void mouseUp(MouseEvent e) {
System.out.println("Versuchen Sie's nochmal!");
}
});
```

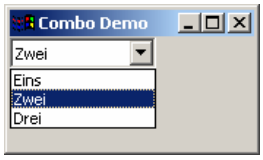
Weil wir auf die Variable `list2.getSelection()[0]` aus einer inneren Klasse zugreifen möchten, muss die Liste `list2` **`final`** deklariert sein.

1.3.5. Combo

Beim Combo Widget kann der Benutzer:

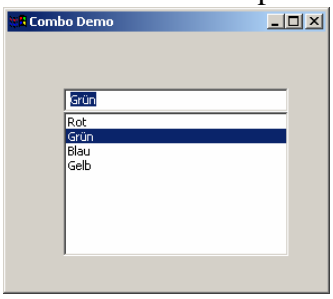
- ein Item aus der Collection auswählen
- eine Eingabe in das Combo Widget tätigen.

Als erstes bauen wir uns ein Combo mit mehreren fixen Einträgen und vom Drop Down Style:




```
Combo combo1 = new Combo(shell,
SWT.DROP_DOWN|SWT.READ_ONLY);
combo1.setItems(new String[] {"Eins","Zwei","Drei"});
combo1.select(0);
combo1.setLocation(0,0);
combo1.setSize(100,20);
```

Beim nächsten Beispiel wird das selektierte Item oben angezeigt:




```
Combo combo2 = new Combo(shell, SWT.SIMPLE);
combo2.setItems(new String[]
            {"Rot","Grün","Blau","Gelb"});
combo2.setBounds(50,50,200,150);
combo2.select(1);
```

Das dritte Beispiel ist ein einfaches Drop Down Combo:


```
Combo combo3 = new Combo(shell, SWT.DROP_DOWN);
combo3.setLocation(20,0);
combo3.setSize(50,50);
```

1.3.6. Composite

Composites sind Widgets, welche andere Widgets enthalten können. Sie können Widgets in einem andern Widget genau so platzieren, wie Sie ein Widget auf der Shell platzieren können. Die Position der Widgets relativ zu einander bleibt beim Verschieben eines Widgets erhalten.

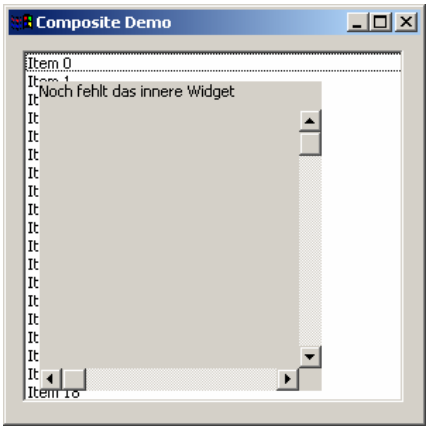
Schauen wir uns zuerst das äussere Widget an:


```
Label label = new Label(composite1,SWT.NONE);
label.setText("Noch fehlt das innere Widget");
label.setBounds(50,20,200,20);
```

Nun fügen wir ein weiteres Widget ein, Scrolling Bars horizontal und vertikal.
Zudem färben wir den Hintergrund ein:

```
composite1.setBackground(new Color(display,31,133,31));
Composite composite2 = new
      Composite(composite1,SWT.H_SCROLL|SWT.V_SCROLL);
composite2.setBounds(10,40,200,200);
```

Und schliesslich fügen wir dem untersten Widget noch eine Liste hinzu:




```
List list = new List(composite1,SWT.MULTI);
for (int i=0; i<50; i++) {
      list.add("Item " + i);
}
list.setSize(300,300);
```

Dadurch ändert sich die Farbe wieder! Wir haben das obige Widget (in grün) einfach zugedeckt.

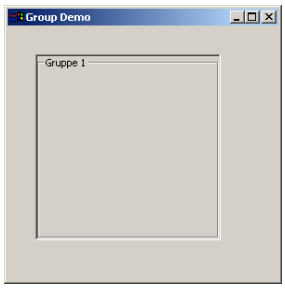
Farben können wir im SWT sehr einfach kreieren, falls Sie wissen wie das Ausgabe-Device bezeichnet wird:

```
import org.eclipse.swt.graphics.*;
Color myColor = new Color(display, 0, 0, 0);
```

1.3.7. Gruppen

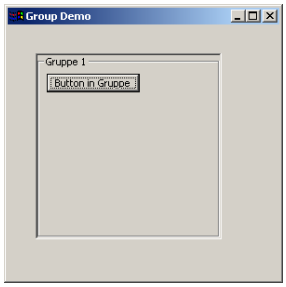
Gruppen sind Widgets, welche andere Widgets enthalten können. Gruppen besitzen einen Rand und optional auch einen Titel. Die Positionen der Widgets untereinander bleibt beim Verschieben unverändert.

Zuerst schauen wir uns eine einfache Gruppe an und füllen anschliessen weitere Widgets ein.



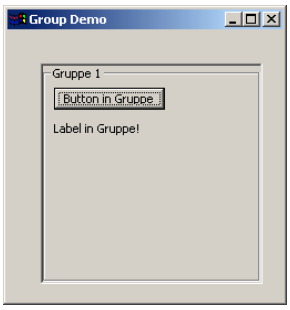

```
Group group1 = new Group(shell, SWT.BORDER);
group1.setBounds(30,30,200,200);
group1.setText("Gruppe 1");
```

Im nächsten Schritt fügen wir einen Push Button in diese Gruppe ein.



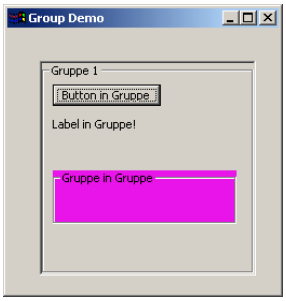

```
Button button = new Button(group1, SWT.PUSH);
button.setBounds(10,20,100,20);
button.setText("Button in Gruppe");
```

Jetzt setzen wir noch ein Label Widget in die erste Gruppe.




```
Label label = new Label(group1, SWT.NONE);
label.setBounds(10,50,80,20);
label.setText("Label in Gruppe!");
```

Aber wir können auch eine ganze Gruppe einfügen.




```
Group group2 = new Group (group1, SWT.NONE);
group2.setBounds(10,100,170,50);
group2.setBackground(new Color(display,233,20,233));
group2.setText("Gruppe in Gruppe");
```

Gruppen können in verschiedenen Styles benutzt werden:

- BORDER,
- SHADOW_ETCHED_IN,
- SHADOW_ETCHED_OUT,
- SHADOW_IN, SHADOW_OUT und SHADOW_NONE.

1.4. Events

Wir haben bereits ein Beispiel mit einem Maus Listener gesehen. Nun fassen wir die Eventsteuerung zusammen und gehen auf mehrere Events bzw. Listener ein.

1.4.1. SelectionListener

In SWT benötigt man zwei Dinge, um Events abzufangen:

- einen Listener, um spezielle Events abzufangen.
  Jeder Listener ist über ein Interface definiert (zum Beispiel `SelectionListener`)
- einer Event Klasse, welche die Informationen über das Event festhalten kann. (zum Beispiel `SelectionEvent`).

Typischerweise implementieren Sie die Methoden des Listeners in einer inneren oder anonymen inneren Klasse:

```
SelectionListener listener = new SelectionListener() {
      public void widgetSelected(SelectionEvent arg0) {
            System.out.println("Button wurde gedrückt");
      }
      public void widgetDefaultSelected(SelectionEvent arg0) {
      }
};
```

Falls das Interface mehrere Methodenbesitzt, kann anstelle des Interfaces ein Adapter kreiert werden (zum Beispiel `SelectionAdapter`). Der Adapter implementiert die Methoden, welche im Interface definiert werden. Damit können Sie Ihren Listener einfacher implementieren und nur jene Methoden, welche Sie wirklich brauchen:

```
SelectionAdapter adapter = new SelectionAdapter() {
      public void widgetSelected(SelectionEvent arg0) {
            System.out.println("Button Selected");
      }
};
```

Die Listener müssen einem Widget angehängt werden:

```
Button button = new Button(shell, SWT.PUSH);
button.addSelectionListener(listener);
```

Oder im Fall eines Adapters:

```
button.addSelectionListener(adapter);
```

Nun betrachten wir einige der Listener etwas genauer.

1.4.2. Key Listener

Der `KeyListener` besitzt zwei Methoden:

- `keyPressed` und
- `keyReleased`

Die Bedeutung dieser Methoden ist soweit klar.

Das Event Objekt ist `KeyEvent`. Der Adapter ist `KeyAdapter`.

Mit dem `KeyEvent` können Sie feststellen, welcher Key gedrückt wurde. Das Event besitzt ein Datenfeld mit dem Zeichen, welches gedrückt wurde. SWT enthält eine Tabelle mit den unterschiedlichen Keys, zum Beispiel `SWT.KEY, SWT.CR`.

Das folgende Codefragment (aus dem Beispiel `ListenerDemo`) zeigt, wie Sie spezielle Zeichen abfragen können:

```
Text text = new Text(shell, SWT.MULTI|SWT.BORDER);
text.setBounds(10,10,100,100);
text.addKeyListener(new KeyListener() {
      public void keyPressed(KeyEvent e) {
            String string = "";
            switch (e.character) {
                  case 0: string += " '\\0'"; break;
                  case SWT.BS: string += " '\\b'"; break;
                  case SWT.CR: string += " '\\r'"; break;
                  case SWT.DEL: string += " DEL"; break;
                  case SWT.ESC: string += " ESC"; break;
                  case SWT.LF: string += " '\\n'"; break;
                  default: string += " '" + e.character +"'";
                  break;
            }
            System.out.println (string);
      }
      public void keyReleased(KeyEvent e) {
            if (e.stateMask == SWT.CTRL && e.keyCode != SWT.CTRL)
            System.out.println("Command can execute here");
      }
});
```

Falls wir den Adapter benutzt hätten, könnten wir kompakteren Code schreiben, da die zweite Methode nicht implementiert werden müsste.

1.4.3. Mouse Listeners

Es gibt mehrere Maus Listener zum Überwachen und Auswerten der Mausaktivitäten.

1.4.3.1. MouseListener

Der MouseListener besitzt drei Methoden:

- `mouseDown`
- `mouseUp`
- `mouseDoubleClick`

Die Bedeutung sollte aufgrund der Namen klar sein. Da mehrere Methoden vorhanden sind, existiert auch ein Adapter, `MouseAdapter`.

Die eigentliche Information über das Event ist in der Klasse `MouseEvent` enthalten. Die Klasse enthält Informationen über die x, y Koordinaten, den Button und eine Zustandsmaske. Sie können beispielsweise abfragen, wie viele Buttons gedrückt oder losgelassen wurden (Button 1 liefert ‚1', Button 2 eine ‚2' usw.).

Im folgenden Beispiel testen wir den Maus Adapter. Dabei verwenden wir zwei Buttons, stellen im Programm fest, welcher Button gedrückt wurde und wo. Das Doppelklicken öffnet ein Fenster (`MausListenerDemo`):

```
Button button = new Button(shell, SWT.PUSH);
button.setText("Klicken!");
button.setBounds(10, 10, 60, 20);
Button button2 = new Button(shell, SWT.PUSH);
button.setText("Auch Klicken!");
button.setBounds(100, 10, 60, 20);
MouseAdapter mouseAdapter = new MouseAdapter() {
	public void mouseDown(MouseEvent e) {
	System.out.println("Button " + e.button + " wurde gedrückt");
		System.out.println(
		"Die Maus wurde bei (" + e.x + "," + e.y + ") gedrückt");
	}
	public void mouseUp(MouseEvent e) {
		System.out.println(
		"Die Maus wurde bei (" + e.x + "," + e.y + ") gedrückt");
	}
};
button.addMouseListener(mouseAdapter);
button2.addMouseListener(mouseAdapter);
Label label = new Label(shell, SWT.NONE);
label.setBounds(10, 40, 100, 20);
label.setText("Doppelkllicken!");
label.addMouseListener(new MouseAdapter() {

	public void mouseDoubleClick(MouseEvent e) {
		Shell shell2 = new Shell(buttonDisplay);
		shell2.setSize(200, 100);
		shell2.setText("Doppelklick Fenster");
		Label label2 = new Label(shell2, SWT.NONE);
		label2.setText("Hallo (neues Fenster)!");
		label2.setBounds(0, 10, 150, 50);
		shell2.open();
	}
});
```

1.4.3.2. Mouse Move Listener

Der MouseEventListener besitzt eine Methode, mouseMove, Diese wird ausgeführt, sobald die Maus über das Kontrollelement geschoben wird, an dem der Listener angehängt ist. Sie finden ein Beispiel weiter unten.

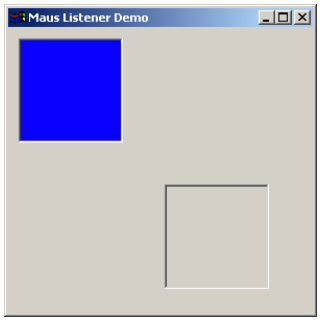
1.4.3.3. Mouse Track Listener

Der MouseTrack Listener besitzt drei Zustände:

- `mouseEnter`
- `mouseExit`
- `mouseHover`

`mouseEnter` wird ausgelöst, wenn die Maus in die Fläche eintritt, an die der Maus Listener angehängt wurde; analog bei `mouseExit`. `mouseHover` wird aktiviert, wenn die Maus über die Kontrollfläche „hoovered“ / sich darüber bewegt.

Als Beispiel bauen wir eine Demo mit zwei Canvas. Wenn die Maus sich über ein Canvas fährt, soll sich die Farbe ändern.




```
final Canvas canvas = new Canvas(shell, SWT.BORDER);
canvas.setBounds(10, 10, 100, 100);
Canvas canvas2 = new Canvas(shell, SWT.BORDER);
canvas2.setBounds(150, 150, 100, 100);
final Text text = new Text(canvas2, SWT.READ_ONLY);
text.setBounds(40, 30, 20, 20);

MouseMoveListener mouseMove = new MouseMoveListener()
{
	public void mouseMove(MouseEvent e) {
		Color color = canvas.getBackground();
		color.dispose();
		canvas.setBackground(new Color(buttonDisplay, count1, 0, 255));
		count1++;
		count1 = count1 % 255;
	}
};
canvas.addMouseMoveListener(mouseMove);
MouseTrackListener mouseTrack = new MouseTrackListener() {
	public void mouseEnter(MouseEvent arg0) {
		text.setText(Integer.toString(count2));
	}
	public void mouseExit(MouseEvent arg0) {
		text.setText("");
		count2 = 0;
	}
	public void mouseHover(MouseEvent arg0) {
		count2++;
		text.setText(Integer.toString(count2));
	}
};
canvas2.addMouseTrackListener(mouseTrack);
```

1.4.4. Text Listeners

Text Widgets können mithilfe von Text Listener gesteuert werden:

- `ModifyListener`
- `VerifyListener`

`ModifyListener` werden aufgerufen, falls der Text modifiziert wird.
`VerifyListener` werden aufgerufen, bevor die Änderung durchgeführt wird.

1.4.4.1. ModifyListener

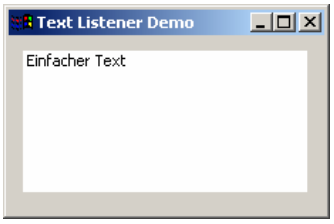
Der `ModifyListener` besitzt nur eine Methode, `modifyText()`. Das `ModifyEvent` enthält Informationen über das Widget.

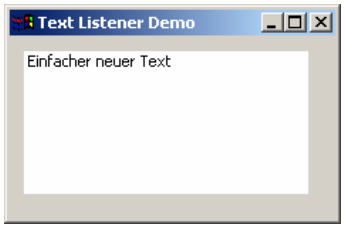
1.4.4.2. VerifyListener

Der `verifyListener` besitzt die Methode `verifyText()`. Das verifyEvent Objekt besitzt alle Eigenschaften von `TypedEvent`, `KeyedEvent` und die Felder

- `start`
- `end`
- `doit`
- `text`

`start` und `end` bezeichen sich auf den Text (Anfangs und End-Position). `text` ist der geänderte Text.






```
Text text = new Text(shell, SWT.MULTI | SWT.WRAP);
text.setBounds(10, 10, 200, 100);
text.setText("Einfacher Text");
text.addModifyListener(new ModifyListener() {
      public void modifyText(ModifyEvent e) {
      System.out.println("Modifiziert um " + e.time);
      }
            });
text.addVerifyListener(new VerifyListener() {
      public void verifyText(VerifyEvent e) {
      if (e.text.equals("*")) {
      System.out.println("Kann * nicht tippen");
      e.doit = false;
      }
}
});
```

Bei der Ausführung wird im obigen Fall folgene Konsolenausgabe generiert:

```
Modifiziert um 3442710
Modifiziert um 3443201
Modifiziert um 3443541
Modifiziert um 3443711
Modifiziert um 3443992
Modifiziert um 3444432
Kann * nicht tippen
```

1.4.5. Focus Listeners & TraverseListener

Der `FocusListener` wird aktiviert, falls ein Widget den Fokus erhält, also ausgewählt oder angeklickt wird oder sonst wie aktiv wird.

Der `TarverseListener` wird generiert, falls das Widget in den Fokus gerät, weil es traversiert wird.

Der `FocusListener` besitzt zwei Methoden:

- `focusGained`
- `focusLost`

1.4.5.1. TraverseListener

Der `TraverseListener` besitzt nur die Methode `keyTraversed()`. Das entsprechende Event Objekt ist `TraverseEvent`. Dieses enthält nützliche Zusatzinformationen, wie:

- `doit`
- `detail`

`doit` entsprecht jenem im `VerifyListener`.

1.5. Zusammenfassung

In diesem Teil des Tutorials haben wir verschiedene einfache Widgets kennen gelernt und mithilfe der Listener, der Eventsteuerung, nutzen gelernt.

Im nächsten Tutorial gehen wir auf komplexere Widgets ein.

Referenz:
http://download.eclipse.org/downloads/documentation/2.0/html/plugins/org.eclipse.platform.doc.isv/reference/api/