

3 Die Kunst der (visuellen) Komposition

Benutzer, die noch in Eclipse 3.2 den in der *Callisto*-Distribution enthaltenen *Eclipse Visual Editor* (VE) verwendet hatten, werden in der *Europa*-Distribution von Eclipse 3.3 vergeblich nach diesem Produkt suchen: Es wurde nicht mehr weiterentwickelt. Der Grund war wohl, dass es inzwischen leistungsfähige kostenlose Angebote von kommerziellen Anbietern wie *WindowBuilder* von *Instantiations* (www.instantiations.com) oder *Jigloo* von *CloudGarden* (www.cloudgarden.com)¹ gibt.

Visual Editor for Java

Aus den genannten Gründen werde ich von meiner bisherigen Praxis abweichen und an dieser Stelle ein kommerzielles Produkt (*WindowBuilder*) vorstellen. Sie erhalten *WindowBuilder* über <http://www.instantiations.com/windowbuilderpro/default.htm>. Sie können entweder den angebotenen Installer verwenden (empfohlen) oder aber eine ZIP-Datei herunterladen, müssen aber dann unbedingt den Installationsanweisungen folgen. Ich habe hier *WindowBuilder* in der Version 6.4.1 verwendet.

Bevor der *WindowBuilder* benutzt werden kann, muss er registriert und aktiviert werden. Dabei haben Sie die Wahl zwischen einer zeitlich befristeten Evaluationslizenz mit dem vollen Funktionsumfang oder einer unbefristeten kostenlosen Lizenz mit deutlich beschnittenem Funktionsumfang, der aber dennoch – wie schon erwähnt – weit über die Funktionalität des *Eclipse Visual Editor* hinausgeht. Die Registrierung erfolgt in Eclipse unter *Window>Preferences>Designer*.

Nach der Installation und Freischaltung sieht man erst einmal wenig vom *WindowBuilder*. In der Eclipse-Hilfe findet man jedoch nun als neues Hilfebuch den *Designer Guide*. Außerdem gibt es unter *File>New>Other...>Designer>...* eine ganze Reihe zusätzlicher *New-Wizards*, darunter auch eine Gruppe für Swing. Mit diesen Wizards

Aufruf

1. Allerdings sind beide Produkte nur auf Windows- und Linux-Plattformen lauffähig.

kann eine neue visuelle Klasse angelegt werden. Neben AWT- und Swing-Behältern wie `JApplet`, `Window`, `JDialog`, `JPanel` und `JFrame` stehen auch je nach Lizenz SWT- und JFace-Behälter (siehe Abschnitt 8.5) wie `Composite`, `Shell`, `Window` und `Dialog` zur Verfügung. Außerdem besteht in der Gruppe *RCP* die Möglichkeit, für die Plugin- und Rich-Client-Entwicklung komplette Workbench-Editoren und -Views (siehe Abschnitte 11.5.6 und 11.5.7), Menüs (siehe Abschnitt 14.3.4) und weitere Konstrukte zu bearbeiten.

Anders als beim *Eclipse Visual Editor* ist es jedoch nicht unbedingt erforderlich, visuelle Klassen auf diese Weise explizit anzulegen. Der *WindowBuilder* ist in der Lage, beim Öffnen einer Java-Quelldatei deren Code zu analysieren. Befinden sich visuelle Elemente im Code, wird automatisch statt dem Java-Editor der *WindowBuilder* verwendet. Man erkennt das daran, dass sich nun am unteren Rand des Editorbereichs die Reiter *Source* und *Design* befinden. Änderungen können freizügig sowohl im Quellcode (*Source*) als auch im visuellen Layout (*Design*) gemacht werden, sie werden beim Umschalten in den jeweils anderen Bereich reflektiert. Im Designbereich finden Sie oberhalb des eigentlichen Layout-Bereichs eine Werkzeugleiste. Mit der allerersten Taste *Quickly test/preview the window without compiling or running it* kann das erzeugte GUI ausprobiert werden. (Die gleiche Funktion erscheint auch in der Werkzeugleiste der Workbench unter dem neuen Button *Test Designer Window*.) Rechts neben der Taste befindet sich ein Auswahlfeld, mit dem für Swing der zu verwendende *Look&Feel* bestimmt werden kann.

3.1 Einstellungen

Der *WindowBuilder* kann unter *Window>Preferences>Designer* auf vielfältige Weise konfiguriert werden. So kann z.B. im Unterabschnitt *Code Generation* festgelegt werden, in welcher Form Code generiert werden soll. Im Unterabschnitt *Editor Layout* hat mir besonders die Option *Show Thumbnail in Outline View* gefallen. Ist diese Option gewählt, wird beim nächsten Öffnen der Datei im *Outline*-View ein verkleinertes Abbild des Designbereichs gezeigt. Insbesondere dann, wenn das Layout größer als der verfügbare Editorbereich ist, ist diese Option sinnvoll, denn dann kann im Thumbnail durch Verschieben des dargestellten Sichtfensters auf einfache Weise navigiert werden.

Für die Arbeit mit Swing kann im Unterabschnitt *Swing* das Standard-Layout gewählt werden. Außerdem können die einzelnen Swing-Layout-Manager individuell konfiguriert werden.

3.2 Komposition

Die eigentliche Komposition eines GUI verläuft recht einfach. Auf der linken Seite des Editorbereichs befindet sich die Palette mit den GUI-Elementen, die mit einem Klick auf die Pfeiltaste links oben auch minimiert werden kann. Sie können diese Palette auch vom rechten Rand abreißen und stattdessen am linken Rand andocken. Die Palette selbst ist in Gruppen aufgeteilt. Je nach gewählter GUI-Architektur (Swing oder SWT) erscheinen z.B. für Swing die Gruppen *Swing Containers*, *Swing Layouts* und *Swing Controls*. Mit einem Klick auf eine Gruppe wird diese expandiert, wobei andere bereits geöffnete Gruppen wieder geschlossen werden, wenn der Platz in der Palette nicht reicht. Mit einem Klick auf den Pin rechts neben dem Gruppennamen können Sie eine Gruppe auch auf Dauer geöffnet halten (Abb. 3–1).

Um ein GUI-Element in den Designbereich zu bringen, wählen Sie es zunächst per Mausklick aus. Dann klicken Sie im Designbereich den Zielort an. Es wird also nicht per Drag&Drop gearbeitet, sondern mit Klick auf Quelle und Ziel, ähnlich wie beim Spiel *FreeCell* beim Umliegen einer Karte.

Versuchen wir es einmal mit einem kleinen Beispiel. Erzeugen Sie im Projekt `HelloWorld` wie oben beschrieben eine neue visuelle Klasse `HelloGUI` mit Hilfe der Funktion `File>New>Other...>Designer>Swing>JFrame`. Im sich nun öffnenden Editor klicken Sie auf den Reiter *Design*. Nun erscheint im Designbereich eine `JFrame`-Instanz in einer vorgegebenen Standardgröße. Klicken Sie nun diese Komponente an der Ecke rechts unten an und ziehen Sie sie auf die gewünschte Größe. Wie Sie im Fenster *Property Editor* sehen, enthält die `JFrame`-Instanz bereits (wie bei Swing üblich) ein `JContentPane`-Objekt vom Typ `JPanel`, welches die gesamte Fläche der `JFrame`-Instanz ausfüllt.

HelloWorld

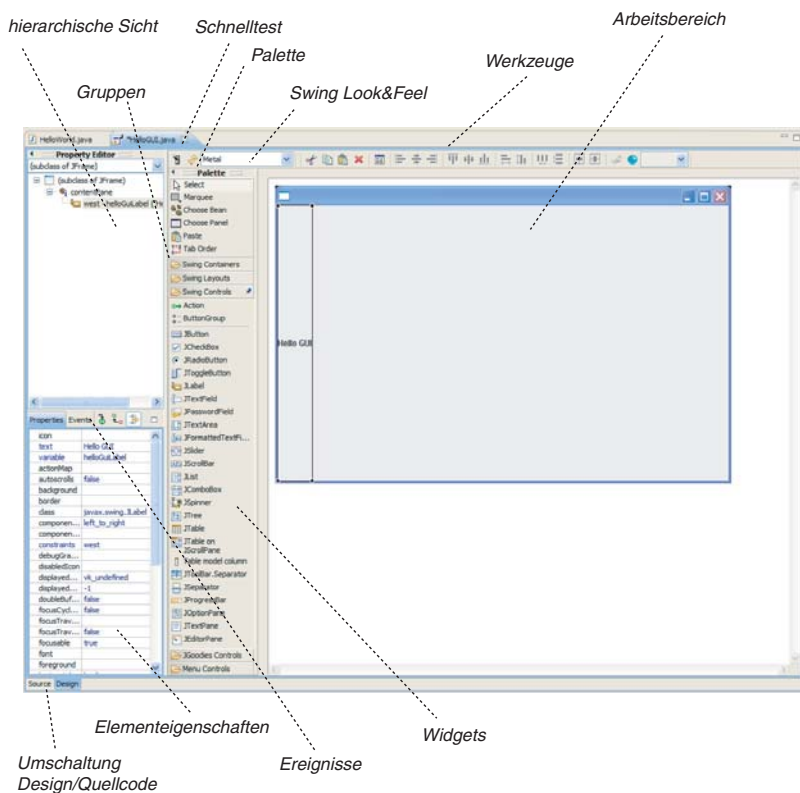


Abb. 3-1 Die Komponenten des Designers. Oben links sehen Sie den *Property Editor*, darunter das *Properties-* und das *Event-Fenster*, rechts daneben die *Palette* mit den GUI-Elementen. Rechts daneben befindet sich der *Designbereich*. Unter dem gesamten Designer, verdeckt auf einer weiteren Seite des Pultordners, befindet sich der *Java-Editor* mit dem Quellcode.

Palette Nun öffnen Sie in der *Palette* mit einem Klick die Gruppe *Swing Controls*. Dort selektieren Sie die Komponente *JLabel* und lassen die Maustaste wieder los. Nun bewegen Sie die Maus über die noch selektierte *JFrame*-Komponente im *Designbereich*. Dort erscheinen nun mit *North*, *Center*, *South*, *East* und *West* beschriftete Bereiche, denn die *ContentPane* ist bereits mit einem *BorderLayout* ausgerüstet (dazu später mehr). Legen Sie die *JLabel*-Komponente mit einem Klick im Bereich *Center* ab.

Es erscheint nun eine *JLabel*-Instanz mit der Aufschrift »New JLabel«. Es kann sein, dass diese Instanz nicht genau am gewählten Zielpunkt positioniert wird, denn der *BorderLayout*-Manager sorgt für eine automatische Positionierung. Im *Properties-Fenster*, das alle Eigenschaften des gerade selektierten GUI-Elements zeigt, überschreiben Sie nun in der Reihe *text* den Text »New JLabel« durch »Hello GUI«. Das Resultat sehen Sie, wenn Sie auf den Reiter *Source* klicken:

```
public class HelloGUI extends JFrame {

    /**
     * Launch the application
     * @param args
     */
    public static void main(String args[]) {
        try {
            HelloGUI frame = new HelloGUI();
            frame.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Create the frame
     */
    public HelloGUI() {
        super();
        setBounds(100, 100, 668, 437);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        final JLabel helloGuiLabel = new JLabel();
        helloGuiLabel.setText("Hello GUI");
        getContentPane().add(helloGuiLabel, BorderLayout.WEST);
        //
    }
}
```

Sie können nun dieses GUI sofort mit einem Klick auf die Werkzeug-
taste *Test Designer Window* oder mit *Strg+Alt+T* testen.

Testen

3.3 Layouts

Selektieren Sie nun im *Property Editor*-Fenster die Komponente *contentPane*. Wie alle Swing-Container besitzt auch dieser Container ein *Layout*². Gehen Sie nun im *Properties*-Fenster zur Eigenschaft *layout*. Dort sehen Sie, dass bereits *BorderLayout* eingetragen ist. Mit einem Klick auf *BorderLayout* und einem weiteren auf die Pfeiltaste haben Sie die Möglichkeit, andere Swing-Layout-Manager auszuwählen. Links neben dem *layout*-Eintrag sehen Sie ein Plus-Zeichen. Mit einem Klick auf dieses Zeichen können Sie diesen Eintrag expandieren und weitere Einstellungen für das gewählte Layout vornehmen, z.B. Angaben für

2. Zur Information über Swing und Swing-Layout-Manager sei auch hier wieder auf [Robinson2000] verwiesen.

horizontal gap und vertical gap machen. Schließlich gibt es noch die Möglichkeit, ganz auf einen Layout-Manager zu verzichten. Dazu wählen Sie aus der Auswahlliste die Option `Absolute (null) Layout`. Mit dieser letzteren Option können Sie die `JLabel`-Komponente innerhalb der `contentPane` frei positionieren.

In der Regel sollten Sie die `layout`-Einstellungen erst ändern, nachdem Sie einen Container mit Komponenten gefüllt haben. Denn bei bestimmten Layouts kann ein leerer Container die Größe null haben, und dann ist es schwierig, diesen Container mit Komponenten zu füllen. Freilich gibt es in diesem Fall immer noch einen Ausweg: Statt die Komponente im Designbereich im Container abzulegen, können Sie diese Operation auch im *Property Editor*-Fenster durchführen.

Drag&Drop

Wollen Sie später Komponenten an einer anderen Stelle positionieren oder gar in einen anderen Container bringen, so ist das kein Problem: Sowohl im Designbereich als auch innerhalb des *Property Editor*-Fensters können Sie Komponenten mittels `Drag&Drop` verschieben. Auch das Einfügen aus der Zwischenablage (`Cut/Copy&Paste`) ist möglich.

3.4 Ereignisverarbeitung

Sehen wir uns zum Schluss noch an, wie mit Hilfe des `WindowBuilder` eine Ereignisverarbeitung programmiert werden kann. Dazu bringen wir zunächst eine weitere Komponente, einen `JButton`, auf die `JFrame`-Komponente. Sie können die Position frei wählen, z.B. im Bereich `South` (falls Sie noch das `BorderLayout` verwenden). Sollten Sie den Layout-Manager abgeschaltet haben, verkleinern Sie die `JLabel`-Komponente etwas, so dass sie nicht die neu erzeugte `JButton`-Komponente überdeckt.

Anschließend suchen Sie sich im *Properties*-Fenster den Eintrag `text` und tragen dort »OK« ein. Alternativ können Sie natürlich auch die erzeugte Taste einfach anklicken und im Texteingabefenster dann »OK« eingeben.

Nun können Sie eine Ereignisverarbeitung für die neue Taste definieren. Bringen Sie dazu das unter dem *Properties*-Fenster liegende *Events*-Fenster durch einen Klick auf den Reiter *Events* nach vorn. Dort expandieren Sie den Eintrag `action`. Im Untereintrag `actionPerformed` führen Sie auf das Eingabefeld einen Doppelklick aus. `WindowBuilder` schaltet damit in den Quelltext um, wo bereits ein anonymer innerer `ActionListener` generiert wurde. Sie müssen nur noch in die Methode `actionPerformed()` Ihre Anwendungslogik eintragen.

```
public HelloGUI() {
    super();
    setBounds(100, 100, 668, 437);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    final JLabel helloGuiLabel = new JLabel();
    helloGuiLabel.setText("Hello GUI");
    getContentPane().add(helloGuiLabel, BorderLayout.WEST);

    final JButton okButton = new JButton();
    okButton.addActionListener(new ActionListener() {
        public void actionPerformed(final ActionEvent e) {
        }
    });
    okButton.setText("OK");
    getContentPane().add(okButton, BorderLayout.SOUTH);
    //
}
```

Wollen Sie die Ereignisverarbeitung testen, so reicht freilich dafür der Schnelltest mit der Taste *Test Designer Window* nicht mehr. Stattdessen müssen Sie die Klasse z.B. aus dem Quelltext heraus mit *Run as>Java Application* ausführen.

Damit sind wir am Ende unserer kurzen Einführung in den *WindowBuilder*. In Abschnitt 5.4.4 werden wir ein komplexeres GUI im Rahmen einer größeren Anwendung mit Hilfe des *WindowBuilder* realisieren. Für unsere Zwecke reicht die kostenlose Version des *WindowBuilder* völlig aus. Benötigt man jedoch Features wie die Erstellung mehrsprachiger GUIs oder gemischter Swing- und SWT-GUIs oder Unterstützung für komplexe Workbench-Komponenten und Erweiterungsfähigkeit für eigene Widgets, so ist man dann doch auf eine volle (und kostenpflichtige Version) des *WindowBuilder* angewiesen. Wer das vermeiden will, kann den ähnlich leistungsfähigen Designer *Jigloo* (siehe oben) verwenden, der allerdings nur bei nichtkommerziellem Einsatz kostenlos ist.

Resümee

Ich muss zugeben, dass ich bei der praktischen Arbeit kaum auf GUI-Designer zurückgreife. So ersparen diese Designer nicht präzise Kenntnisse des Swing- oder SWT-API. Die wenigsten Eintragungen erfolgen in der Tat visuell, sondern in Tabellenfeldern im *Properties*-Fenster – und hier muss man wissen wo. Verwendet man z.B. *LayoutManager*, was die Regel ist, so ist eine genaue Kenntnis der einzelnen Parameter nötig. Kennt man aber diese Parameter, so ist man im Quellcode bei intensiver Verwendung des Eclipse-Inhaltsassistenten oft schneller. Allerdings unterstützen die GUI-Designer auch diese Arbeitsweise und erlauben den leichten Wechsel zwischen codeorientierter und visueller Arbeitsweise.