

Errata und Addenda

für

Java Entwicklung mit Eclipse 3

Stand 4.1.2006

Korrekturen mit **schwarzen** Seitenzahlen sind in der Neuauflage von *Java Entwicklung mit Eclipse 3* bereits berücksichtigt. Diese Auflage trägt den Vermerk „Korrigierte Nachdruck 2005“. **Rote** Seitenzahlen dagegen zeigen neue Korrekturen an.

Seite 15:

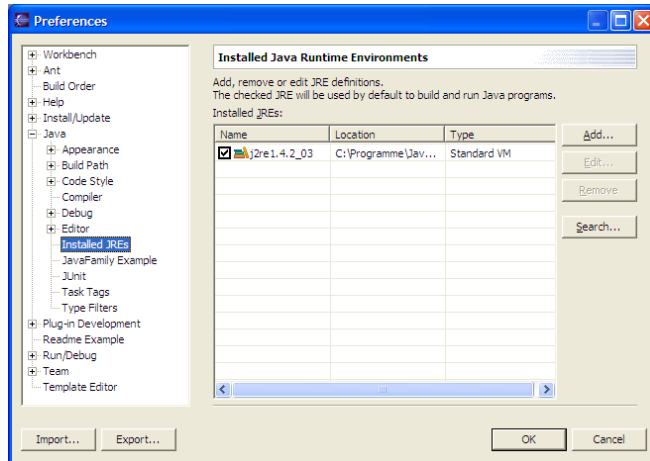
Auf der Toolbar (Abb. 1–6) klicken wir das ~~Create a~~**New** Java Project-Icon an.

Seite 16:

(~~Create a~~**New** Java Class)

Seite 21:

Abbildung 1-11 ist falsch und muss durch folgende Abbildung ersetzt werden:

**Seite 25:**

Schneller geht's freilich mit dem Tastenkürzel
Strg+Umschalt+F (unter Linux mit *Esc Strg+F*).

Seite 33:

Man muss die *Scrapbook*-Seite nicht vor einer Ausführung speichern – die Kompilierung erfolgt unmittelbar durch den *Run Snippet Execute*-Befehl.

Seite 34:

Das Ergebnis zeigt Eclipse die Fehlermeldung zunächst in einem Popup-Fenster. Auf Wunsch kann man mit *Strg+Shift+D* die Fehlermeldung direkt unter oder über den fehlerhaften Ausdruck einfügen. bzw. die Fehlermeldung zeigt Eclipse direkt hinter dem eingegebenen Ausdruck an.

Seite 45:

Strg+1 funktioniert manchmal auch, wenn gar kein Fehler vorliegt.

Seite 45:

Stellt man z.B. die Schreibmarke auf den Parameter einer Methodendefinition und drückt *Strg+1* oder klickt die grüne Glühbirne, die auf dem linken Editorrand erscheint, so erhält man verschiedene Funktionen zur Auswahl, darunter auch die Funktion *Assign parameter to new field*.

Seite 55:

... , wo sie nur auf das selektierte Element (z.B. eine Methode) wirkt.

Seite 57:

Inzwischen ist die Version 1.0 des VE freigegeben. Die Bedienung entspricht weitgehend der im Buch geschilderten Bedienung der Version M1 – auch bei SWT-Anwendungen. Neue visuelle Java-Klassen lassen sich mit *File>New>Visual Class* erzeugen. Die Art der Klasse lässt sich dabei im folgenden Wizard im Feld *Style* bestimmen. Man kann dort zwischen einer SWT-Applikation (siehe Kapitel 8) und verschiedenen Swing- und AWT-Containern wählen. SWT- und Swing-Komponenten können im VE nicht gemischt werden, hier ist man auf manuelle Programmierung angewiesen (siehe Abschnitt 8.8). Es empfiehlt sich, das SWT-JAR nicht manuell dem Projekt hinzuzufügen, sondern diese Arbeit automatisch vom VE erledigen zu lassen.

Seite 61:

Ausführliche Informationen zum Schreiben von Beans finden Sie in [Steams2000].

Seite 99:**Die Klasse AnimatedDiphoneVoice.java:**

```
package com.sun.speech.freetts.en.us;
import java.io.IOException;
import java.net.URL;
import java.util.Locale;
import com.sun.speech.freetts.Age;
import com.sun.speech.freetts.Gender;
import com.sun.speech.freetts.UtteranceProcessor;
import com.sun.speech.freetts.relp.AnimatedAudioOutput;
```

Seite 101:

Dann öffnen wir im [PackageProjekt](#) FreeTTS die Klasse `JavaClipAudioPlayer` und selektieren mit *Strg+A* den gesamten Text.

Seite 111:

```
// Die Größe des PlayerPanels für Player übernehmen  
setSize(playerPanel.getContentPane().getSize());
```

Seite 120:

Setzen wir z.B. einen Breakpoint, wie in Abbildung 6–2 gezeigt, auf die Anweisung `Dimension d = getSize()` in Methode `paintComponent()` in Klasse `PlayerFace`.

Seite 150:

Wir werden die Ressourcenverwaltung ausführlich in Abschnitt 8.911 diskutieren.

Seite 160:

Für andere Betriebs- und Fenstersysteme gilt Ähnliches.¹ Eine einfachere Art und Weise, einem Projekt die SWT- und gegebenenfalls auch die JFace-Funktionalität hinzuzufügen, besteht in der Möglichkeit, statt der Taste *Add External JARs...* die Taste *Add Library...* zu betätigen. Aus der folgenden Liste wählt man dann *Standard Widget Toolkit (SWT)* aus. Eclipse fügt dann alle benötigten JAR-Dateien dem Build Path hinzu.

Seite 204:

Außerdem verfügt die Klasse `SWT_AWT` noch über die Methode `new_Shell()`. Diese erzeugt für einen gegebenen AWT-Canvas eine gleich große SWT-Shell, ~~so dass der AWT-Canvas zwar in einem eigenen Fenster, aber noch innerhalb der SWT-Applikation ausgeführt wird.~~ Diese Shell kann dann weitere SWT-GUI-Elemente aufnehmen, die so in eine AWT-Umgebung eingebettet werden können.

Seite 205:

```
import java.awt.RenderingHints;
import java.util.ArrayList;
import java.util.Iterator;
import org.eclipse.swt.SWT;
import org.eclipse.swt.awt.SWT_AWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
public class SWT2D {
```

Seite 235:

Dies geschieht mit Hilfe von `IPositionUpdater`-Instanzen, von denen ebenfalls beliebig viele einem Dokument zugefügt werden können.

Seite 241:

Inhaltsassistenten werden von einer `SourceViewerConfiguration`-Implementierung mittels der `getContentAssistant()`-Methode an den `SourceViewer` übergeben. Dabei folgen `Formatierer` **Inhaltsassistenten** dem Interface `IContentAssistant`.

Seite 244:

Eingabefelder, die diese Funktionalität nutzen möchten, müssen das Interface `IContentAssistSubject` **Control** implementieren.

Seite 284 ff:**Die Klasse `DescriptionWindow`**

```
package com.bdaum.jukebox;
```

```
import org.eclipse.jface.viewers.ISelectionChangedListener;  
import org.eclipse.jface.viewers.IStructuredSelection;  
import org.eclipse.jface.viewers.SelectionChangedEvent;  
import org.eclipse.jface.window.Window;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.layout.FillLayout;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.swt.widgets.Control;  
import org.eclipse.swt.widgets.Shell;  
/**  
 * @author Berthold Daum  
 *  
 * Diese Klasse implementiert das Fenster für die Anzeige der Playlist.  
 */  
  
public class PlaylistWindow extends Window  
----- implements ISelectionChangedListener {  
    PlaylistViewer viewer;  
    Player player;  
    IPlaylist model;  
  
import java.util.StringTokenizer;  
  
import org.eclipse.jface.window.Window;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.browser.Browser;  
import org.eclipse.swt.layout.FillLayout;  
import org.eclipse.swt.layout.GridData;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.swt.widgets.Control;  
import org.eclipse.swt.widgets.Display;  
import org.eclipse.swt.widgets.Shell;  
  
public class DescriptionWindow extends Window {
```

```
// Das Browser-Widget
private Browser browser;
// Das Playlist-Modell
private IPlaylist model;
// Das aktuelle Display
private Display display;
/**
 * Konstruktor.
 *
 * @param parent - die übergeordnete Shell
 * @param player - der Player
 */
public PlaylistWindow(Shell parent, IPlaylist model) {
    super(parent);
    this.model = model;
}
/**
 * Konstruktor.
 *
 * @param parent - die übergeordnete Shell
 * @param model - das Player-Modell
 */
public DescriptionWindow(Shell parent, IPlaylist model) {
    super(parent);
    display = parent.getDisplay();
    this.model = model;
}
protected Control createContents(Composite parent) {
    parent.setLayout(new FillLayout());
    Composite composite = new Composite(parent, SWT.NONE);
    composite.setLayout(new FillLayout());
}
```

```
viewer = new PlaylistViewer(composite, SWT.SINGLE  
| SWT.VERTICAL | SWT.H_SCROLL |  
SWT.V_SCROLL  
| SWT.BORDER | SWT.FULL_SELECTION, model);  
// Ereignisverarbeitung für Selektionsänderungen im  
// Playlist-Viewer  
viewer.addSelectionChangedListener(  
new ISelectionChangedListener() {  
public void selectionChanged(  
SelectionChangedEvent e) {  
IStructuredSelection selection =  
(IStructuredSelection) e.getSelection();  
// selektiertes Element holen  
Object selected = selection  
.getFirstElement();  
// und als aktuelles Element im Modell  
// setzen.  
model.setCurrent(selected);  
}  
});  
// aktuelle Playlist ermitteln  
String playlistFile = model.getPlaylistName();  
// und als Inputdaten setzen  
viewer.setInput(playlistFile);  
return composite;  
}  
/*  
* Fenster öffnen und beim Modell als Listener eintragen  
*/  
public int open() {  
// Selektion beim Viewer initialisieren  
viewer.setSelection(model.getSelection());  
// als Listener beim Modell eintragen  
model.addSelectionChangedListener(this);
```

```
—— // Fenster öffnen  
—— return super.open();  
—— }  
—— /*  
—— * Fenster schließen und beim Modell als Listener entfernen  
—— */  
—— public boolean close() {  
—— // als Listener beim Modell entfernen  
—— model.removeSelectionChangedListener(this);  
—— // Fenster schließen  
—— return super.close();  
—— }  
—— /*  
—— * Modell hat sich geändert, Viewer muss angepasst werden.  
—— */  
—— public void selectionChanged(SelectionChangedEvent event) {  
—— // Aktualisierung der Tabelle erzwingen  
—— viewer.refresh();  
—— // Selektion neu setzen  
—— viewer.setSelection(model.getSelection());  
—— }  
+  
/**  
 * Diese Methode wird aus Superklasse Window heraus  
 * aufgerufen. Wir bauen die Fensterinhalte auf  
 */  
protected Control createContents(Composite parent) {  
    parent.setLayout(new FillLayout());  
    Composite composite = new Composite(parent, SWT.NONE);  
    composite.setLayout(new FillLayout());  
    // Browser-Widget erzeugen  
    browser = new Browser(composite, SWT.NONE);  
    GridData data = new GridData(GridData.FILL_BOTH);
```

```
        browser.setLayoutData(data);
        return composite;
    }

    /**
     * Inhalt des Textbereiches setzen
     */
    public void update() {
        String description = model.getFeature(Player.DESCRPTION);
        if (description == null)
            browser.setText("");
        else {
            // Alle Schlüsselwörter farbig hervorheben
            StringBuffer html = new StringBuffer(
                "<html><small><font color='green'>");
            StringTokenizer tokenizer = new StringTokenizer(
                description, "<> \\n\\t", true);
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                if (token.length() == 1) {
                    html.append(token);
                } else if (token.startsWith("$")) {
                    html.append("<font color='red'>");
                    html.append(token.substring(1));
                    html.append("</font>");
                } else
                    html.append(token);
            }
            html.append("</font></small></html>");
            // Ausgabe im Browser
            browser.setText(html.toString());
        }
    }
}
```

```
/**
 * open()-Methode aus Window wurde überschrieben, um beim
 * Öffnen den angezeigten Text zu aktualisieren.
 */
public int open() {
    update();
    return super.open();
}
}
```

Seite 335:

Insbesondere erfahren wir mittels der Methode `getLocation()` den Ort des ~~Workspace-Wurzelverzeichnisses~~ **Eclipse-Arbeitsverzeichnisses**.

Seite 336:

Mit der Methode `find()` erhält man die URL einer ~~Workspace~~ **Plugin**-Ressource, und mit der Methode `openStream()` kann man einen Eingabestrom zu einer **solchen** Ressource ~~im Eclipse-Workspace~~ öffnen.

Seite 349:

So können mit den Methoden ~~setPluginPreferences()~~ **setPluginPreferences()** und ~~getPluginPreferences()~~ **getPluginPreferences()** die aktuellen Einstellungen des Plugin gespeichert bzw. abgefragt werden.

Seite 357:

Dieser Erweiterungspunkt erlaubt die Definition von **AktivitätenCapabilities** wie z.B. »Java-Development« oder »Plug-in-Development Team«. Für jede **AktivitätCapability** können der Workbench verschiedene Features aus Plugins zugeordnet werden. Aktiviert der Endbenutzer eine solche **AktivitätCapability**, werden die zugeordneten Features in der Workbench sichtbar. Dabei können **AktivitätenCapabilities** voneinander abhängig sein. Wird eine **AktivitätCapability** aktiviert, werden auch alle **AktivitätenCapabilities** aktiviert, von denen die aktuelle **AktivitätCapability** abhängig ist.

Seite 379:

Wie kann sich nun eine Komponente einem Selektionsdienst bekannt machen, um ihm Selektionsereignisse mitzuteilen? Zu diesem Zweck ~~implementiert~~ **registriert** die Komponente ~~lediglich das Interface~~ **eine Instanz des Typs** `ISelectionProvider` ~~und dessen~~ **mit den** Methoden `addSelectionListener()`, `removeSelectionListener()`, `getSelection()` **und** `setSelection()` **mit der ihr zugehörigen** `IWorkbenchSite`. Immer dann, wenn eine Komponente aktiviert wird, prüft die Workbench, ob ~~die Komponente dieses Interface implementiert~~ **ein** `ISelectionProvider` **registriert ist**.

Seite 383:

Alle diese Methoden sind von `EditorPart`-Erweiterungen geeignet zu überschreiben. Im Eclipse-SDK gibt es bereits ~~dreier~~ **abstrakte** Erweiterungen von `EditorPart`: `AbstractTextEditor`, `AbstractDecoratedTextEditor`, `MultiEditor` und `MultiPageEditorPart` (siehe Abb. 11–17).

Seite 411:

```
href="../../org.eclipse.jdt.doc.user_3.0.0/tips/jdt-tips.html"
```

Seite 413:

```
<script language="JavaScript"  
  src="../../org.eclipse.help/livehelp.js"/>
```

Seite 414:

Ein Beispiel für eine solche Klasse und ihren Aufruf finden Sie in Abschnitt 13.10**1.3**.

Seite 444:

Zunächst stellen wir sicher, dass die Funktionen für die Plugin-Entwicklung in der Eclipse-Plattform aktiviert sind. ~~Rufen Sie die Funktion~~ **Rufen Sie die Funktion** `Window>Configure Activities ...` auf und markieren Sie die Gruppe ~~Plugin Development~~ **Plugin Development**. ~~Rufen Sie die Einstellung~~ **Rufen Sie die Einstellung**

Window>Preferences>Workbench>Capabilities auf und markieren Sie die Gruppe *Development*.

Seite 453:

```
<extension point="org.eclipse.ui.preferencePages">
  <page id="com.bdaum.SpellChecker.preferences.defaultPreferences"
        name="%Spelling"
        class="com.bdaum.SpellChecker.preferences.
DefaultSpellCheckerPreferencePage>
        class=
"com.bdaum.SpellChecker.preferences.DefaultSpellCheckerPreferencePage">
  </page>
</extension>
```

Seite 455:

```
<attribute name="preferences" type="string">
  <annotation>
    <appInfo>
      <meta.attribute kind="java" basedOn=
"com.bdaum.SpellChecker.preferences.SpellCheckerPreferences"/>
    </appInfo>
  </annotation>
</attribute>
```

Seite 464:

```
    } catch (Exception e1) {
    }
  }
}
}
```

```

    return instance;
}

```

Seite 490:

Dies ist erforderlich, da einige dieser Änderungen aus einem anderen Thread, dem Thread für die Rechtschreibprüfung, kommen (siehe Abschnitt 8.5.32).

Seite 507:

Alle diese Operationen werden in einen `syncExec()`-Aufruf gekapselt, um Thread-Fehler zu vermeiden (siehe Abschnitt 8.5.32).

Seite 507:

<code>postRestore</code>	Wird aufgerufen, nachdem Fenster wiederhergestellt wurden, deren Zustand dauerhaft gespeichert wurde.
--------------------------	---

Seite 567:

Nun fehlen nur noch die Einträge für das Hilfe-Inhaltsverzeichnis und für die Kontext-Assoziationen. Das verschieben wir jedoch am besten, da für diesen Schritt das Hilfe-Inhaltsverzeichnis `toc.xml` und die Kontextdatei `contexts.xml` benötigt werden. Wie das funktioniert, haben wir bereits im vorigen Beispiel in Kapitel 13.10 gesehen. Wir verzichten deshalb für dieses Beispiel auf eine Dokumentation dieses Schritts.

Allerdings ist in der Rich Client Plattform eine explizite Spezifikation des Hilfesystems notwendig. Dazu fügen wir dem Abschnitt *Dependencies* die folgenden fünf Hilfemodule zu:

```
org.eclipse.help, org.eclipse.help.base, org.eclipse.help.ui,
org.eclipse.help.webapp, org.eclipse.tomcat
```

und spezifizieren die Erweiterungspunkte `org.eclipse.help.base.webapp` und `org.eclipse.ui.helpSupport`. Letzterem fügen wir als *config*-Klasse die Klasse `org.eclipse.help.ui.internal.DefaultHelpUI` zu, die standardmäßig in der Eclipse-Workbench ihren Dienst als Hilfe-Benutzeroberfläche versieht. Alternativ könnte man freilich auf die `DefaultHelpUI` und `Tomcat` verzichten und eine eigene Hilfeoberfläche auf Basis der Klasse `AbstractHelpUI` implementieren.

```

<requires>
<import plugin="org.eclipse.core.runtime"/>
<import plugin="org.eclipse.help"/>
<import plugin="org.eclipse.help.base"/>

```

```
<import plugin="org.eclipse.help.ui"/>
<import plugin="org.eclipse.help.webapp"/>
<import plugin="org.eclipse.tomcat"/>
<import plugin="org.eclipse.ui"/>
<import plugin="org.eclipse.ui.forms"/>
</requires>
```

Seite 568:

```
<extension point="org.eclipse.help.base.webapp"/>
<extension point="org.eclipse.ui.helpSupport">
    <config class="org.eclipse.help.ui.internal.DefaultHelpUI"/>
</extension>
<extension id="product"
    point="org.eclipse.core.runtime.products">
    <product name="Hex Game Machine"
        application="com.bdaum.Hex.RcpApplication"
        description="The game of Hex">
        <property name="appName" value="Hex"/>
        <property name="windowImage" value="hexWindow.gif"/>
</product/>
</product>
</extension>
```

Seite 590:

Was vielleicht etwas störend wirkt ist der Reiter am Hex-View. Den kann man allerdings dadurch loswerden, dass man in der Klasse `RcpPerspective` anstelle von `addView()` die Methode `addStandaloneView()` verwendet.

Seite 612:

[Steams2000] Beth Steams: Java Beans 101; Sun Microsystems, <http://java.sun.com/developer/onlineTraining/Beans/bean01/index.html>; 2000.