

HANSER

Robert Hanson, Adam Tacy

# GWT im Einsatz

AJAX-Anwendungen entwickeln mit dem Google Web Toolkit

ISBN-10: 3-446-41241-7

ISBN-13: 978-3-446-41241-5

Leseprobe

Weitere Informationen oder Bestellungen unter  
<http://www.hanser.de/978-3-446-41241-5>  
sowie im Buchhandel

## 4 Mit Widgets arbeiten

### Dieses Kapitel

- erläutert die Funktionsweise von Widgets,
- zeigt, wie die GWT-Widgets verwendet werden,
- beschreibt die Interaktion mit Widgets,
- erklärt, wie Widgets erstellt werden.

Nachdem wir als Aufwärmübung die erste Version der Anwendung *Dashboard* erstellt haben, wollen wir auf dem Weg zur Vollversion des Dashboards nun einige Schritte weitergehen. Diese Vollversion verwendet viele verschiedene Arten von Widgets und Panels und muss auch mit unterschiedlichen Arten von Ereignissen umgehen können. Um dorthin zu kommen, müssen wir Ihnen aber erst einmal erklären, was das überhaupt für Komponenten sind!

In diesem Kapitel geht es speziell um *Widgets*. Widgets sind sichtbare Komponenten einer GWT-Anwendung, die der Benutzer auf der Browserseite sieht, also etwa Schaltflächen, Beschriftungen, Bilder und das Menüsystem. Stellen Sie sich vor, Sie kaufen einen neuen Plasmabildschirm und stellen dann fest, dass der Hersteller weder Bedienelemente vorgesehen noch eine Fernbedienung beigelegt hat. Dasselbe wäre Ihre Anwendung ohne Widgets: vollkommen nutzlos.

### Definition

Widgets sind die sichtbaren Komponenten einer GWT-Anwendung, die der Benutzer auf der Browserseite sehen kann.

In diesem und den nächsten fünf Kapiteln werden wir einige Grundlagen zu GWT behandeln, um Sie in die Lage zu versetzen, den Aufbau des Dashboards vollständig zu verstehen. Auf diesem Wege werden Sie einige der für das Dashboard erforderlichen Komponenten erstellen. Die Kapitel behandeln die Konzepte, die in Tabelle 4.1 aufgeführt sind.

**Tabelle 4.1** Die fünf Buchkapitel, die die GWT-Grundlagen zu Widgets, Panels, Ereignissen und zusammengesetzten Widgets sowie die Verwendung von JSNI behandeln

Kapitel	Gegenstand	Details
4	Widgets	Widgets sind die sichtbaren Komponenten einer GWT-Anwendung, die der Benutzer auf dem Bildschirm sieht: Schaltflächen, Beschriftungen, Menüleisten usw. Sie werden für das Dashboard zwei Widgets erstellen: <code>PNGImage</code> und <code>ToggleMenuItem</code> .
5	Panels	Panels unterstützen Sie bei der Strukturierung der Ansicht auf dem Bildschirm. Mit ihnen können Sie Widgets positionieren (z. B. mit dem Panel <code>VerticalPanel</code> ) und ihre Sichtbarkeit verwalten (z. B. mit <code>DeckPanel</code> ). Sie werden das Panel <code>DashboardContainer</code> erstellen, das alle Dashboard-Komponentenanwendungen aufnimmt, die Sie im Verlauf dieses Buchs erstellen.
6	Ereignisse	Die Funktionalität in GWT basiert auf Ereignissen. Ein Ereignis liegt etwa vor, wenn ein Benutzer eine Schaltfläche anklickt, ein Formularversand durchgeführt wird oder der Benutzer eine Komponente via Drag & Drop verschiebt. Sie werden das Panel <code>DashboardContainer</code> aus Kapitel 5 so erweitern, dass es Doppelklick- und Fokusereignisse verarbeiten kann.
7	Zusammengesetzte Widgets	Schließlich kommen wir zu den zusammengesetzten Widgets, die die Leistung der vorangegangenen drei Kapitel unter ihrer Haube vereinigen. Dies sind Widgets, die aus anderen Widgets bestehen und in einem oder mehreren unterschiedlichen Panels abgelegt werden. Sie stellen die leistungsfähigste Form einer erstellbaren Komponente dar und werden das zusammengesetzte Widget <code>EditableLabel</code> sowie das Objekt <code>DashboardComposite</code> erstellen, die beim Dashboard zum Einsatz kommen.
8	JSNI (JavaScript Native Interface)	JSNI ermöglicht Ihnen den Zugriff auf natives JavaScript. Sie können sich dies ähnlich vorstellen wie die Verwendung von Assemblycode in einem C-Programm. Kapitel 8 behandelt die für die Verwendung von JSNI geeigneten Klassen und die Frage, wie man vorhandene JavaScript-Bibliotheken von Drittanbietern für die Verwendung in den GWT-Programmen kapselt.

Da dieses Kapitel die Widgets behandelt, klären wir zunächst, was Widgets eigentlich sind. Danach werfen wir einen kurzen Blick auf die standardmäßig mit GWT ausgelieferten Widgets und beschäftigen uns mit ihrer Verwendung in den Komponenten der Anwendung *Dashboard*.

Im zweiten Teil dieses Kapitels erfahren Sie, wie man eigene Widgets erstellt – nur für den Fall, dass die standardmäßig vorhandenen Widgets nicht ausreichen oder Ihren Anforderungen nicht entsprechen. Im Zuge dieser Beschreibung werden Sie ein Widget `PNGImage` erstellen, mit dem Sie PNG-Bilder im Dashboard verwenden können. (Bei GWT 1.3 unterstützt das Widget `Image` die Transparenz von PNG-Grafiken nicht bei allen Browsern korrekt, weswegen man zu diesem Zweck ein eigenes Widget erstellen muss.) Wir werden zudem das Widget `MenuItem` so erweitern, dass es die Bedürfnisse der Anwendung *Dashboard* erfüllt. Wenn Sie nun bereit sind, kann es losgehen. Definieren wir „Widget“ also erst einmal.

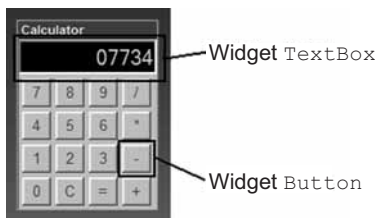
## 4.1 Was ist ein Widget?

Widgets stellen einen der vier wesentlichen Bausteine von GWT-Anwendungen dar (die anderen sind Panels, Ereignisse und die Serverkommunikation einschließlich RPC, Formularversand, JSON- und XML-Behandlung sowie die traditionelle `Ajax-XMLHttpRequest`). Wenn ein Benutzer Ihre GWT-Anwendung startet, betrachtet er einen Satz Widgets, die mithilfe von Panels positioniert wurden und auf Ereignisse reagieren. Widgets sind – ebenso wie die Bedienelemente des oben erwähnten Plasmabildschirms – Komponenten, mit denen der Benutzer interagiert. Glücklicherweise bietet GWT bereits zahlreiche Widgets kostenlos, darunter auch „die üblichen Verdächtigen“: Schaltflächen (wie in Abbildung 4.1), Textfelder und Menüs.

Click me

**Abbildung 4.1** Schaltflächen-Widget als gerendertes HTML in Firefox

Die meisten Anwendungen werden unter Verwendung mehrerer Widgets erstellt, die Sie in Panels ablegen, um eine gewisse Struktur zu erstellen. Dies wird offensichtlich, wenn Sie sich etwa die Komponentenanwendung Calculator des Dashboards ansehen (Abbildung 4.2).



**Abbildung 4.2**

Dashboard-Anwendung Calculator. Sie zeigt, wie sich aus einer Anzahl von Widgets eine vollständige Anwendung zusammensetzen lässt.

Widgets wie auch die im nächsten Kapitel behandelten Panels existieren in GWT quasi in doppelter Ausführung: Sie können sie als Java-Objekte wie auch als DOM-Elemente begreifen. Die Sichtweise „Java-Objekt“ verwenden Sie in der täglichen Programmierung, um Anwendungen zu erstellen. Die DOM-Ansicht haben die Java-Objekte in Ihrem Programm, wenn Sie sie im Kontext dessen betrachten, was der Webbrowser anzeigt. Wir werden uns diese beiden Ansichten in den nächsten Abschnitten ansehen und dabei mit der Java-Objektansicht von Widgets beginnen.

### 4.1.1 Widgets als Java-Objekte verwenden

Der Zweck von GWT besteht darin, Rich Internet Applications in Java zu entwickeln und den GWT-Compiler dann den HTML- und JavaScript-Code generieren zu lassen, den man benötigt, um die Anwendung in einer Vielzahl von Browsern ausführen zu können. Dafür müssen Sie verschiedene Browserobjekte – bei GWT *Widgets* genannt – in Java-Programmen darstellen können.

Dieser Ansatz profitiert von der in der Welt der objektorientierten Programmierung vorhandenen Fähigkeit, Objekte und Konzepte als Programmierobjekte zu modellieren. In ei-

dem GWT-Programm können Sie beispielsweise problemlos ein Java-Objekt verwenden, das `Button` heißt. Dieses `Button`-Objekt bildet verschiedene Eigenschaften nach, die Sie bei einer Schaltfläche erwarten, z. B. die Möglichkeit, den angezeigten Text festzulegen und anzuklicken. Sie können alle Komponenten, die Sie in einem Browser sehen wollen (also die Widgets), als Java-Objekte mit Methoden und Eigenschaften modellieren.

Beim täglichen Programmierereinsatz von GWT werden Sie alle Widgets in ihrer natürlichen Java-Objektform zu sehen bekommen. Die Schaltfläche, die wir in der Einleitung dieses Abschnitts erwähnten, wird wie folgt durch Aufruf des Konstruktors der GWT-Java-Klasse `Button` erstellt:

```
Button theButton = new Button("Click Me");
```

Dieser Code erstellt ein neues GWT-spezifisches Java-Schaltflächenobjekt, für das Sie dann verschiedene Klassenmethoden ausführen können. Die in Tabelle 4.2 gezeigten Aufgaben sind typische Operationen, die Sie bei einem GWT-Widget `Button` durchführen können.

**Tabelle 4.2** Aus der Anwendung einiger Methoden der Java-Klasse `Button` auf das Java-Objekt `Button` resultierende Funktionalitäten

Code	Beschreibung
<code>theButton.setStyleName("buttonStyle");</code>	Legt den CSS-Klassennamen (Cascading Style Sheet) für die Schaltfläche fest. Ein entsprechender Eintrag sollte sich in dem CSS-Stylesheet befinden, welches an das Webdokument angehängt ist. Beachten Sie aber, dass dessen Name ein Punkt vorangestellt ist (z. B. <code>.buttonStyle{...}</code> ).
<code>theButton.addClickListener( new ClickListener(){     public void onClick(Widget sender){     } });</code>	Fügt ein <code>ClickListener</code> -Objekt zur Schaltfläche hinzu (dies ist ein Ereignis-Listener, der gezielt auf Mausclickereignisse achtet). Wenn die Schaltfläche angeklickt wird, wird der Code in der Methode <code>onClick()</code> , die im <code>ClickListener</code> definiert ist, ausgeführt.
<code>theButton.setText("Go on, click me");</code>	Ändert den auf der Schaltfläche angezeigten Text vom ursprünglichen „Click Me“ zu „Go on, click me“.
<code>theButton.setStyleName("buttonStyle");</code>	Blendet die Schaltfläche im Webbrowser aus, d. h. sie ist nicht mehr sichtbar.

Die Java-Ansicht der Widgets ist für jemanden, der mit dem Schreiben von Java-Programmen oder der Verwendung ähnlicher objektbasierter Hochsprachen vertraut ist, völlig unkompliziert: Sie erstellen Objekte aus Klassen und rufen die Methoden auf, die diese Klassen aufweisen. Was Ihnen die Java-Ansicht jedoch nicht bietet, ist eine Antwort auf die Frage, wie diese Widgets auf einer Webseite dargestellt werden. Diese gewährt Ihnen die alternative DOM-Darstellung von Widgets.

### 4.1.2 Widgets als DOM-Elemente betrachten

Die Java-Darstellung der Widgets, die Sie soeben kennengelernt haben, funktioniert im Java-Code gut und gestattet Ihnen die Erstellung von GWT-Anwendungen nach Belieben und unter Verwendung einer beliebigen Anzahl von Widgets und der zugehörigen Methoden zur Implementierung der jeweiligen Funktionalität. Allerdings können Sie diese Java-Objekte in keinem Webbrowser darstellen, d. h. Sie haben noch keine Ajax-Anwendung. Und hier kommt die alternative DOM-Darstellung der Widgets ins Spiel.

DOM (Document Object Model) ist die Ansicht des Browsers von der angezeigten Webseite. Sie können das DOM mithilfe einer Vielzahl von Sprachen manipulieren, und die Auswirkungen der meisten Manipulationen sind auf der aktuellen Browserseite sofort sichtbar. Zu diesen Manipulationen kann etwa das Hinzufügen oder Entfernen von Elementen, das Ein- oder Ausblenden von Elementen oder das Ändern der Elementposition gehören. Im Fall von GWT wird diese Manipulation letztendlich über JavaScript im kompilierten Code durchgeführt; in Ihrem Programm jedoch verwenden Sie Java. Versuchen wir nun, diese Klippe zu umschiffen.

Alle GWT-Widgets verfügen über eine alternative DOM-Darstellung, die parallel zum Java-Objekt erstellt wird. Der Java-Konstruktor ist im Allgemeinen für die Erstellung dieser DOM-Darstellung zuständig. Wenn Sie sich den Konstruktor für die Klasse `Button` genau betrachten, dann sehen Sie folgende Definition:

```
public Button() {
    super(DOM.createButton());
    adjustType(getElement());
    setStyleName("gwt-Button");
}
```

Der Aufruf von `DOM.createButton()` erstellt das DOM-Element `<button>` über die GWT-Klasse `DOM`. Ferner wird im Konstruktor des übergeordneten Elements die Methode `setElement()` aufgerufen, um die DOM-Darstellung der Java-Klasse des Widgets auf dieses neue `<button>`-Element festzulegen. Mithilfe dieses von der Methode `setElement()` festgelegten Wertes erhalten Sie Zugriff auf die DOM-Darstellung der Java-Ansicht (dies können Sie dann mit der Methode `getElement()` tun). Würden Sie sich diese DOM-Darstellung einer GWT-Java-Schaltfläche ansehen, dann sähe Letztere in etwa folgendermaßen aus:

```
<button class="gwt-Button"
        eventbits="7041"
        onchange="null"
        onload="null"
        onerror="null">
    Click me
</button>
```

Dies ist die standardmäßige DOM-Darstellung eines `<button>`-Objekts mit einigen zusätzlichen GWT-spezifischen Attributen. Diese zusätzlichen Attribute (`eventbits`, `onload` usw.) werden vom Java-Konstruktor des Objekts festgelegt, wenn diese DOM-Darstellung erstellt wird. Lassen Sie sich vom Namen des ersten Attributs `class` nicht irritieren: Dieser verweist auf die CSS-Formatierung, die auf ein Widget angewendet werden kann, und nicht auf den Java-Klassennamen des Widgets.

Das nächste Attribut `eventBits` tritt bei vielen Widgets auf. Es gibt gegenüber GWT an, auf welche Ereignisse das betreffende Widget horcht (man spricht auch davon, welche Ereignisse es *versenkt*). Wir behandeln diesen Aspekt in Kapitel 6 eingehender, bis dahin wird er aber in den Beispielen immer wieder auftauchen. Ein Widget versenkt Browserereignisse, für deren Behandlung es sich interessiert.

Eine goldene Regel, die für diese DOM-Darstellung gilt, besagt, dass Sie sich nicht darauf verlassen können, dass ein bestimmtes Widget als ein bestimmtes DOM-Element implementiert ist, weil nicht zu verhindern ist, dass künftige Implementierungen von Widgets mit anderen DOM-Elementen dargestellt werden als bei vorherigen Versionen. Indem Sie Ihre Programmierbemühungen auf die Java-Codeansicht konzentrieren und hierzu die von den einzelnen GWT-Widget-Klassen zur Umsetzung von Funktionalitäten bereitgestellten Methoden verwenden, schützen sie sich gegen mögliche zukünftige Änderungen auf der DOM-Ebene. Als Faustregel gilt: Sie sollten – wenn überhaupt – in Ihren Anwendungen nur äußerst selten auf das DOM zugreifen müssen.

Nun wissen Sie, was ein Widget ist und wie es in Java wie auch im DOM dargestellt wird. Als Nächstes folgt eine kurze Besichtigung der Widgets, die mit der GWT-Distribution ausgeliefert werden.

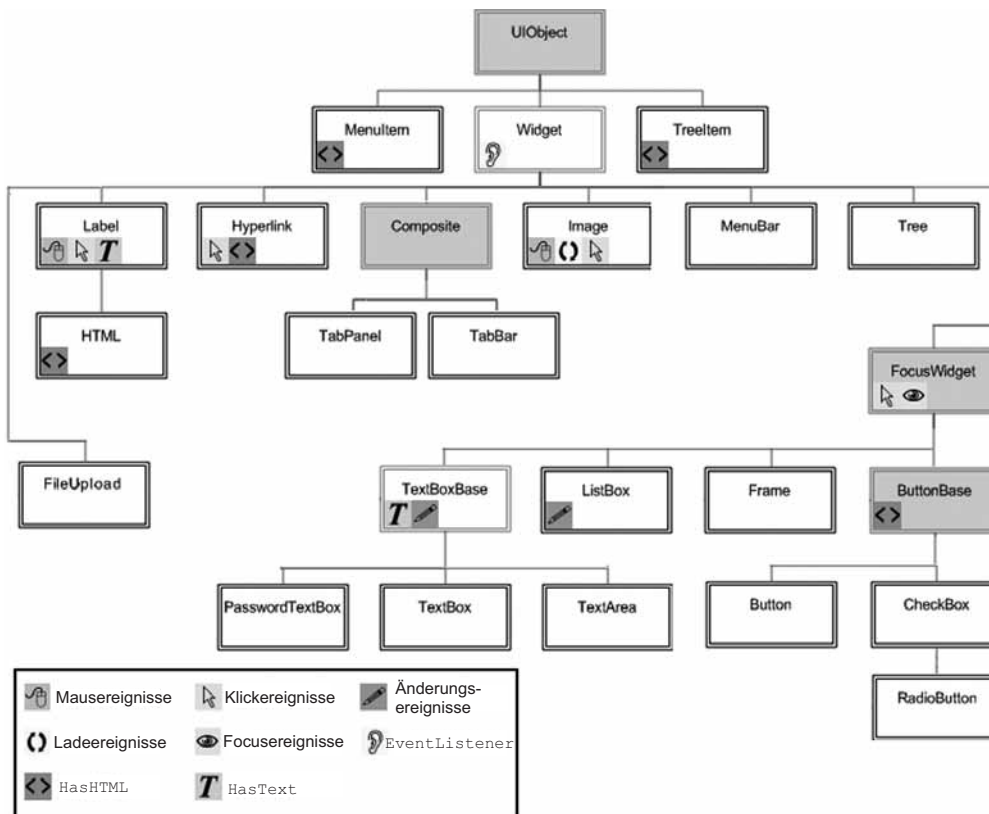
## 4.2 Die Standard-Widgets in GWT

---

Die GWT-Standarddistribution wird mit einer Vielzahl von Widgets ausgeliefert, die Sie in Ihren Anwendungen einsetzen können. Diese Widgets decken alle erwarteten Bereiche ab: Schaltflächen, Textfelder usw. Allerdings fehlen einige Widgets, mit denen man eigentlich hätte rechnen können. Hierzu gehören Fortschrittsbalken und Schieberegler (einen solchen basteln wir uns allerdings in Kapitel 7).

Bei den Widgets haben die GWT-Entwickler eine strikte Hierarchie der Java-Klassen implementiert, um dort, wo eine natürliche Konsistenz vorhanden ist, ein solches Konsistenzelement auch Widget-übergreifend bereitzustellen. Nehmen wir etwa die Widgets `TextBox`, `TextArea` und `PasswordTextBox`: Es wäre wohl kaum abwegig, davon auszugehen, dass diese bestimmte Eigenschaften miteinander teilen. GWT erkennt diese Tatsache an und fasst die gemeinsamen Eigenschaften in einer Klasse `TextBoxBase` zusammen, die diese drei Widgets erben. Um einen Eindruck von dieser Hierarchie zu erhalten, werfen Sie einen Blick auf Abbildung 4.3.

Sie können dieser Hierarchie entnehmen, dass alle Widgets in letzter Konsequenz von der Klasse `UIObject` erben. Diese Klasse enthält eine Anzahl wesentlicher Organisations- und Eigenschaftsaspekte. In der Klasse `UIObject` finden Sie die Methode `setElement()` vor, die wir weiter oben bereits erwähnten, weil mit ihr die physische Verbindung zwischen dem Java-Objekt und den DOM-Darstellungen eines Widgets festgelegt werden kann. Von `UIObject` abgeleitete Klassen müssen *zallererst* diese Methode aufrufen, bevor andere Methoden aufgerufen werden, da nur so sichergestellt ist, dass die Verknüpfung mit einem Browser-Element hergestellt wird.



**Abbildung 4.3** GWT-Widget-Klassenhierarchie, die die Widgets von GWT 1.3 zeigt. (Weitere Widgets werden in den nachfolgenden Releases fortlaufend hinzugefügt, aber bei GWT 1.3 zeigt sich die Hierarchie auf ebenso ansprechende wie stringente Weise.) Ebenfalls angegeben sind die Typen der Ereignis-Listener, die für die einzelnen Widgets registriert werden können, und die Verknüpfbarkeit mit Text und/oder HTML.

#### Hinweis

Alle GWT-Widgets erben von der Klasse `UIObject`. Diese Klasse stellt einen allgemeinen Satz von Methoden und Attributen für alle Widgets bereit, etwa zur Einstellung von Größe, Sichtbarkeit und Formatnamen. Gleichzeitig ermöglicht sie die Verknüpfung zwischen den Java- und DOM-Darstellungen.

Wir werden nicht alle Methoden der Klasse `UIObject` einzeln behandeln, sondern die typische Funktionalität beleuchten, die Sie bei allen Widgets erwarten können. Die Klasse `UIObject` gewährt Zugriff auf umfangreiche DOM-Funktionalitäten, ohne dass Sie direkt auf das DOM zugreifen müssten. So können Sie beispielsweise die Höhe eines GWT-`UIObject` mit der Methode `setHeight()` festlegen, die ihrerseits die Methode `setAttribute()` der Klasse `DOM` verwendet:

```

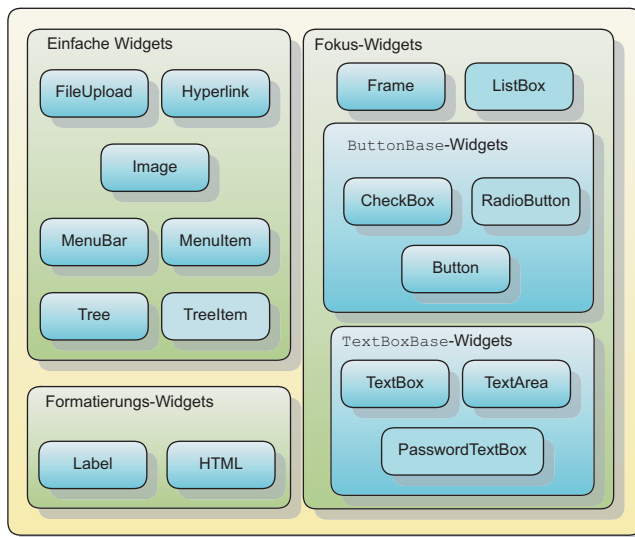
public void setHeight(String height) {
    DOM.setAttribute(element, "height", height);
}

```

Die übrigen in diesem Format vorhandenen Methoden ermöglichen das Festlegen von Breite, Titel (der angezeigt ist, wenn der Mauszeiger ein Element überfährt) sowie Höhe und Breite gleichzeitig über die Methode `setSize()`. Alle diese Methoden nehmen Strings als Parameter entgegen (z. B. `setSize("100px", "200px")`). Im Gegensatz dazu gestattet die Methode `setPixelSize()` auch Integer-Werte, z. B. `setPixelSize(100, 200)`. Obwohl diese Methoden zur Festlegung von Formatattributen vorhanden sind, empfehlen wir, die Formatierung grundsätzlich mit CSS durchzuführen. Grund hierfür ist die praktische Trennung zwischen funktionsspezifischem Code und dem Erscheinungsbild der Anwendung. Außerdem können Sie der Grafikaufteilung auf diese Weise die Formatierung der Anwendung auf gewünschte Weise überlassen.

Neben `UIObject` müssen alle Widgets (mit Ausnahme von `TreeItem` und `MenuItem`) von der Klasse `Widget` erben. Diese Klasse erst macht Widgets zu dem, was sie sind, und vererbt auch die Methoden, die aufgerufen werden, wenn ein Widget an ein Panel angehängt oder von diesem getrennt wird. Die Klasse enthält zudem die Standardimplementierung der Methode `onBrowserEvent()`, die einem Widget die Verwaltung aller Ereignisse ermöglicht, die es versenkt hat (wie dies genau funktioniert, sehen Sie in Kapitel 6).

Im nächsten Abschnitt werfen wir einen kurzen Blick auf einige Widgets, die Sie kostenlos mit der GWT-Distribution erhalten, und erfahren, wie Sie sie in der Anwendung *Dashboard* einsetzen. Unsere Absicht besteht nicht darin, alle Widgets zu beschreiben, zumal ihre Anzahl mit jedem neuen GWT-Release ansteigt. Abbildung 4.4 zeigt, welche Widgets wir im Folgenden behandeln wollen.



**Abbildung 4.4**  
Übersicht der in GWT enthaltenen Widgets, die auch zeigt, wie diese Widgets in diesem Kapitel zu Gruppen zusammengefasst und erläutert werden

Bei der Beschreibung der Widgets in diesem Kapitel werden wir die jeweiligen Einsatzmöglichkeiten anhand einfachen Codes erläutern und angeben, wo im Dashboard sie zum Einsatz kommen. Der Code mag hier und dort etwas fremdartig wirken, aber keine Sorge: Wir haben die meisten im Code verwendeten Konzepte schlichtweg noch nicht vorgestellt,

weil sie erst in den nächsten Kapiteln an der Reihe sind. Trotzdem sollten Sie in aller Regel erkennen können, was dort passiert.

Es gibt einige Grundsätze, die Sie beim Lesen der folgenden Abschnitte beachten sollten. Zunächst einmal ist das Folgende keineswegs als vollständige Beschreibung des GWT-API für Widgets gedacht, da dies ausgesprochen langweilig wäre. Stattdessen werden wir versuchen, einige der wichtigsten Methoden und Fallstricke zu veranschaulichen und darzustellen, wo Sie diese Methoden in der Anwendung *Dashboard* einsetzen, damit der Code auch Sinn ergibt. Wann immer eine `getter`-Methode – z. B. `getText()` – verwendet wird, bietet das GWT-API auch die zugehörige `setter`-Methode (`setText()`). Schließlich werden wir nicht allzu häufig Namen oder die Anzahl der Parameter für eine Methode angeben, sofern dies nicht unbedingt notwendig ist. In unserem Buch wollen wir uns auf das Wesentliche konzentrieren, ohne allzu sehr in Details abzuleiten. Die Onlinereferenz für das GWT-API (<http://code.google.com/webtoolkit/documentation/gwt.html>) ist ebenso eine Unterstützung von unschätzbarem Wert wie eine gute IDE.

Wir werden die Behandlung der Widgets in fünf Hauptkategorien aufschlüsseln, die auch in Abbildung 4.4 gezeigt sind: Basis-, Beschriftungs-, Fokus-, `ButtonBase`- und `TextBoxBase`-Widgets. Beginnen wir mit den Basis-Widgets.

### 4.2.1 Mit Basis-Widgets interagieren

Basis-Widgets definieren wir als solche, die direkt von der Klasse `Widget` erben. Es gibt insgesamt fünf dieser Widgets, die wir kurz im Kontext der Anwendung *Dashboard* betrachten wollen, die Sie gerade erstellen. Hier und dort werden wir auch ein – isoliertes – Codebeispiel anführen, um einen bestimmten Aspekt oder eine spezielle Eigenschaft zu betonen.

#### Dateien hochladen mit `FileUpload`

Das Widget `FileUpload` agiert als GWT-Wrapper für das Standardtextfeld des Browsers, mit dem Sie dem Benutzer das Hochladen einer Datei vom lokalen Computer auf Ihren Server ermöglichen (siehe Abbildung 4.5).



Abbildung 4.5 Das Widget `FileUpload`

Bedenken Sie, dass dies nur die clientseitige Komponente ist: Das Anklicken der Schaltfläche `DURCHSUCHEN` gestattet dem Benutzer die Auswahl einer Datei auf seinem Computer über das Standarddialogfeld `DATEI SUCHE`, ermöglicht ihm aber noch nicht das Speichern der Datei auf Ihrem Server. Hierzu ist ein bisschen mehr Arbeit erforderlich. Dieses Widget sollte in ein Formular eingebettet werden, dessen Kodierung auf `multipart` festgelegt ist. Sobald Sie bereit sind, senden Sie das Formular an Ihren Server, um den Upload der gewählten Datei zu verarbeiten (ein Beispiel für diese Funktionalität stellt die Klasse `FileUploadServlet` im Server-Package von *Dashboard* dar). Dieses Widget bietet keinen serverseitigen Code zur Behandlung des Datei-Uploads: Sie müssen diesen selbst

bereitstellen, können dies aber in der von Ihnen bevorzugten serverseitigen Sprache tun. Wir werden uns ein `FileUpload`-Widget bei der Beschreibung der serverseitigen Komponenten in Kapitel 12 genauer ansehen. Aber auch weiter unten in diesem Kapitel wird es eine Rolle spielen, wenn wir beschreiben, wie man Widgets erstellt.

### Hinweis

Da aus Ihrer GWT-Anwendung JavaScript-Code wird, unterliegt der Zugriff auf Dateien den gleichen Beschränkungen wie bei JavaScript. Dies bedeutet, dass Dateien nicht direkt auf dem Computer eines Benutzers gespeichert werden können. Zudem besteht die einzige Möglichkeit, auf Dateien auf dem Computer eines Benutzers zuzugreifen, in der Verwendung des Widgets `FileUpload`.

`FileUpload` ist eines der am inkonsistentesten implementierten Widgets: Die verschiedenen Browser gewähren für seine Formatierung unterschiedliche Sicherheitsbeschränkungen und Fähigkeiten. So gestatten die meisten Browser beispielsweise keine Festlegung eines Standardwertes für das Textfeld, weil andernfalls Webanwendungen Dateien „angeln“ könnten. Wie bei allen Widgets müssen Sie auch hier daran denken, dass, wenn Sie in HTML und JavaScript etwas nicht mit dem Widget tun können, dies auch in GWT nicht möglich ist. GWT bietet lediglich eine Methode `getFilename()`, die den vom Benutzer gewählten Dateinamen abrufen. Verwechseln Sie diese Methode nicht mit den Methoden `getName()` und `setName()`, die der Festlegung des DOM-Namens für das Widget `FileUpload` dienen.

Doch genug zum Datei-Upload – zumindest bis Kapitel 12. Wir wollen uns ja noch ein paar andere von GWT gebotene Basis-Widgets ansehen.

### Durch Ihre Anwendung mit Hyperlinks navigieren

Das Widget `Hyperlink` agiert als interner Hyperlink in Ihrer GWT-Anwendung. Für den Benutzer sieht es genauso aus wie ein normaler Hyperlink auf der Webseite: Wenn er darauf klickt, dann erwartet er, dass innerhalb der Anwendung eine Navigation erfolgt. Dieser Vorgang wird gewöhnlich als Manipulation des GWT-Objekts `History` kodiert, um so den Anwendungszustand zu verändern. Wie dies funktioniert, sehen Sie in der Dashboard-Komponentenanwendung *Slideshow* (`org.gwtbook.client.ui.slideshow.Slideshow`). Diese Anwendung bietet am unteren Rand zwei Hyperlinks, die dem Benutzer das Springen an den Anfang bzw. das Ende der Diashow gestatten (siehe Abbildung 4.6).

Eine Komponente, die ein `Hyperlink`-Widget verwendet, sollte auch die Schnittstelle `HistoryListener` erweitern und die Methode `onHistoryChange()` implementieren, um das Anklicken von Hyperlinks abzufangen und zu verwalten. Wie Sie sehen, implementiert die Dashboard-Komponente *Slideshow* zwei Hyperlinks, die im Code wie folgt aussehen:

```
Hyperlink startLink = new Hyperlink("Start", "0"); ❶  
Hyperlink endLink = new Hyperlink("End", ""+(maxNumberImages- 1)); ❷
```

Jeder `Hyperlink`-Widget-Konstruktor besteht aus zwei Elementen: dem auf dem Bildschirm anzuzeigenden Text und einem Verlaufs-Token (ein beliebiger String). Im Falle von



**Abbildung 4.6**  
Zwei `Hyperlink`-Widgets („Start“ und „End“) in der Dashboard-Anwendung *Slideshow*

*Slideshow* stellen alle Verlaufs-Token Zahlen eines Bildes dar. Das erste Bild hat die Nummer 0, das letzte einen Wert, der der Anzahl der Bilder in der Diashow minus 1 entspricht. Sie können einen `Hyperlink`, der auf den Anfang und das Ende der Diashow verweist, ganz einfach erstellen, indem Sie die passenden Werte im `Hyperlink`-Konstruktor verwenden: „0“ für den Anfang (❶) und die String-Darstellung der höchsten Bildnummer (❷).

Wenn Sie ein `History`-Objekt verwenden, vergessen Sie nicht, die folgende Zeile in den Textkörper Ihrer HTML-Seite einzufügen:

```
<iframe id="__gwt_historyFrame"
        style="width:0;height:0;border:0">
</iframe>
```

Andernfalls treten Fehler auf, weil der Verlaufs-Frame von GWT zum Speichern und Abrufen von Verlaufs-Token verwendet wird. Im Hostmodus erkennt man diesen Fehler an den entsprechenden Fehlermeldungen in der Hostmoduskonsole (vgl. Abbildung 4.7); im Webmodus erfolgt jedoch keine derartige Warnung.



**Abbildung 4.7** Fehler beim Versuch, das GWT-Subsystem `History` im Hostmodus ohne vorherige korrekte Initialisierung zu verwenden

(Die Rebinding-Meldung in Abbildung 4.7 ist Folge einer GWT-Manipulation, die Sie erst im weiteren Verlauf dieses Buchs kennenlernen werden. Hierbei wird mithilfe von GWT-Generatoren auf der Grundlage der Basiskomponentenanwendung automatisch neuer Code generiert, um das Menüelement ABOUT anzuzeigen.)

Mithilfe der Methoden in der Klasse `Hyperlink` können Sie das Verlaufs-Token ganz einfach mit einem bestimmten Link verknüpfen (`getTargetHistoryToken()`) oder es nach Bedarf ändern (`setTargetHistoryToken()`). Auf ähnliche Weise können Sie mit den Methoden `setText()` und `getText()`, die den `Hyperlink` als einfachen Text behandeln,

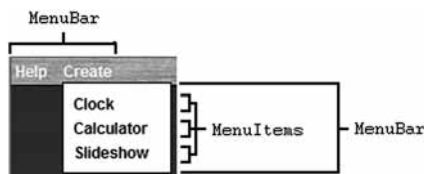
Letztere festlegen bzw. abfragen. Dies ist für HTML-Code auch mit den Methoden `setHTML()` und `getHTML()` möglich, falls Sie den Hyperlink mit dem Konstruktor `Hyperlink(String, boolean, String)` statt mit der einfacheren Variante `Hyperlink(String, String)` so erstellt haben, dass der Text als HTML-Code aufgefasst wird.

Die Behandlung von Hyperlinktext als HTML-Code hat zur Folge, dass der gesamte Auszeichnungscodex – z. B. für fett formatierten oder unterstrichenen Text oder Bilder – angezeigt wird. Wenn Sie einen normalen Hyperlink etwa auf eine andere HTML-Seite benötigen, verwenden Sie das Widget `HTML`, welches wir später betrachten werden, statt eines `Hyperlink`-Widgets, dessen Text auf HTML-Code festgelegt ist.

Hyperlinks stellen eine Möglichkeit dar, durch eine Anwendung zu navigieren; eine andere ist die Verwendung eines Menüsystems.

### Durch Ihre Anwendung mit Menüs navigieren

Das von GWT vermittelte Menüsystem basiert auf den Widgets `MenuBar` (für Menüleisten) und `MenuItem` (für Menüelemente). `MenuItem`-Widgets werden zu `MenuBar`-Widgets und diese wiederum zu anderen `MenuBar`-Widgets hinzugefügt – auf diese Weise entsteht das Menüsystem Ihrer Anwendung. Abbildung 4.8 zeigt das Menüsystem von *Dashboard*, wo die Menüelemente `CLOCK`, `CALCULATOR` und `SLIDESHOW` zu einer Menüleiste `CREATE` hinzugefügt wurden. Die Menüleiste `CREATE` sowie eine Menüleiste `HELP` werden dann zu einer anderen Menüleiste hingefügt, die auf der Browserseite angezeigt wird.



**Abbildung 4.8**

In der Anwendung *Dashboard* verwendete `MenuBar`- und `MenuItem`-Widgets. In diesem Beispiel werden drei Menüeinträge in einer Menüleiste abgelegt, die dann einer übergeordneten Menüleiste hinzugefügt werden.

In der Klasse `Dashboard` (`org.gwtbook.client.Dashboard`) definieren Sie mit dem folgenden Code genau eine globale Menüleiste:

```
MenuBar menu = new MenuBar();
```

Diese einfache Zeile erstellt eine horizontale Menüleiste. (Sie ließe sich ebenso einfach mit dem alternativen Konstruktor erstellen, der einen Booleschen Parameter entgegennimmt, dessen Wert auf `false` festgelegt ist, als etwa `new MenuBar(false)`. Ist der Parameter auf `true` festgelegt, dann wird eine vertikale Menüleiste erstellt.) Zum jetzigen Zeitpunkt weist das *Dashboard* zwei Standardmenüleisten auf. Zwei weitere werden Sie in diesem Buch erstellen, und weiter unten im Code zu *Dashboard* finden Sie zwei Methoden, die zur Erstellung der Menüleisten `create` und `help` eingesetzt werden. Die in Listing 4.1 gezeigte Methode `buildHelpMenu()` erstellt die ausgangssseitige Menüleiste `CREATE` mithilfe vertikaler Menüleisten.

**Listing 4.1** Erstellen der Menüleiste CREATE, der Menüelemente sowie der verschachtelten Menüleiste LOCALE

```

protected MenuBar buildHelpMenu(){
    MenuBar menuHelp = new MenuBar(true);
    MenuBar menuLocale = new MenuBar(true); // ❶ Vertikale Menüleisten erstellen

    menuLocale.addItem(constants.EnglishLocale(), ❷ Menüelement hinzufügen
        new ChangeLocaleToEnglishCommand());
    menuLocale.addItem(constants.SwedishLocale(),
        new ChangeLocaleToSwedishCommand());
    menuHelp.addItem(constants.LocaleMenuItemName(), menuLocale);
    return menuHelp;
}

```

Vertikale Menüleisten erstellen Sie durch Übergabe des Booleschen Wertes `true` als Parameter an den Konstruktor ❶. In einer Menüleiste werden ein oder mehrere Menüelemente oder andere Menüleisten zusammengefasst und ergeben so die in Abbildung 4.8 gezeigte optische Struktur. Es ist möglich, `MenuItem`-Widgets innerhalb des Codes zu erstellen; genau dies geschieht bei ❷, wo der erste Parameter für die Methode `addItem()` ein neues Menüelement ist. Jedes Menüelement ist an ein Codesegment gebunden, das bei Anklicken des Menüelements ausgeführt wird: den zweiten Parameter für die Methode `addItem()`.

Mithilfe des *Befehlsmuster* beschreiben Sie den Befehl, der ausgeführt wird, wenn ein Menüelement angeklickt wird. In der Praxis bedeutet dies, dass Sie eine neue Instanz der GWT-Klasse `Command` oder einer davon abgeleiteten Klasse erstellen, die eine Methode `execute()` enthält, in der der Code gespeichert ist. Für das Dashboard wollen wir mehrere von `Command` abgeleitete Klassen als innere Klassen der Anwendung *Dashboard* erstellen, da diese Vorgehensweise unseren Anforderungen am besten entspricht. Ein Beispiel für eine dieser Klassen zeigt Listing 4.2 (dieser Befehl wird an die in Schritt ❷ von Listing 4.1 definierten Menüelemente angehängt).

**Listing 4.2** Von der Klasse `Command` abgeleitete Klasse zum Umschalten auf das englische Gebietschema

```

class ChangeLocaleToEnglishCommand implements Command{
    public void execute(){
        Window.removeWindowCloseListener(dashboardWindowCloseListener);
        changeLocale("");
    }
}

```

Mit dem Befehlsmuster kann die GWT-Anwendung eine Anforderung zukünftiger Funktionalität in ein Objekt umwandeln, das sich im ganzen Code einsetzen lässt; die definierte Funktionalität kann dann später aufgerufen werden. Im Falle der Menüelemente stellt GWT das erforderliche Drumherum bereit, damit bei Anklicken eines Menüelements die zugehörige Methode `execute()` aufgerufen wird. In diesem Beispiel wird, wenn der Benutzer ein Menüelement anklickt, dessen Befehl auf eine Instanz der Klasse `ChangeLocaleToEnglishCommand` festgelegt ist, diese Methode `execute()` aufgerufen und ein `WindowCloseListener` aus der Anwendung entfernt, bevor die Methode `changeLocale()` aufgerufen wird. (Andernfalls würden Sie die beiden in Kapitel 6 eingerichteten Ereignisse zum Schließen des Fensters aufrufen; im Falle des Dashboards zeigen diese Ereignis-

Handler zwei Meldungsfenster an, was nicht passieren sollte, wenn Sie lediglich das Gebietsschema wechseln wollen.)

Sie können mithilfe der Methode `getParentMenu()` bei einem Menüelement erfragen, welches Menü ihm übergeordnet ist; mit der Methode `getSubMenu()` finden Sie zudem heraus, ob über das Element ein Untermenü geöffnet wird. Ähnlich können Sie das Untermenü eines `MenuItem`-Widgets über die Methode `setSubMenu()` festlegen; für ein übergeordnetes Menü ist dies jedoch nicht möglich. In einigen Anwendungen müssen Sie den mit einem bestimmten Menüelement verknüpften Befehl unter Umständen mit `setCommand()` ändern oder aber mit `getCommand()` ermitteln, wie der Befehl heißt.

Der letzte Implementierungsaspekt eines `MenuItem`-Widgets, den wir behandeln wollen, bezieht sich auf den Text, als der das Element angezeigt wird. Dieser Text kann abhängig davon, ob im Konstruktor von `MenuItem` ein Boolescher Parameter vorhanden ist oder nicht, als reiner Text oder als HTML-Code behandelt werden. Wie bei jedem Widget, das die Festlegung von HTML-Code gestattet, sollten Sie auch hier sorgfältig darauf achten, dass keine skriptbasierten Sicherheitsprobleme in Verbindung mit der Anwendung entstehen. Das Erstellen eines Menüelements mit `new MenuItem(String, true, Command)` behandelt den String als HTML-Code; wird hingegen einer der Konstruktoren `MenuItem(String, false, Command)` oder `MenuItem(String, Command)` verwendet, erfolgt die Behandlung als reiner Text.

Ein `MenuBar`-Widget erlaubt die Festlegung, ob sich untergeordnete Menüs automatisch öffnen oder aber abwarten, bis der Benutzer darauf klickt, um sie zu öffnen. Dies wird durch die Methode `setAutoOpen()` bestimmt. Beim Dashboard tun Sie dies in der Methode `onModuleLoad()`, wo Sie das gesamte Menüsystem mithilfe des in Listing 4.3 gezeigten Codes erstellen.

**Listing 4.3** Erstellen des Menüsystems der Anwendung *Dashboard*

```
MenuBar menuCreate = buildCreateMenu(); // Untermenüs erstellen
MenuBar menuHelp = buildHelpMenu();
menu.addItem(constants.HelpMenuName(), menuHelp); // Untermenüs zur
menu.addItem(constants.CreateMenuName(), menuCreate); // Menüleiste hinzufügen
menu.setAutoOpen(true); // Automatisches Öffnen der Untermenüs aktivieren
```

Oben auf dem Bildschirm befindet sich das Menüsystem des Dashboards (vgl. Abbildung 4.9), welches mehrfach im Code vorhanden ist. Die Menüleisten `CREATE` und `HELP` werden als einfache Implementierungen für eine Internetansicht (in der Klasse `Dashboard`) erstellt und dann (in der Klasse `Dashboard_intranet`) überschrieben, um eine Intranetversion mit mehr Funktionen bereitzustellen. Ob die Intranet- oder die Internetversion verwendet wird, bestimmt man mithilfe benutzerdefinierter GWT-Eigenschaften und durch Festlegung der benutzerdefinierten Eigenschaft `externalvisibility` in der Datei `Dashboard.html`. (All dies behandeln wir in Kapitel 15.)

Der Text dieser beiden Menüleisten wird mithilfe der GWT-Internationalisierungskonstanten erstellt, die für ein Standardgebietsschema (Englisch) sowie ein alternatives Schema (Schwedisch) konfiguriert sind. Diesen Internationalisierungsansatz, den wir bereits in Kapitel 3 kurz angerissen haben, behandeln wir in Kapitel 15 ausführlich.



**Abbildung 4.9** Das Dashboard-Menüsystem mit den vier möglichen Menüleisten. Die Menüleisten HELP und CREATE sind stets vorhanden, das Menü BOOKMARKS wird als XML-Code vom Server geladen, und die Optionsmenüleiste wird bei Aktivierung einer Komponentenanwendung angezeigt.

Sie verwenden im Menüsystem von *Dashboard* außerdem die beiden neuen `MenuItem`-Widgets, die Sie im weiteren Verlauf dieses Kapitels erstellen werden. Wenn das Dashboard im Intranetmodus läuft, kann der Benutzer über das Menü HELP die Bestätigung aktivieren oder deaktivieren, die angefordert wird, wenn eine Komponentenanwendung gelöscht werden soll. Dies erfolgt über ein `TwoComponentMenuItem`. In beiden Modi kann der Benutzer das Gebietsschema über zwei `TwoComponentMenuItem`-Widgets ändern: eines pro von der Anwendung unterstütztem Gebietsschema. Abbildung 4.10 zeigt die Intranetansicht des Menüs HELP, wo die beiden neuen Widgets im Einsatz sind.



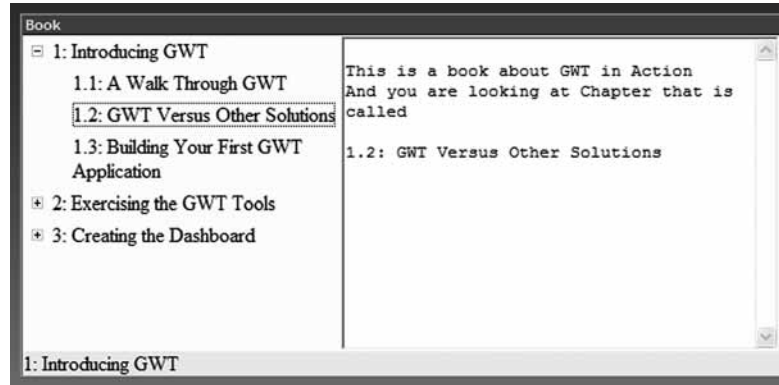
**Abbildung 4.10** Menüsystem des Dashboards mit den beiden neuen `MenuItem`-Widgets, die in diesem Kapitel erstellt werden. Die Menüelemente für das englische und das schwedische Gebietsschema sind Instanzen von `TwoComponentMenuItem`, das Element CONFIRM DELETE eine Instanz von `ToggleMenuItem` (wird mit dem Status EIN angezeigt).

Es gibt zwei weitere Möglichkeiten zur Manipulation der Menüleiste. Zunächst erstellen Sie eine Menüleiste für Lesezeichen, deren Inhalt mithilfe der GWT-Implementierung des `XMLHttpRequest`-Objekts aus einer XML-Datei geladen wird (dies behandeln wir in Kapitel 12). Zweitens kann jede Komponentenanwendung ein Optionsmenü registrieren, welches im Hauptmenü angezeigt wird, sobald die Anwendung den Fokus erhält. Hierzu verwenden Sie einen GWT-Generator, um für jede Anwendung automatisch ein Element `About` im Optionsmenü zu erstellen, das dann alle Methoden und Felder auflistet, die in der Anwendung enthalten sind (GWT-Generatoren werden in Kapitel 14 besprochen). Abbildung 4.10 zeigt das Optionsmenü der Komponentenanwendung *Google Search*. An diese Komponentenanwendungen werden auch einige universelle Funktionsanforderungen gestellt.

Der letzte Punkt, der in Verbindung mit dem `MenuBar`-Widget zu erwähnen ist, ist die Tatsache, dass es die Schnittstelle `PopupListener` implementiert, mit deren Hilfe beim Schließen der Menüleiste eine bestimmte Funktionalität realisiert werden kann. Wenn Sie beim Schließen des `MenuBar`-Widgets eine andere als die Standardfunktionalität verwenden wollen, können Sie die vorhandene Klasse überschreiben und eine eigene Methode `onPopupClosed()` implementieren. (Im Dashboard werden wir diese Funktionalität allerdings nicht verwenden.)

### Datenansicht mit Baumstrukturen verwalten

Wir haben unsere Beschreibung der grundlegenden GWT-Widgets beinahe abgeschlossen. Zwei weitere stehen noch aus, deren erstes `Tree` ist. Dieses Widget stattet Anwendungen mit einer konventionellen hierarchischen Baumstruktur aus, die aus `TreeItem`-Widgets besteht (siehe Abbildung 4.11).



**Abbildung 4.11** Die Dashboard-Anwendung *Book* mit dem `Tree`-Widget auf der linken Seite. In Kapitel 6 erfahren Sie, wie man Ereignisse verwendet, wenn `TreeItem`-Widgets selektiert oder erweitert werden.

Ein `Tree` ist ähnlich aufgebaut wie das Menü weiter oben. Dort haben Sie `MenuItem`-Widgets einem `MenuBar`-Widget hinzugefügt; hier fügen Sie `TreeItem`-Widgets zu einem `Tree`-Widget hinzu. Die Konstruktoren für ein `TreeItem` sind flexibel und gestatten Ihnen die Erstellung eines oder mehrerer leerer `TreeItem`-Widgets aus Strings oder anderen Widgets (d. h. aus Standard- oder zusammengesetzten Widgets). In Listing 4.4, welches der Anwendung *Book* des Dashboards entnommen ist (`org.gwtbook.client.ui.book.Book`), erstellen Sie eine einfache Baumstruktur zur Darstellung der obersten Überschriftenebene dieses Buchs.

**Listing 4.4** Erstellen des Baumsystems in der Anwendung *Dashboard*

```
private Tree buildTOC() {
    TreeItem chapter1 = new TreeItem("1: Introducing GWT"); // Struktur für
    chapter1.addItem("1.1: A Walk Through GWT");           // Kapitel 1, ...
    chapter1.addItem("1.2: GWT Versus Other Solutions");
    :
    TreeItem chapter2 = new TreeItem("2: Exercising the GWT Tools"); // ...Kapitel 2, ...
    chapter2.addItem("2.1: Setting up Dashboard Version 1");
    :
    TreeItem chapter3 = new TreeItem("3: Creating the Dashboard"); // ...Kapitel 3 und ...
    chapter3.addItem("3.1: Stage 2 - Developing the Application");
    :
    Tree t = new Tree();
    t.addItem(chapter1); t.addItem(chapter2); t.addItem(chapter3); // ... Inhaltsverzeichnis erstellen
    return t;
}
```

Anders als beim Menüsystem fügen Sie zu `TreeItem`-Widgets keine Befehle hinzu, um Funktionalität für das Anklicken oder Erweitern zu implementieren; stattdessen implementieren Sie einen `TreeListener`. Zu diesem Zweck müssen Sie zwei Methoden implementieren: eine Methode `onTreeItemSelected()`, die bei Selektion eines `TreeItem`-Widgets aufgerufen wird, und die Methode `onTreeItemStateChanged()`, die aufgerufen wird, wenn sich der Status (geöffnet oder geschlossen) eines `TreeItem` ändert.

Bei der Dashboard-Anwendung *Book* implementieren Sie den `TreeListener` wie in Listing 4.5 gezeigt.

**Listing 4.5** Hinzufügen einer Ereignisbehandlung zur Baumstruktur der Dashboard-Anwendung *Book* zwecks Ändern des Anzeigetexts

```
Tree theTree = buildTOC();
theTree.addTreeListener(new TreeListener() {
    public void onTreeItemSelected(TreeItem item) { // ❶ Aufruf bei Auswahl von
                                                    TreeItem

        changeText(item.getText()); // ❷ Text aus gewählttem Element holen
    }
    public void onTreeItemStateChanged(TreeItem item) { // ❸ Aufruf, wenn der Status
                                                         von TreeItem sich ändert

        if (item.getState()) { // ❹ Zustand des geänderten Elements holen

            currChapter.setText(item.getText()); // ❺ Text abrufen
        } else {
            currChapter.setText("-----");
        }
    }
});
```

Dieser Listener ruft die Methode `changeText()` ❷ der Anwendung *Dashboard* auf, um das Textfeld rechts in der Anwendung *Book* mit Text zu füllen, wenn ein `TreeItem` ausgewählt wird ❶. Wenn sich der Status eines `TreeItem`-Widgets ändert, wird die Methode `onTreeItemStateChanged()` aufgerufen ❸. Diese Methode fordert den Status des `TreeItem` an, welches geändert wurde ❹, und setzt, sofern dieses Element nun geöffnet ist, den Text am unteren Rand des Widgets ein, indem es ihn mit der Methode `getText()` abrufen ❺. Andernfalls werden Striche als Text eingesetzt.

Zum Widget `TreeItem` gehört eine ganze Anzahl von Hilfsmethoden, die Sie aufrufen können, um mehr zu erfahren oder vorhandene Eigenschaften des Elements zu ändern. Sie können das untergeordnete Element zu einem bestimmten Index ermitteln (`getChild(int index)`), die Anzahl der untergeordneten Elemente zählen (`getChildCount()`) oder den Index eines bestimmten untergeordneten Elements erfragen. Ferner können Sie die Baumstruktur ermitteln, deren Bestandteil ein bestimmtes `TreeItem` ist (`getTree()`), herausfinden, ob untergeordnete Elemente eines Elements gegenwärtig angezeigt werden (`getState()`), das übergeordnete Element bestimmen (`getParentItem()`) und sich erkundigen, ob das gewählte Element gegenwärtig selektiert ist (`isSelected()`).

Ein `TreeItem` kann mit einem Widget verknüpft sein; ist dies der Fall, kann es nicht direkt mit Text verbunden sein (es sei denn, es ist als zusammengesetztes Widget konfiguriert).

Häufig wird man beispielsweise ein `CheckBox`-Widget mit einem `TreeItem` verknüpfen. Sie können ein Widget wahlweise über die Methode `setWidget()` oder mithilfe des Konstruktors `TreeItem(Widget)` mit einem `TreeItem` verknüpfen.

Die verschiedenen Zustände eines `TreeItem` – ob also ein Element geschlossen oder geöffnet ist – werden als Bilder angezeigt. Standardmäßig befinden sich diese Bilder im öffentlichen Ordner Ihrer Anwendung und heißen `tree_closed.gif`, `tree_open.gif` und `tree_white.gif`. Sie können diese Bilder nach Bedarf durch eigene Versionen ersetzen, lediglich die Namen müssen dieselben bleiben. Wenn Sie die Bilder in einem anderen Verzeichnis speichern wollen, geben Sie den Speicherort mithilfe der Methode `setImageBase()` an.

### Bilder anzeigen

Wenn Sie ein Bild in einer GWT-Anwendung anzeigen wollen, können Sie das Basis-Widget `Image` verwenden. Dieses Widget ermöglicht das Laden und Anzeigen von Bildern (vgl. Abbildung 4.12).



**Abbildung 4.12**  
Widget `Image` in der Dashboard-Anwendung *Slideshow*

Ein interessanter Aspekt des `Image`-Widgets ist die Möglichkeit, ihm einen `LoadListener` hinzuzufügen, sodass eine bestimmte Aktion durchgeführt werden kann, wenn das Laden des Bildes abgeschlossen ist (`onLoad()`) oder beim Laden des Bildes ein Fehler aufgetreten ist (`onError()`).

#### Tipp

Der `LoadListener` funktioniert nur, wenn man das `Image`-Widget zur Browserseite hinzufügt. Dies erfolgt gewöhnlich über ein Panel, welches selbst zum Browser hinzugefügt wird. Das einfache Erstellen des Java-Objekts und das Hinzufügen eines `LoadListener` reichen nicht aus, um die Ereignisse abzufangen; dies liegt an der Funktionsweise des GWT-Systems zur Ereignisbehandlung (siehe Kapitel 6).

Listing 4.6 zeigt den Code der Dashboard-Komponente *Slideshow* (`org.gwt.client.ui.slideshow.Slideshow`), mit dem sich Bilder vorab in eine Anwendung laden lassen.

**Listing 4.6** Vorabladen von Bildern für eine Diashow mit einem `LoadListener`

```

Image[] testLoading = new Image[maxNumberImages];
preloadImages.setVisible(false); ❶ Panel ausblenden
for(int loop=0;loop<maxNumberImages;loop++){
    testLoading[loop] = new Image(theImages[loop][1]);
    testLoading[loop].addLoadListener(new LoadListener(){ ❷ LoadListener hinzufügen

        public void onError(Widget sender) { ❸ onError-Code definieren
            Window.alert("Expected Error - onError() Method works.");
        }
        public void onLoad(Widget sender) { ❹ onLoad-Code definieren
            Window.setTitle("Loaded Image: "+imageName);
        }
    });
    preloadImages.add(testLoading[loop]);
}

```

Das Objekt `preloadImages` ist ein `HorizontalPanel`, welches Sie an dieser Stelle gezielt der Anwendung hinzufügen, um später Bilder vorab laden zu können. Aufgrund der Funktionsweise des GWT-Ereignismechanismus müssen Sie die Bilder in eine Komponente laden lassen, die zur Browserseite hinzugefügt wird (andernfalls ist kein Einsprung in den Ereignismechanismus möglich, und der `LoadListener` wird ignoriert). Allerdings ist es nicht erforderlich, dass die Komponente sichtbar ist, weswegen Sie sie bei ❶ ausblenden, um ein unansehnliches Durcheinander zu vermeiden!

Bei ❷ fügen Sie einen neuen `LoadListener` hinzu und definieren die Methode `onError()` so, dass bei Auftreten eines Fehlers ein JavaScript-Hinweis auf dem Bildschirm erscheint ❸; wird das Bild erfolgreich geladen, ändern Sie die Titelleiste des Browserfensters so ab, dass mit der Methode `window.setTitle()` der Bildname angezeigt wird ❹.

Ist ein `Image`-Objekt vorhanden, dann lässt sich mit den Methoden `prefetch()` oder `setUrl()` ein neues Bild laden, statt – wie im Beispiel gezeigt – ein neues Objekt erstellen. Beide Vorgehensweisen sind zulässig, und Sie werden sich entscheiden müssen, welche Möglichkeit Sie zur Einrichtung in Ihren eigenen Anwendungen einsetzen werden.

Ferner sollten Sie sich darüber im Klaren sein, dass, wenn Sie in Internet Explorer 5.5 oder 6 ein transparentes PNG-Bild über ein anderes Bild legen, dies hässliche Auswirkungen auf den Seitenhintergrund hat. Wir werden im weiteren Verlauf dieses Kapitels ein neues Widget erstellen, das dieses Problem beseitigt.

Eine spannende neue Funktion in GWT 1.4 ist die Möglichkeit, Bilder zu bündeln und zu beschneiden. Die ursprüngliche Klasse `Image` erhielt dabei eine neue Methode, die es Ihnen gestattet, ein Bild auch nur teilweise anzuzeigen. Um beispielsweise nur die oberen linken  $50 \times 50$  Pixel eines Bildes anzuzeigen, können Sie den Konstruktor `new Image("image.png", 0, 0, 50, 50)` verwenden.

Letztendlich besteht der Zweck der Bildbeschneidung in der Optimierung der Ladezeit für Ihre GWT-Anwendung. Das Laden eines Bildes, welches mehrere verwendete Bilder enthält, ist effizienter als das separate Laden jedes einzelnen Bildes. Betrachten wir eine HTML-Seite, die fünf Bilder lädt, welche in der Dashboard-Anwendung *Events Widget Browser* angezeigt werden. Wenn Sie alle fünf Bilder separat auf eine einzelne Seite laden, meldet die Firefox-Erweiterung *Firebug* die in Abbildung 4.13 gezeigten Zeitangaben.

Request	Size	Time
ChangeEvents.PNG	436 b	110ms
ClickEvents.PNG	328 b	110ms
FocusEvents.PNG	506 b	100ms
LoadEvents.PNG	516 b	100ms
MouseEvents.PNG	484 b	90ms
<b>5 requests</b>	<b>3 KB</b>	<b>130ms</b>

Abbildung 4.14 Ladezeiten für fünf separate Bilddateien

Die fünf Bilder benötigen für den Download ca. 130 ms. Die Bilder stammen von einem Webserver, der für den Computer lokal ist; betrachten Sie jedoch einmal die Ergebnisse in Abbildung 4.14, wo wir ein Bild, das alle fünf Bilder auf einmal enthält, manuell erstellt haben.

Request	Size	Time
AllEvents.PNG	2 KB	50ms
<b>1 request</b>	<b>2 KB</b>	<b>50ms</b>

Abbildung 4.15 Alle fünf Bilder, zusammengefasst auf einem Bild

Jetzt sind wir schon bei 50 ms. Wenn Sie diesen Wert auf eine größere Anwendung hochrechnen, die Bilder über ein echtes Netzwerk lädt, erahnen Sie vielleicht, dass das Bündeln von Bildern erhebliche Vorteile mit sich bringt. Es gibt natürlich auch einen Nachteil: Wollen Sie wirklich alle Bilder, die Sie einsetzen, per Hand in eine einzige Bilddatei einfügen? Zum Glück gibt es die Klasse `ImageBundle`, die dieses Problem umgeht. Sie stellen Ihre Bilder wie gewünscht bereit und erstellen dann eine Schnittstelle, die `ImageBundle` erweitert und Ihre Bilder referenziert; GWT puzzelt dann alles für Sie zusammen. Das wollen wir uns genauer ansehen.

Im Ordner `Public` der Anwendung `Dashboard` erstellen Sie einen Ordner `ToolBarImages`, in dem Sie alle Einzelbilder der Symbolleiste ablegen. Danach erstellen Sie im Package `org.gwtbook.client.ui.EventsWidgetBrowser` eine neue Schnittstellenklasse, die `ImageBundle` erweitert; Listing 4.7 zeigt ein Beispiel dieser Schnittstelle.

Listing 4.7 Ein `ImageBundle` erstellen

```
public interface ToolBarImageBundle extends ImageBundle{
    /**
     * @gwt.resource org/gwtbook/public/ToolBarImages/ChangeEvents.png ❶
     */
    AbstractImagePrototype ChangeEvents(); ❷

    /**
     * @gwt.resource org/gwtbook/public/ToolBarImages/ClickEvents.png
     */
    AbstractImagePrototype ClickEvents();
}
```

Die Schnittstelle definiert eine Methode ❷ für jedes Bild, das gebündelt werden soll, und die von GWT generierten Implementierungen dieser Methoden werden verwendet, um die beschnittenen Bilder zu extrahieren. Standardmäßig entnimmt GWT die Bilder den Pack-

ages, in denen die Bündelungsschnittstelle definiert ist (dies ist ein Unterschied zu anderen Bildern und/oder Ressourcen) und sucht nach einem Bild, das den Namen der Methode trägt. Um das Zusammenspiel mit Ihrer Anwendungsstruktur zu optimieren, können Sie GWT anweisen, die Ressource einem Speicherort Ihrer Wahl zu entnehmen. Dies ist durch Einfügen von Annotationen in die Kommentare möglich. So weist etwa **1** den Compiler an, das von der Methode `changeEvents()` zurückgegebene Bild dem Verzeichnis `ToolBarImages` zu entnehmen.

Die Verwendung des gebündelten Bildes ist einfach. Listing 4.8 zeigt das Erstellen einer Instanz Ihres Bundles unter Verwendung des Deferred Binding. Sie verschaffen sich Zugang zum Bild-Bundle, rufen die für das zu verwendende Bild erforderlichen Methoden auf und erhalten einen `AbstractImagePrototype` zurück. Um das benötigte Image zu erhalten, rufen Sie dann die Methode `createImage()` von `AbstractImagePrototype` auf. Nun verfügen Sie über ein Standardbild, das Sie Ihrer Anwendung hinzufügen können.

**Listing 4.8** Creating an ImageBundle

```
ToolBarImageBundle toolBarImages =
    (ToolBarImageBundle)GWT.create(ToolBarImageBundle.class); // Bundle-Objekt
                                                                // erstellen

AbstractImagePrototype changeEventImagePrototype =
    toolBarImages.changeEvents(); // Bildprototyp erstellen
Image changeEventImage = changeEventImagePrototype.createImage() // Bild erstellen
```

Aber Bilder sind noch nicht alles, was man in einer Anwendung anzeigen kann. Es gibt auch zahlreiche Möglichkeiten, Text darzustellen.

## 4.2.2 Text in der Anwendung anzeigen

Das soeben betrachtete `Image`-Widget ist zum Anzeigen von Bildern und Grafiken im Webbrowser durchaus nützlich. Viele Anwendungen müssen jedoch Text entweder als passive Information oder aber auf aktivere und interessantere Weise darstellen. Wenn wir uns nun weiter durch die Widget-Hierarchie hangeln, stoßen wir auf zwei Widgets, die Text auf dem Bildschirm darstellen können: `Label` und `HTML`.

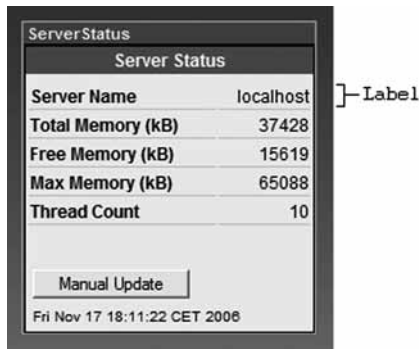
### Text als Beschriftung anzeigen

Ein `Label`-Widget enthält beliebigen Text, der genau so angezeigt wird, wie er angegeben ist. So erscheint etwa die mit dem Code `new Label("Hi <B>there</B>")` erstellte Beschriftung auf der Browserseite als „Hi <b>there</b>“, d. h. das Wort *there* wird nicht als HTML-Code interpretiert und auch nicht fett ausgezeichnet.

Es ist möglich, die horizontale Anordnung von Beschriftungen zu steuern, auch wenn die Größe einer Beschriftung standardmäßig der Größe des enthaltenen Texts entspricht. Die rechtsbündige Anordnung mithilfe des Befehls

```
theLabel.setHorizontalAlignment(HorizontalAlignmentConstant.ALIGN_RIGHT)
```

wirkt sich optisch erst aus, wenn Sie mithilfe eines Stylesheets (oder der Methode `TextLabel.setWidth()`, was aber nicht vorzuziehen ist) die Breite der Beschriftung auf einen höheren Wert als die Textbreite stellen. Das in der Anwendung *Server Status* des Dashboards (`org.gwtbook.client.ui.serverstatus.ServerStatus`, vgl. Abbildung 4.15) gezeigte Layout wird durch Ausrichten der Beschriftungen in einem `Grid`-Panel ermöglicht (siehe auch Kapitel 5). Eine Beschriftung kann auch einen Zeilenumbruch aufweisen, sofern die Methode `setWordWrap()` aufgerufen wird. Diese Methode nimmt eine Boolesche Variable entgegen, die auf `true` festgelegt ist, wenn die Beschriftung umbrochen werden soll (andernfalls `false`).



**Abbildung 4.15**  
Dashboard-Anwendung *Server Status* mit den GWT-Beschriftungs-Widgets im Einsatz

GWT gestattet Ihnen das Hinzufügen eines `ClickListener` und eines `MouseListener` zu einem `Label`-Standard-Widget, was die Möglichkeit der Interaktion zwischen Benutzer und Widget eröffnet. Standardmäßig erfolgt keine Aktion – Sie müssen zunächst `ClickListener` und/oder `MouseListener` hinzufügen. In dem Widget `EditableLabel` (`org.gwtbook.client.ui.EditableLabel`), das wir in Kapitel 7 erstellen, wird ein Textfeld anstelle der Beschriftung angezeigt, wenn ein Benutzer die Beschriftung anklickt. Auf diese Weise kann man den Text der Beschriftung ändern. Sie fügen den `ClickListener` wie in Listing 4.9 gezeigt hinzu.

**Listing 4.9** Hinzufügen eines `ClickListener` zum GWT-Widget `EditableLabel`

```

text = new Label(labelText);
text.setStyleName("editableLabel-label");
text.addClickListener(new ClickListener()
{
    public void onClick (Widget sender){
        changeTextLabel();
    }
});

```

Der `ClickListener` agiert ähnlich wie die Klasse `Command`, die Sie für das Widget `MenuItem` verwendeten. Er registriert eine Methode `onClick()`, die GWT ausführt, wenn der Benutzer der Anwendung auf die Beschriftung klickt.

Bei einer vorhandenen Beschriftung können Sie den Text programmgesteuert mithilfe der Methode `setText()` ändern und mit `getText()` den aktuellen Text abrufen (siehe Listing 4.10).

**Listing 4.10** Ändern des Beschriftungstexts in der Anwendung *Clock* über die Methode `setText()`

```
Date d = new Date();
if (! local) {
    d = new Date(d.getTime() - (d.getTimezoneOffset() * 60 * 1000));
}
clockLabel.setText(d.getHours() + " || Beschriftung auf neue Uhrzeit setzen
    ":" + twoDigit(d.getMinutes()) +
    ":" + twoDigit(d.getSeconds()));
```

Sie können aber auch ein in punkto Textdarstellung etwas aktiveres Widget verwenden: `HTML`.

### Text aktiv anzeigen mit dem `HTML`-Widget

Wenn Sie Ihren Text ansprechender darstellen wollen, ist das `HTML`-Widget unter Umständen genau die Komponente, nach der Sie suchen. Dieses Widget agiert auf die gleiche Weise wie `Label`, aber – und das ist wichtig – interpretiert beliebigen Text als `HTML`-Code. Während bei einer Beschriftung der Text „Hi `<B>there</B>`“ so ausgegeben wird, wie er angegeben ist, wird für den Code `new HTML("Hi <B>there</B>")` der Text als „Hi **there**“ ausgegeben.

Das Widget `HTML` ist außerdem nützlich, wenn es um die Realisierung echter Hyperlinks geht. Als wir weiter oben das Widget `Hyperlink` erörterten, wiesen wir darauf hin, dass man dem Benutzer zwar etwas anzeigen kann, das wie ein Hyperlink aussieht, aber lediglich den Verlaufsaspekt der Anwendung ändert, wenn es angeklickt wird. Verwenden Sie stattdessen das Widget `HTML`, dann können Sie richtige Links bauen, wie wir es später in der Anwendung *About* tun werden, um auf die Webseiten zu diesem Buch zu verweisen:

```
new HTML("<a href='http://www.manning.com/hanson'>Manning</a>");
```

Sie müssen mit diesem Widget vorsichtig zu Werke gehen, weil bei der Ausführung beliebigen `HTML`-Codes Sicherheitsrisiken für Ihre Anwendung entstehen können, wenn böswillig konstruierter Code im Spiel ist. Berücksichtigen Sie ferner die Frage, ob das in Kapitel 5 behandelte `HTML`-Panel unter Umständen Ihren Bedürfnissen mehr entspricht.

Beschriftungen und `HTML` sind sehr gut zur Darstellung von Informationen geeignet, doch gibt es bei der Interaktion mit dem Benutzer noch eine andere Seite der Medaille: die Erfassung von Benutzereingaben.

### 4.2.3 Benutzerinteraktion mit Fokus-Widgets erfassen

Zum Erfassen von Benutzereingaben bietet GWT eine Reihe von Widgets, die alle in die in Abbildung 4.3 gezeigte Kategorie der Fokus-Widgets fallen. Diese Widgets erben von der Klasse `FocusWidget`. Bevor wir sie uns aber genauer ansehen, sollten Sie sich vergegenwärtigen, dass `FocusWidget` kein Widget im üblichen Sinne ist: Sie können keine Instanz hiervon erstellen und im Browser anzeigen. Es trifft zwar zu, dass `FocusWidget` die Klasse `Widget` erweitert, doch besteht ihr einziger Zweck darin, eine Klasse zur Verfügung zu stellen, die Fokus-, Klick- und Tastaturereignisse behandelt und so erweiterbar ist, dass sie die erforderliche allgemeine Funktionalität für untergeordnete Klassen bereitstellt. Insofern handelt es sich um eine *abstrakte* Klasse.

Wiewohl abstrakt, bietet `FocusWidget` die allgemeine Funktionalität für alle Fokus-Widgets. Hierzu gehört das Festlegen der Widget-Position im Tabulatorindex des Browsers mithilfe der Methode `setTabIndex()`. Der Tabulatorindex eines Widgets gibt die Reihenfolge an, in der die Widgets markiert werden, wenn der Benutzer mithilfe der Tabulatortaste durch die Elemente der Webseite navigiert. Das Festlegen von Tabulatorindizes ist besonders sinnvoll für Benutzer, wenn Widgets innerhalb eines Formulars verwendet werden. Insofern sind die Standardkomponenten für Formulare von `FocusWidget` abgeleitete Klassen.

Sie können auch die Methode `setAccessKey(char key)` verwenden, um eine bestimmten Tastenkombination zuzuweisen, die das Widget in den Fokus setzt, wenn sie betätigt wird. Alternativ lässt sich dies auch programmgesteuert über die Methode `setFocus()` bewerkstelligen. Im Widget `EditableLabel`, welches wir in Kapitel 7 erstellen werden, ändern Sie ein `Label`-Widget zur Bearbeitung in ein `TextArea`-Widget. Nach dem Austausch des Widgets müssen Sie gewährleisten, dass die `TextArea` sofort im Fokus liegt; hierzu rufen Sie die Methode `setFocus()` auf (siehe Listing 4.11).

**Listing 4.11** Definieren des `TextArea`-Widgets, sodass der Fokus auf dem Widget `EditableLabel` liegt

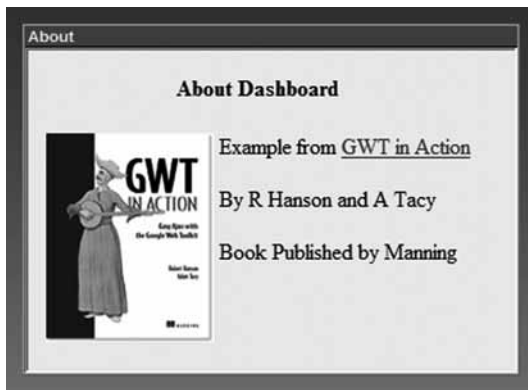
```
changeTextArea.setText(originalText); // Text aus FocusWidget festlegen
changeTextArea.setVisible(true); // Aus UIObject geerbte Sichtbarkeit festlegen
changeTextArea.setEnabled(true); // FocusWidget als aktiviert festlegen
changeTextArea.setFocus(true); // von FocusWidget geerbten Fokus festlegen
```

In GWT erweitern zwei Widgets `FocusWidget` direkt, um neue Widgets zu generieren: `Frame` und `ListBox`. Diese beiden sowie zwei weitere abstrakte, von `FocusWidget` abgeleitete Klassen namens `TextBoxBase` und `ButtonBase`, denen ihrerseits weitere Widgets untergeordnet sind, sehen wir uns als Nächstes an.

### Interessenbereich in `Frame` stellen

Im Dashboard wollten wir die Komponentenanwendung *About* (`org.gwtbook.client.ui.about>About`) ein wenig flexibler gestalten. Statt die Informationsmeldung fest einzukodieren, legten wir fest, dass sie aus einer separaten HTML-Datei geladen werden soll. Um diese Funktionalität zu unterstützen, verwenden wir das `Frame`-Widget, in das sich problemlos eine neue HTML-Seite laden lässt. Sie können dies entweder über den Konstruktor (z. B. `new Frame("resource-to-load")`) oder mithilfe der Widget-Methode `frame.setURL("resource-to-load")` tun.

Abbildung 4.16 zeigt die grundlegende Form der Komponentenanwendung *About* im Dashboard, deren Inhalte aus einer anderen HTML-Datei geladen werden (im Beispielcode wird hierzu der Dateiname *About.html* verwendet). Weil das Widget ein `IFRAME`-DOM-Element darstellt, kann dieser Inhalt ganz einfach aus einer statischen HTML-Datei, aus JSP-Code, einem Servlet, einem PHP-Skript, einer .NET-Anwendung usw. erstellt werden.

**Abbildung 4.16**

Verwendung des `Frame`-Widgets im Dashboard-Dialogfeld *About*. Der angezeigte Inhalt wird aus einer anderen HTML-Datei in einen `Frame` geladen.

Eine von `Frame` abgeleitete Klasse ist das Widget `NamedFrame`, mit dessen Hilfe Sie einen Namen zuweisen können. Diese benannten Frames werden gewöhnlich als Ziel für ein `FormPanel` verwendet. Sie werden sie in Kapitel 12, in dem es um Formulare geht, näher kennenlernen.

### Listenoptionen

Das Widget `ListBox` wird in der Dashboard-Anwendung *Address Book* (`org.gwtbook.client.ui.addressbook.AddressBook`) wie in Abbildung 4.17 gezeigt umfassend eingesetzt. Es stellt dem Benutzer – wahlweise als Dropdownliste oder als Standardlistenfeld – eine Anzahl von Optionen dar. In der Anweisung *Address Book* verwenden Sie das Listenfeld zur Auswahl von Namen und die Dropdownliste zur Auswahl des Landes.

Um ein Listenfeld im normalen Format zu erstellen, verwenden Sie den Standardkonstruktor `new ListBox()` wie in Listing 4.12 gezeigt.

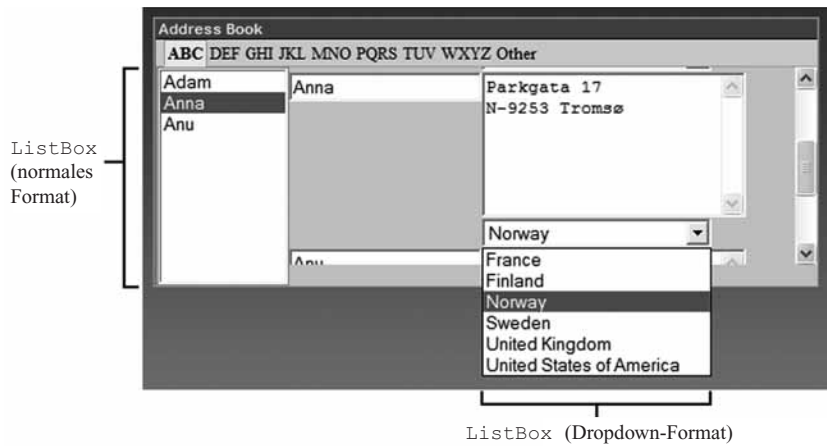
**Listing 4.12** Erstellen eines `Listbox`-Standard-Widgets und Hinzufügen eines `ChangeListener` zur Anwendung *Address Book*

```
addressLinks = new ListBox(); ❶ Erstellen des Standardlistenfelds

addressLinks.setVisibleItemCount(10); ❷ Festlegen der Anzahl sichtbarer Elemente

super.add(addressLinks);
addressLinks.addChangeListener(new ChangeListener(){ ❸ Hinzufügen eines
    ||| ChangeListener
    public void onChange(Widget sender) {
        int val = addressLinks.getSelectedIndex(); ❹ Gewählten Index holen
        String text = addressLinks.getItemText(val); ❺ Gewählten Text holen
        showAddress(text);
    }
});
```

Nach dem Erstellen des `Listbox`-Objekts ❶ geben Sie die maximale Anzahl sichtbarer Elemente an, die mit der Methode `setVisibleItemCount(number)` (❷) angezeigt werden. Im Grunde genommen gibt der Wert die Höhe des Listenfelds an. Für *Address Book* werden standardmäßig zehn Elemente angezeigt, danach wird das Widget mit einer Bildlaufleiste versehen.



**Abbildung 4.17** Die Dashboard-Anwendung *Address Book* verwendet beide Arten des `Listbox`-Widgets. Namen werden in einem gewöhnlichen Listenfeld gespeichert, Länder aus einem Listenfeld im Dropdownformat ausgewählt.

Listenfelder stellen eine Möglichkeit dar, dem Benutzer Optionen anzuzeigen. Um zu verstehen, welches Element der Benutzer auswählt, fügen Sie zur `Listbox` einen `ChangeListener` hinzu ❸. Ändert sich der gewählte Wert im Listenfeld, wird die Methode `onChange()` aufgerufen. Mit diesem `ChangeListener` zeigen Sie die Adresse der neu gewählten Person an, indem Sie zunächst – mithilfe der Methode `getSelectedIndex()` ❹ – den Index der vom Benutzer gewählten Option ermitteln und diesen Indexwert dann als Parameter an die Methode `getItemText()` übergeben, um die Textdarstellung der gewählten Option zu erhalten ❺. Dieser zurückgegebene Text wird dann als Parameter in der Methode `showAddress()` von *Address Book* verwendet.

Die alternative Ansicht von `Listbox` als Dropdownliste mit Optionen wird erstellt, indem die Anzahl sichtbarer Elemente auf 1 gesetzt wird. Um im Adressbuch Abwechslung zu schaffen, leiten Sie Klassen vom Widget `Listbox` ab, um die Auswahl des Landes als Dropdown-Widget zu ermöglichen (siehe Listing 4.13).

**Listing 4.13** Erstellen einer Dropdownliste für die Dashboard-Anwendung *Address Book*

```
private class CountryChoiceBox extends Listbox{
    public CountryChoiceBox(){
        super(); // Aufruf des übergeordneten Konstruktors
        this.setVisibleItemCount(1); // Widget als Dropdownliste festlegen
        this.addItem("France"); // Optionen hinzufügen
        this.addItem("Finland");
        this.addItem("Norway");
        this.addItem("Sweden");
        this.addItem("United Kingdom");
        this.addItem("United States of America");
    }
}
```

Sie können eine Option programmgesteuert mit der Methode `setSelectedIndex()` auswählen. Diese Methode funktioniert am besten – und ist auch sinnvoller –, wenn das Lis-

tenfeld in der Dropdownansicht gezeigt wird. Hiermit legen Sie beim Dashboard fest, dass für jede Adresse die korrekte Länderoption gezeigt wird.

Was in keiner Komponentenanwendung des Dashboards möglich ist: die Mehrfachauswahl in einem Listefeld. Dies wäre lediglich ein Fall für die Methode `setMultipleSelect()`. Wenn Sie dies tun, müssen Sie beim Abrufen des gewählten Elements ein bisschen gewitzter vorgehen, denn die Methode `getSelectedItem()` gibt nur das erste ausgewählte Element zurück. Um alle ausgewählten Elemente zu holen, müssen Sie über alle Elemente iterieren und dabei für jedes Element die Methode `isItemSelected()` aufrufen, um zu ermitteln, ob es ausgewählt ist oder nicht.

### Wie man sich durch eine Anwendung klickt

Die Verwendung eines Listefeldes bietet dem Benutzer viele Möglichkeiten. Wenn Sie bezüglich der Auswahlmöglichkeiten, die dem Benutzer zur Verfügung stehen, zurückhaltender sein wollen, sollten Sie die Verwendung eines der Widgets in Betracht ziehen, die die `ButtonBase`-Widgets erweitern.

Zu diesen Widgets gehören `Button` sowie `RadioButton` und `CheckBox`. `ButtonBase` stellt die Standardmethoden bereit, die allen drei untergeordneten Widgets gemein sind, also etwa das Festlegen bzw. Abrufen des mit ihnen verbundenen Texts über die Methoden `setText()` und `getText()` bzw. die entsprechenden Methoden für HTML (`setHTML()` und `getHTML()`).

Wir wollen uns diese drei untergeordneten Widgets und ihre Verwendung im Dashboard genauer ansehen.

### Auf Knopfdruck

Das `Button`-Widget haben Sie in diesem Kapitel bereits gesehen. Dabei handelt es sich um eine Schaltfläche in der Art, wie man sie von Webseiten her kennt. Zu ihrer Erstellung wird der Konstruktor wie folgt aufgerufen:

```
new Button("7");
```

Wie Sie in Abbildung 4.18 sehen können, sind die Schaltflächen auf dem Dashboard-Taschenrechner alle sorgfältig angeordnet. Fügen Sie sie einem `Grid`-Panel hinzu und ersetzen Sie mithilfe einer CSS-Formatierung die graue Schaltfläche und den schwarzen Text durch eine Kombination aus hellblauen Schaltflächen und grünem Text (diese Farbgebung ist der folgenden Abbildung natürlich nicht zu entnehmen).



**Abbildung 4.18**

Eine Anzahl von `Button`-Widgets bildet die Zifferntastatur der Dashboard-Anwendung *Calculator*

Gewöhnlich registrieren Sie einen `ClickListener` mit einem `Button`, um beim Anklicken der Schaltfläche bestimmte Aktionen durchzuführen. Bei Tasten, die in der Anwendung

*Calculator* (`org.gwtbook.client.ui.calculator.Calculator`) bestimmte Operationen darstellen – Addition, Subtraktion usw. –, fügen Sie jeweils `ClickListener` zu den Schaltflächen hinzu:

```
Button w = new Button("+");
w.addClickListener(new ClickListener() {
    public void onClick(Widget sender) {
        performCalculation(op);
    }
});
```

Wenn Sie bestimmte Tasten auf der Tastatur betätigen, während sich *Calculator* im Fokus befindet, sollten diese die gleiche Funktionalität aufweisen wie die entsprechenden Schaltflächen. Dies lässt sich durch programmgesteuertes Ausführen des Anklickens von Schaltflächen über den Aufruf der Methode `button.click()` erreichen.

`Button` ist eine von `ButtonBase` abgeleitete Klasse, es gibt aber noch zwei weitere. Als Nächstes behandeln wir `CheckBox`.

### Kontrollkästchen ansteuern

Das `CheckBox`-Widget implementiert die aus Browsern bekannte Kontrollkästchenfunktionalität (siehe Abbildung 4.19). Bei normalen Prä-Ajax-Anwendungen konnte man dieses Widget gewöhnlich als Bestandteil eines Formulars, dessen Werte bei Versenden des Formulars an den Server übergeben wurden. In der Ajax-Welt gilt diese Beschränkung nicht mehr: Zwar ist ein Kontrollkästchen nach wie vor in Formularen vorhanden, doch kann es auch als eigenständige Komponente fungieren. Verwenden Sie das Kontrollkästchen eigenständig, so können Sie einen `ClickListener` hinzufügen, um ein Codesegment ausführen zu können, sobald das Element aktiviert oder deaktiviert wird. In Kapitel 5 demonstrieren wir dies in der Anwendung *Dashboard*.



Abbildung 4.19 `CheckBox`-Komponenten im Einsatz

### Sie haben die Wahl

Das letzte von `ButtonBase` abgeleitete Widget, das in der GWT-Distribution enthalten ist, heißt `RadioButton`. Dieses auch *Optionsfeld* genannte Element stellt die Implementierung einer Gruppe sich gegenseitig ausschließender Auswahlmöglichkeiten dar. Wie bei `CheckBox` trifft man auf Optionsfelder sowohl in Formularen als auch in eigenständiger Form. Anders als das Kontrollkästchen benötigen Sie jedoch eine Möglichkeit, anzugeben, welche Optionsfelder zu einer bestimmten Gruppe gehören. Dies tun Sie in dem Konstruktor, von dem es drei Formen gibt, die jeweils die Definition eines Gruppennamens erfordern. Listing 4.14 zeigt die Konstruktion zweier Gruppen von Optionsfeldern.

Listing 4.14 Erstellung zweier Optionsfeldgruppen

```
RadioButton rb1 = new RadioButton("Grp1")
RadioButton rb2 = new RadioButton("Grp1", "Second Choice");
RadioButton rb3 = new RadioButton("Grp1", "<B>Third</B> Choice", true);
```

|| Gruppe 1  
|| erstellen

```

RadioButton rb10 = new RadioButton("Grp2", "Other");
RadioButton rb11 = new RadioButton("Grp2", "Other 2");
rb1.addClickListener(new ClickListener() {
    public void onClick(Widget sender) {
        performAction1(op);
    }
});
rb2.addClickListener(new ClickListener() {
    public void onClick(Widget sender) {
        performAction2(op);
    }
});

```

|| Gruppe 2 erstellen  
 || ClickListener zu den Optionsfeldern hinzufügen

Wenn die Verwaltung von Optionen nicht Zweck Ihrer Anwendung ist und Sie Ihren Benutzern mehr Freiheit dabei lassen wollen, sich zu äußern, dann interessieren Sie sich ganz bestimmt für die nächste Gruppe von Widgets – die `TextBoxBase`-Widgets.

#### 4.2.4 Benutzereingaben durch Texteingaben erhalten

Das Widget `TextBoxBase`, welches `FocusWidget` erweitert, ist wiederum kein Widget, sondern eine abstrakte Klasse, die die Standardfunktionalität für die Widgets `PasswordTextBox`, `TextArea` und `TextBox` bereitstellt. Die Klasse `TextBoxBase` stellt dabei die Funktionen bereit, die Sie von einem editierbaren Textelement auf dem Bildschirm erwarten würden: Verwerfen von Tastaturbetätigungen, Festlegen und Abfragen des sichtbaren Texts, Festlegen der Cursorposition und Erfassen eines Tastaturanschlags sowie nachfolgendes Ersetzen im Textfeld durch ein anderes Zeichen.

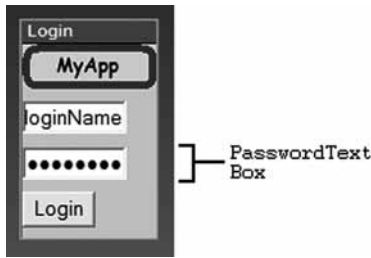
`TextBoxBase` bietet eine Reihe von Methoden, deren Implementierung man sinnvollerweise von einem Widget erwarten würde, das `TextBoxBase` erweitert. Sie können mit der Methode `getText()` den gesamten Text im Widget oder mit `getSelectedText()` den vom Benutzer selektierten Text abrufen. Ferner können Sie die Länge des ausgewählten Texts mithilfe von `getSelectionLength()` und die Cursorposition mit `getCursorPosition()` ermitteln. Außerdem ist es möglich, alle diese Eigenschaften mithilfe der entsprechenden Methodengegenstücke festzulegen, also etwa `setText()` und `setCursorPosition()`. Mit `selectAll()` können Sie zudem den gesamten Text selektieren.

Wir wollen uns die drei `TextBoxBase` untergeordneten Widgets und ihre Verwendung im Dashboard genauer ansehen.

##### Passworteingabe schützen

Das Widget `PasswordTextBox` stellt das Standardtextfeld in Browsern dar, in dem die Benutzereingabe maskiert wird. Meist wird ein solches Feld verwendet, um den Anwendungsbenutzern die Eingabe von Passwörtern zu gestatten. Das Dashboard enthält eine einfache Sicherheitsanwendung namens *Login* (`org.gwtbook.client.ui.login.Login`). Diese verwendet das `PasswordTextBox`-Widget wie in Abbildung 4.20 gezeigt.

Dieses Widget ist mit dem Konstruktor `PasswordTextBox()` schnell erstellt, enthält aber außer den von `TextBoxBase` geerbten keine weiteren Methoden.



**Abbildung 4.20**  
Das Widget `PasswordTextBox` verbirgt in der Dashboard-Anwendung *Login* die als Passwort eingegebenen Zeichen.

### Mehrere Textzeilen eingeben

Das Widget `TextArea` gestattet Anwendungsb Benutzern die Eingabe von mehrzeiligem Text. Im Dashboard verwenden Sie dieses Widget wie in *Abbildung 4.21* gezeigt im zusammengesetzten Widget `EditableLabel` (siehe Kapitel 7).



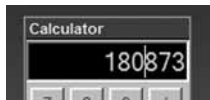
**Abbildung 4.21** Verwendung von `TextArea` im zusammengesetzten Widget `EditableLabel`, welches für das Dashboard erstellt wird (dieses werden wir uns in Kapitel 7 im Detail ansehen)

Das Erstellen einer `TextArea` erfordert lediglich den Einsatz des Konstruktors `TextArea()`. Hiermit können Sie die Breite und Höhe des Textfeldes wahlweise über CSS oder mit den dafür vorgesehenen Methoden `setCharacterWidth()` und `setVisibleLines()` festlegen.

Manchmal brauchen Sie aber nur eine einzige Textzeile. In diesem Fall ist das Widget `TextBox` besser geeignet.

### Eine Textzeile eingeben

Wenn Sie nicht mehrere Zeilen für editierbaren Text benötigen, dann ist `TextBox` wahrscheinlich das bessere Widget. Im Dashboard verwenden Sie Textfelder an vielen Stellen, so auch in der Anwendung *Calculator*. Hier kann ein Benutzer Zahlen direkt im Bildschirmbereich, aber auch über das Anklicken von Schaltflächen eingeben (siehe *Abbildung 4.22*).



**Abbildung 4.22**  
Widget `TextBox` in der Dashboard-Anwendung *Calculator*

Ein Aspekt der `TextBoxBase`-Widgets, den wir bislang nicht erwähnten, ist ihre Fähigkeit, mit einem `KeyListener` Tastaturanschläge abzufangen und entweder zu ändern oder sie zu verwerfen. Ein Taschenrechner sollte in seinem Textfeld keine anderen Eingaben akzeptieren als Ziffern. Dies ermöglicht der in *Listing 4.15* gezeigte Code

**Listing 4.15** Abweisen von Eingaben in der *Calculator*-Anzeige mithilfe eines `KeyListener`

```

theDisplay.addKeyListener(new KeyboardListenerAdapter() {
    public void onKeyPress(Widget sender,
                           char keyCode,
                           int modifiers) {
        if (!Character.isDigit(keyCode)) {
            ((TextBox)sender).cancelKey();
        }
    }
});

```

||| **KeyListener-Adapter hinzufügen**

||| **Angeschlagene Taste überprüfen  
Ungültige Eingaben verwerfen**

Zunächst verknüpfen Sie einen `KeyListenerAdapter` mit dem Textfeld (`theDisplay`) und überschreiben dann seine Methode `onKeyPress()`. In dieser Methode überprüfen Sie mit der Java-Methode `Character.isDigit()`, ob der `keyCode` eine Ziffer ist. Ist dies nicht der Fall, dann verwerfen Sie die Eingabe, d. h. sie erreicht das Widget nicht und wird deswegen auch keinesfalls im Browser angezeigt. Die hierfür verwendete Syntax mag Ihnen etwas befremdlich erscheinen: Sie müssen das `sender`-Objekt in ein `TextBox`-Objekt umwandeln, bevor Sie die Methode `cancelKey()` aufrufen, weil die Listener-Methode nur reine `Widget`-Klassenobjekte bearbeitet.

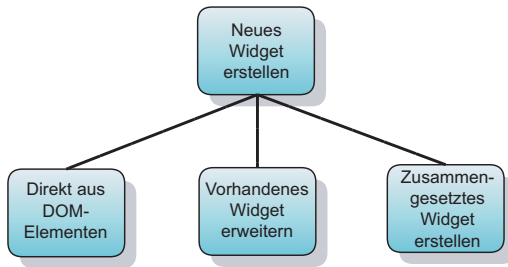
Sie können einige Formataspekte von `TextBox` ebenso via CSS oder programmgesteuert ändern, wie es mit `TextArea` möglich ist. Bei der Programmierung verwenden Sie `setMaxLength()` und `setVisibleLength()`. Die erste Methode definiert die maximale Anzahl von Zeichen, die in das Textfeld eingegeben werden können, die zweite die maximale Anzahl gleichzeitig sichtbarer Zeichen.

Diese Abhandlung hat Ihnen hoffentlich verdeutlicht, dass es eine Menge Widgets gibt, die Sie in Ihren Anwendungen einsetzen können, und konnte Ihnen zeigen, wo Sie sie bei den weiteren Entwicklungsarbeiten in diesem Buch verwenden werden. Allerdings reicht diese Anzahl einfacher Widgets manchmal nicht aus oder erfüllt die spezifischen Anforderungen Ihrer Anwendung nicht. In diesen Fällen können Sie entweder eigene Widgets erstellen oder vorhandene erweitern.

## 4.3 Neue Widgets erstellen

Zwar bietet GWT eine umfassende Menge an Widgets, aber trotzdem kann es vorkommen, dass genau die Funktionalität fehlt, die Sie für Ihre Anwendung benötigen. Unter Umständen sind die vorhandenen Widgets nicht geeignet, oder ein Basis-Widget fehlt schlichtweg. Abbildung 4.23 zeigt die drei Möglichkeiten, neue Widgets zu erstellen.

In diesem Abschnitt wenden wir uns den beiden ersten Ansätzen zu: dem Erstellen neuer Widgets direkt aus dem DOM und die Erweiterung vorhandener Widgets. Zur Erinnerung: Wenn hier von einem *Basis-Widget* die Rede ist, sprechen wir von einem Widget, das eine bestimmte Grundfunktionalität darstellt, die ein Browser bereitstellen kann. Alles andere fällt unter den dritten Ansatz: die Erstellung zusammengesetzter Widgets. In Kapitel 7 behandeln wir die zusammengesetzten Widgets detailliert. Sie werden verwendet, wenn Sie ein Widget erstellen müssen, das aus zwei oder mehr vorhandenen Widgets besteht.



**Abbildung 4.23**  
 Die drei Möglichkeiten zur Erstellung eines neuen Widgets in GWT. In diesem Abschnitt behandeln wir das Erstellen aus dem DOM und das Erweitern eines vorhandenen Widgets. Zusammengesetzte Widgets erörtern wir in Kapitel 7.

Für den ersten Ansatz haben wir lange nach einem Beispiel gesucht, da für unsere Zwecke die aktuelle GWT-Distribution umfassend genug ist. In den früheren Distributionen fehlte ein `FileUpload`-Widget, weshalb wir ein solches konstruierten. Wie wir dabei vorgehen, beschreiben wir in diesem Abschnitt.

Kommen wir nun zur Erweiterung vorhandener Widgets. Auf diese Weise werden Sie drei neue Widgets für das Dashboard erstellen. Das erste ist das Widget `PNImage`, welches das Standard-Widget `Image` erweitert, um Probleme im Internet Explorer 5.5 und 6 mit der Darstellung der Transparenz bei PNG-Bildern zu beheben. (Sie werden es für das Papierkorbsymbol verwenden, um dafür auch PNG-Bilder benutzen zu können).

Das zweite neue Widget erweitert das Widget `MenuItem`, um so das Dashboard-Widget `ToggleMenuItem` zu erstellen. GWT bietet ein `MenuItem`-Widget, das in Menüs verwendet wird, doch besteht dieses nur aus Text. `ToggleMenuItem` besteht hingegen aus Text und einem oder zwei Bildern, die den Auswahlzustand des Menüeintrags angeben (diese Bilder könnten etwa die Abbildung eines Häkchens und ein Leerbild sein, die angeben, dass die von dem Menüeintrag dargestellte Funktionalität ein- bzw. ausgeschaltet ist).

Nehmen wir also an, dass Sie ein neues Widget erstellen müssen. Beschreiben wir zunächst, wie Sie Ihr Ziel durch Manipulation des DOM erreichen.

### 4.3.1 Neue Widgets durch Manipulation des DOM erstellen

Der erste Ansatz zur Erstellung eines neuen Widgets, den wir behandeln wollen, betrifft die direkte Manipulation des DOM (so werden in der Regel auch die GWT-Standard-Widgets erstellt). Diese Aufgabe wird nicht allzu oft auf Sie zukommen, weil die meisten Widgets bereits vorhanden sind. Zudem sollten Sie es nur tun, wenn ein grundlegendes Browser-Widget fehlt.

Zum Zeitpunkt der Abfassung dieses Buchs konnten wir uns kein Widget vorstellen, das auf diese Weise erstellt werden müsste, doch wir erinnerten uns, dass ein wichtiges Widget fehlte, als wir begannen, GWT zu benutzen: `FileUpload`. Mittlerweile gibt es eine GWT-Standardversion. Trotzdem wollen wir darauf eingehen, wie wir unsere Variante erstellten.

#### Das Widget `FileUpload` entwickeln

Angenommen, Ihre Version von `FileUpload` soll genau so aussehen wie das Standardfeld für den Datei-Upload im Browser (Abbildung 4.24).



**Abbildung 4.24** GWT-Standard-Widget für den Datei-Upload im Browser (genau so soll das neue Widget aussehen)

Der erste Schritt beim Erstellen eines neuen Widgets besteht darin, zu entscheiden, welches DOM-Element das Widget implementieren wird. In diesem Fall fällt die Wahl nicht schwer: Weil Sie ein `FileUpload`-Widget erstellen wollen, benötigen Sie ein DOM-Element `<input>`, dessen Attribut `type` auf `file` festgelegt ist (dies stellt im Browser die einzige Möglichkeit dar, das in Abbildung 4.24 gezeigte Widget zu erstellen).

Im nächsten Schritt legen Sie fest, wo das neue Widget in der bereits vorhandenen Hierarchie angeordnet wird. Der Ansatz erfordert gelegentlich ein wenig Ausprobieren, weil die Hierarchie sehr umfangreich ist. Wenn man sich am gesunden Menschenverstand orientiert, sollte das Widget über Fokusereignisse und Klicks in Kenntnis gesetzt werden; insofern wird es unter der Klasse `FocusWidget` einsortiert.

Das Widget ist eher wie ein Textfeld, und ein Benutzer könnte erwarten, dass es sich auch so verhält, weswegen auch `TextBoxBase` ein geeigneter Kandidat zu sein scheint, doch würde dies eine Reihe von Problemen aufwerfen. Aufgrund der von den meisten Browsern implementierten Sicherheitsbeschränkungen ist es nicht möglich, den Text des zugrunde liegenden DOM-Eingabeelements festzulegen. Außerdem gilt das Konzept der Textausrichtung nicht für diesen Widget-Typ. Legen Sie für unser neues Widget also fest, dass es `FocusWidget` erweitert und die Schnittstelle `HasText` implementiert (wodurch angegeben wird, dass in diesem Widget Text vorhanden ist, den Sie festlegen und abfragen können).

Das Widget besteht aus dem in Listing 4.16 gezeigten Code.

**Listing 4.16** Das Widget `FileUpload`

```
package org.gwtbook.client;
import com.google.gwt.user.client.DOM;
import com.google.gwt.user.client.ui.FocusWidget;
import com.google.gwt.user.client.ui.HasText;

public class FileUpload extends FocusWidget implements HasText{

    public FileUpload(String name){
        super(DOM.createElement("input"));
        DOM.setAttribute(getElement(), "type", "file");
        DOM.setAttribute(getElement(), "name", name);
    }

    public String getText(){
        return DOM.getAttribute(getElement(), "value");
    }

    public void setText(String text){
        throw new RuntimeException("Not possible to set Text");
    }
}
```

❶ **FocusWidget erweitern**

❷ **Konstruktor implementieren**

❸ **Set- und Get-Methoden implementieren**

Zunächst geben Sie den Namen des neuen Widgets an und legen wie oben beschrieben fest, dass es die Klasse `FocusWidget` erweitert ❶. Von dort erbt das Widget eine ganze

Reihe von Methoden. Außerdem geben Sie an, dass das Widget die Schnittstelle `HasText` implementiert, d. h. Sie müssen die Methoden `setText()` und `getText()` angeben.

Nun definieren Sie den Konstruktor des Widgets ❷. Hier erfolgt die direkte DOM-Manipulation, die oben immer wieder erwähnt wurde. Anfangs müssen Sie das DOM-Eingabeelement erstellen. Hier kommt folgender Code zum Einsatz:

```
DOM.createElement("input")
```

Auf diese Weise wird das `<input>`-Element erstellt, das Sie nachfolgend im Konstruktor von `FocusWidget` verwenden, auf den Sie über den Aufruf von `super()` zugreifen, um das Widget ins Leben zu rufen. Gleichzeitig wird dieses neue `<input>`-Element so auch als Elementfeld des Widgets festgelegt.

Um das Widget abzuschließen, verwenden Sie weitere DOM-Manipulationsansätze, um den Typ des Elements (`file`) und seinen Namen zu bestimmen:

```
DOM.setAttribute(getElement(), "type", "file");
DOM.setAttribute(getElement(), "name", name);
```

Die Methode `getElement()` ruft das dem Widget zugrunde liegende Element ab, welches im Aufruf des Konstruktors von `FocusWidget` festgelegt wurde. Sie verwenden dieses zurückgegebene Element in den beiden Aufrufen von `setAttribute()`, um die Attribute `type` und `name` festzulegen.

Nachdem der Konstruktor das Element fertiggestellt hat, ist das folgende DOM-Element vorhanden, das als Java-Objekt behandelt wird:

```
<INPUT type="file" name="name" __eventBits="7040"/>
```

Was auffällt: `eventBits` wurde ungefragt eingestellt. Diese Werte erscheinen, weil Sie `FocusWidget` erweitern (wofür sie festgelegt sind).

Weil Sie die Schnittstelle `HasText` implementiert haben, müssen Sie die beiden Methoden bereitstellen, die diese Schnittstelle benötigt. Es beginnt mit der Methode `getText()`. Beim Widget `FileUpload` ist es sinnvoll, dass der Rückgabewert der Wert des Dateinamens ist, der vom Benutzer ausgewählt wurde. Um diesen Wert abzurufen, müssen Sie einige weitere DOM-Manipulationen vornehmen. Diesmal verwenden Sie die Methode `getAttribute()` wie folgt, um den Wert des Dateieingabefeldes abzurufen ❸:

```
return DOM.getAttribute(getElement(), "value");
```

Eine weitere Folge der Implementierung der Schnittstelle `HasText` besteht darin, dass Sie die Methode `setText()` angeben müssen ❹. Die Mehrzahl der Browser unterwirft den Text, der in ein Feld für den Dateiapload eingegeben wird, bestimmten Sicherheitsanforderungen. Insofern lösen Sie eine Exception aus, wenn diese Methode aufgerufen wird:

```
throw new RuntimeException("Not possible to set Text");
```

Nun können Sie das neue Widget wie jedes andere auch verwenden, indem Sie eine neue Instanz im Java-Code erstellen. Das kann etwa so aussehen:

```
FileUpload myUpload = new FileUpload();
```

Wie sich herausstellt, unterscheidet sich diese Version gar nicht so weit von jener, die nun als Bestandteil von GWT ausgeliefert wird, auch wenn Letztere ein wenig einfallsreicher und flexibler ist. Die zur Erstellung eines neuen Widgets durch DOM-Manipulation erforderlichen Schritte sind in Tabelle 4.3 zusammengefasst.

**Tabelle 4.3** Schritte zur Erstellung eines neuen Widgets durch DOM-Manipulation

Schritt	Aktion	Beschreibung
1	Ermitteln des DOM	Feststellen, welches oder welche DOM-Elemente das Widget kapseln wird.
2	Positionieren des Widgets in der Hierarchie	Feststellen, wo in der Hierarchie das neue Widget abzulegen ist: Soll es sich unmittelbar unter der Klasse <code>Widget</code> befinden, oder kann es sinnvoll sein, von einem Widget zu erben, das sich weiter unten in der Hierarchie befindet?
3	Ermitteln der zu implementierenden Schnittstellen	Ermitteln, welche Schnittstellen (zusätzlich zu den geerbten) das Widget implementieren muss. Wir werden Ereignis-schnittstellen in Kapitel 6 behandeln.
4	Implementieren des Konstruktors	Konstruktor mithilfe der entsprechenden DOM-Methoden erstellen, um das Element anzulegen und erforderliche Attribute zum DOM-Element hinzuzufügen; soll das Widget Ereignisse versenken, die nicht in der Hierarchie enthalten sind, so verwenden Sie die Methode <code>sinkEvents()</code> , um dies anzugeben. Vergessen Sie dann nicht, die Methode <code>onBrowserEvent()</code> im nächsten Schritt zu überschreiben.
5	Implementieren der erforderlichen Methoden	Sie müssen drei Methodentypen implementieren: <ul style="list-style-type: none"> <li>■ Von den Schnittstellen benötigte Methoden, die Sie in Schritt 2 zur Implementierung durch das Widget angegeben haben</li> <li>■ Die Methode <code>onBrowserEvent()</code> muss überschrieben werden, wenn Sie weitere Ereignisse versenken, die nicht von anderen, in der Hierarchie höherstehenden Widgets durchgeführt werden.</li> <li>■ Alle Methoden, die Sie erstellen wollen</li> </ul>

Mehr ist nicht erforderlich, um ein einfaches Widget von Grund auf neu zu erstellen. Doch wie wir bereits sagten: Es ist gar nicht so einfach, Situationen zu ersinnen, in denen dies erforderlich ist – insbesondere, weil die standardmäßig vorhandenen GWT-Widgets immer vollständiger werden. Meistens entstammen neue Widgets einer Erweiterung der vorhandenen oder aber der Erstellung zusammengesetzter Widgets (die wir in Kapitel 7 behandeln werden). Im folgenden Abschnitt erfahren Sie, wie man vorhandene Widgets erweitert. Sie werden drei konkrete Beispiele erstellen, die in der Folge auch im Dashboard zum Einsatz kommen.

### 4.3.2 Neue Widgets durch Erweiterung vorhandener Widgets erstellen

Ein Widget muss nicht stets von Grund auf neu erstellt werden. Es gibt auch Situationen, in denen sich ein neues Widget durch Erweiterung eines vorhandenen erstellen lässt, das beinahe die Funktionalität bietet, die Sie benötigen. In diesem Abschnitt sehen Sie dies anhand dreier Beispiele. Das erste erweitert das Widget `Image` so, dass es das Problem der Transparenzdarstellung in Internet Explorer 5.5 und 6 behebt, die Sie zur Anzeige des Papierkorbsymbols im Dashboard benötigen. Zweitens werden Sie ein Widget erstellen, das das Standard-Widget `MenuItem` dahingehend erweitert, dass Sie ein anderes Widget nach dem Menütext anzeigen können; dieses Widget erweitern Sie dann erneut und erstellen so ein drittes, das es gestattet, das hinter dem Text angezeigte Widget zwischen zwei verschiedenen Widgets umschalten zu lassen, wann immer das Menüelement angeklickt wird.

Die neuen Widgets werden anhand der in Tabelle 4.4 gezeigten Entwicklungsschritte erstellt. Diese ähneln erfreulicherweise stark den Schritten, die zur Entwicklung eines Widgets von Grund auf durchgeführt werden müssen; einzige Ausnahme: Sie müssen die DOM-Elemente nicht mehr angeben und festlegen, wo sich in der Hierarchie das Widget befindet, weil diese Eigenschaften dem Widget entstammen, das Sie erweitern wollen.

**Tabelle 4.4** Erweitern eines vorhandenen Widgets zur Erstellung eines neuen

Schritt	Aktion	Beschreibung
1	Festlegen der Funktionalität des Widgets	Exakt definieren, was Sie mit dem neuen Widget tun wollen.
2	Festlegen des zu erweiternden Widgets	Ermitteln, welches Widget funktionalitätsseitig dem Gewünschten am nächsten kommt. Manchmal finden Sie ein Widget, das in einem Bereich nicht genug, aber in einem anderen zu viel Funktionalität bietet. Dies ist einfach zu korrigieren, weil Sie Methoden überschreiben können, um mehr oder weniger Funktionalität zu bieten. (Manchmal besteht diese „Überfunktionalität“ im Bereich der Ereignisbehandlung; in diesem Fall können Sie die Methode <code>unsinkEvents()</code> verwenden und/oder die Methode <code>onBrowserEvent()</code> ändern.)
3	Ermitteln der zu implementierenden Schnittstellen	Müssen Sie neue Schnittstellen für das neue Widget implementieren? Wir werden Ereignisschnittstellen in Kapitel 6 behandeln.
4	Implementieren des Konstruktors	Erstellen Sie den Konstruktor mithilfe der passenden Aufrufe von <code>super()</code> , um den Konstruktor des übergeordneten Widgets aufzurufen, und implementieren Sie ggf. zusätzlichen Code, den das neue Widget benötigt. Soll das Widget Ereignisse versenken, die nicht im übergeordneten Widget enthalten sind, dann verwenden Sie die Methode <code>sinkEvents()</code> , um dies anzugeben. Wenn Sie möchten, dass Ihr neues Widget keine Ereignisse verwaltet,

Schritt	Aktion	Beschreibung
		die das übergeordnete Widget behandelt, verwenden Sie analog die Methode <code>unsinkEvents()</code> . (Vergessen Sie auch hier nicht, die Methode <code>onBrowserEvent()</code> im nächsten Schritt zu überschreiben.)
5	Implementieren der erforderlichen Methoden	<p>Sie müssen drei Methodentypen implementieren:</p> <ul style="list-style-type: none"> <li>■ Von den Schnittstellen benötigte Methoden, die Sie in Schritt 2 zur Implementierung durch das Widget angegeben haben. Alternativ überschreiben Sie jene Methoden, die von der Schnittstelle des übergeordneten Widgets benötigt werden, Sie aber anders implementieren wollen.</li> <li>■ Die Methode <code>onBrowserEvent()</code> muss überschrieben werden, wenn Sie weitere Ereignisse versenken wollen, die nicht vom übergeordneten Widget verwaltet werden, oder wenn Sie Ereignisse, die vom übergeordneten Widget verwaltet werden, nicht versenken wollen.</li> <li>■ Alle Methoden, die Sie erstellen wollen</li> </ul>

Erstellen wir also das erste der drei neuen Widgets, die Sie in der Anwendung *Dashboard* einsetzen werden: `PNImage`.

### Das Dashboard-Widget `PNImage` entwickeln

Das Dashboard enthält ein Papierkorbsymbol, das sich oben rechts auf dem Bildschirm befindet. In Kapitel 3 implementierten Sie es als einfaches `Image`-Widget. Allerdings ist das `Image`-Widget, das Bestandteil von GWT ist, nicht geeignet, ein transparentes PNG-Bild über einem anderen Bild darzustellen, wenn Sie Internet Explorer 5.5 oder 6 verwenden. Aus diesem Grund könnte das Papierkorbsymbol am Ende auf dem Hintergrundbild zu liegen kommen. Außerdem werden Sie, wenn wir in Kapitel 7 einen Schieberegler erstellen, ein Miniaturbild über ein Hintergrundbild schieben. Wenn Sie das `Image`-Standard-Widget verwenden und der Programmierer Ihnen dann zwei PNG-Dateien zukommen lässt, sieht das Ganze fürchterlich aus.

Das Problem besteht darin, dass das transparente PNG-Bild in Internet Explorer 5.5 einen Haloeffekt (Abbildung 4.25) verursacht und im Internet Explorer 6 einen ganz scheußlichen Hintergrund erzeugt (Abbildung 4.26).



**Abbildung 4.25**

Problem mit der Transparenz von PNG-Bildern in Internet Explorer 5.5: Weißer Halo um den Stern beim Einsatz des GWT-Widgets `Image`



**Abbildung 4.26**

Problem mit der Transparenz von PNG-Bildern in Internet Explorer 6: Grauer Hintergrund um den Stern beim Einsatz des GWT-Widgets `Image`

Die Drittanbieterbibliothek GWT Widget Library enthält ein Widget namens `PNGImage`, das dieses Problem beseitigt. Im vorliegenden Abschnitt werden wir die einzelnen Schritte beschreiben, die wir zur Entwicklung dieses Widgets durchgeführt haben. (Um zu vermeiden, dass Sie die GWT Widget Library bereits jetzt in Ihr Projekt importieren müssen, erstellen Sie das Widget `PNGImage` im Package `org.gwtbook.client.ui`.)

Statt nun ein vollkommen neues Widget zu erstellen, haben wir beschlossen, dass `PNGImage` sich weitestgehend wie das Standard-Widget `Image` verhalten soll – eine Vererbung von diesem Standard-Widget war also der Schlüssel zum Ziel. Das Widget besteht aus drei Klassen: einer Widget-Hauptklasse und zwei Implementierungsklassen (je eine für den Internet Explorer und eine für andere Browser). Listing 4.17 zeigt den vollständigen Code der Implementierungsklasse von `PNGImage`.

**Listing 4.17** Hauptklasse von `PNGImage`

```
package org.gwtbook.client.ui;

import org.gwtbook.client.ui.impl.PNGImageImpl;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.Event;
import com.google.gwt.user.client.ui.Image;

public class PNGImage extends Image ❶ Image-Klasse erweitern
{
    private PNGImageImpl impl;

    public PNGImage (String url, int width, int height){
        impl = (PNGImageImpl) GWT.create(PNGImageImpl.class); ❷ Deferred Binding der Implementierungsklasse

        setElement(impl.createElement(url, width, height)); ❸ Festlegen des DOM-Elements des Widgets

        sinkEvents(Event.ONCLICK | Event.MOUSEEVENTS ❹ Ereignisse versenken
                  Event.ONLOAD | Event.ONERROR);
    }

    public String getUrl (){ ❺ Methode getUrl() bereitstellen
        return impl.getUrl(this);
    }

    public void setUrl (String url){ ❻ Methode setURL() deaktivieren
        throw new RuntimeException("Not allowed for a PNG image");
    }
}
```

Nachdem Sie festgestellt haben, dass die für das Widget `PNGImage` erforderliche Funktionalität abgesehen von der Behandlung der PNG-Transparenz exakt mit der des Widgets `Image` übereinstimmt, geben Sie an, dass die Definition der Klasse `PNGImage` das Widget `Image` erweitert ❶. Es gibt keine neuen Schnittstellen, die `PNGImage` implementieren müsste, weil sich `PNGImage` ja so verhalten soll wie `Image`.

Bei ❷ verwenden Sie Code so ähnlich wie bei der Internationalisierung beschrieben. Allerdings stellen Sie in diesem Fall nicht die Auswahl der korrekten Java-Klasse zurück, bis Sie das Gebietsschema kennen, sondern verschieben die Bindung, bis Sie wissen, in welchem Browser der Code ausgeführt wird (der fachsprachliche Begriff dafür lautet *Deferred Binding*; in Kapitel 15 gehen wir näher darauf ein). An dieser Stelle wird dieser

Kniff verwendet, weil Sie abhängig davon, ob der verwendete Browser Internet Explorer 5.5 oder 6 ist oder nicht, anderen Code implementieren müssen. Wird ein anderer Browser als Internet Explorer 5.5/6 verwendet, dann wählt das Deferred Binding eine Klasse namens `PNGImageImpl` aus, ansonsten die Klasse `PNGImageImplIE6`.

Nun erstellen Sie das DOM-Objekt **3**; es stellt das PNG-Bild dar. Dies tun Sie durch Aufrufen der Erstellungsmethode für die vom Compiler im Rahmen des Deferred Binding gewählte Version der Klasse `PNGImageImpl`. Ist der Browser nicht Internet Explorer 5.5/6, dann enthält die Erstellungsmethode den folgenden Code zur Erstellung eines DOM-Standardelements `<img>` und zur Festlegung einiger Attribute:

```
Element result = DOM.createImg();
DOM.setAttribute(result, "src", url);
DOM.setIntAttribute(result, "width", width);
DOM.setIntAttribute(result, "height", height);
return result;
```

Handelt es sich dagegen um den Internet Explorer in der Version 5.5 oder 6, dann wird die Erstellungsmethode in der Klasse `PNGImageImplIE6` verwendet. Zunächst wird bestimmt, ob das Bild tatsächlich eine PNG-Datei ist; hierzu wird ganz profan die Erweiterung überprüft. Wenn die Datei kein PNG-Bild ist, dann wird der normale Konstruktor zur Erstellung eines `<img>`-Elements benutzt. Ansonsten müssen Sie einen Internet Explorer-spezifischen Filter namens `AlphaImageLoader` auf das Bild anwenden. Hierzu erstellen Sie ein `<div>`-Element und konfigurieren dessen inneren HTML-Code so, dass er `PNGImage` sowie den Filter `AlphaImageLoader` enthält. Das sieht dann so aus:

```
Element div = DOM.createDiv();
DOM.setInnerHTML(div, "<span style=\"display:inline-block;width:\" +
    width + \"px;height:\" + height + \"px;\"+
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(src=\"" +
    url + "\", sizingMethod='scale')\"></span>");
```

In allen Fällen wird ein DOM-Element an den Konstruktor in der Klasse `PNGImage` zurückgegeben. Dieses Element wird dann als das Element des Widgets `PNGImage` festgelegt. Sie finden den vollständigen Code für die Implementierungsklassen auf der Download-website zu diesem Buch (<http://www.manning.com/hanson>).

Die Ereignisse, die dieses Widget versenken muss, sind dieselben wie beim Widget `Image`, weswegen Sie eigentlich keinen neuen Aufruf von `sinkEvents()` einbinden müssen. In diesem Fall jedoch rufen Sie nicht den von `Image` verwendeten Konstruktor mit `super()` auf, d. h. Sie müssen die zu versenkenden Ereignisse explizit angeben **4**. Die Methode `onBrowserEvent()` müssen Sie nicht überschreiben, weil diese direkt vom übergeordneten Widget `Image` geerbt wird.

Um das Widget fertigzustellen, müssen Sie sicherstellen, dass die Methoden der Klasse `Image`, die Sie erben, in geeigneter Weise behandelt werden. Die Methode `getURL()` **5** der Klasse `Image` ist bei dieser Implementierung nicht mehr gültig, weswegen Sie sie überschreiben und dafür sorgen, dass die browserspezifische Implementierung aufgerufen wird, die Sie zur Rückgabe des korrekten Wertes verwenden.

Schließlich ist es bei PNG-Bildern nicht möglich, den URL festzulegen, da dieser für ein `Image`-Widget vorgesehen ist. Dies ist ein Fall, in dem das übergeordnete Widget mehr

Funktionalität bietet, als Sie im erweiterten Widget unterstützen können. Sie lösen dieses Problem, indem Sie eine `RuntimeException` auslösen, wenn jemand versucht, diese Methode aufzurufen ⑥.

Das korrigierte `PNGImage` wird nun in Internet Explorer 5.5 und 6 korrekt angezeigt (vgl. Abbildung 4.27): Der Haloeffekt aus Internet Explorer 5.5 und der Hintergrund aus Internet Explorer 6 sind fort, und `PNGImage` zeigt das PNG-Bild wie gewünscht in allen Browsern an.



**Abbildung 4.27**

Korrigierte PNG-Behandlung ohne Halo oder Hintergrund im Internet Explorer mit dem neuen, das GWT-Widget `Image` erweiternden Widget `PNGImage`

Sie erstellen das neue `PNGImage`-Objekt mithilfe des Java-Standardaufrufs des Konstruktors:

```
PNGImage myImage = new PNGImage("star.png", 100, 100);
```

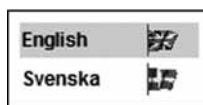
Jetzt können Sie mit der Klasse `PNGImage` das Papierkorbsymbol im Dashboard darstellen und auch bedenkenlos Schieberegler erstellen – zwei Aufgaben, denen wir uns im weiteren Verlauf dieses Buchs eingehender widmen wollen. Als Nächstes erörtern wir die Entwicklung eines weiteren Widgets, das Sie für das Dashboard benötigen: `ToggleMenuItem`.

## 4.4 Das Dashboard-Widget `ToggleMenuItem` entwickeln

---

Wie weiter oben bereits erwähnt, bietet GWT ein nützliches Widget namens `MenuItem`. Ein oder mehrere dieser Menüelemente werden in einer Menüleiste positioniert – auf diese Weise entsteht das GWT-Menüsystem. Wiewohl ausgesprochen praktisch, ist bei `MenuItem` nicht auf den ersten Blick klar, wie man Menüelemente mit zwei Komponenten anzeigen soll, also etwa das Menüelement und eine Tastenkombination oder ein zugehöriges Bild.

In diesem Abschnitt erstellen Sie zunächst ein Widget `TwoComponentMenuItem`, das es Ihnen gestattet, Text gefolgt von einem anderen Widget zu platzieren (dies könnte ein `Label`, das eine Tastenkombination darstellt, oder ein `Image` sein). Der Inhalt eines `MenuItem` ist zwar in der Regel Text, doch können Sie diesen Text auch als HTML-Code interpretieren lassen – an genau dieser Stelle hakt unser neues Widget ein. Im Dashboard werden Sie mit diesem Widget die Gebietsschemata anzeigen (Abbildung 4.28).

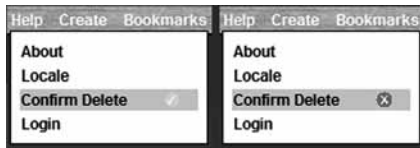


**Abbildung 4.28**

Beispiel für die Verwendung von `TwoComponentMenuItem` im Dashboard. Angezeigt wird eine animierte GIF-Datei einer Landesflagge, die auf das betreffende Gebietsschema verweist.

Wenn Sie ein Widget erstellt haben, mit dem Sie zwei Komponenten in einem Menüelement ablegen können, lässt sich dieses auch erweitern, um am Ende das Widget zu er-

halten, welches auf den Text folgend, abhängig von einem internen Status, eines von zwei Bildern anzeigt. Wird das Widget angeklickt, dann ändern sich, wie Abbildung 4.29 zeigt, Status und Bild. Dieses Widget heißt dann `ToggleMenuItem`.



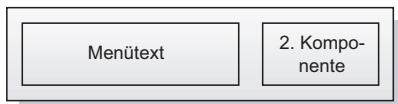
**Abbildung 4.29**

Verschiedene Zustände von `ToggleMenuItem` in der Menüleiste der Dashboard-Anwendung *Help* (die Funktionalität `CONFIRM DELETE` wird wahlweise als aktiviert oder deaktiviert angezeigt)

Beim Dashboard verwenden Sie dieses Widget, um dem Benutzer anzuzeigen, ob er beim Löschen einer Komponente zur Bestätigung aufgefordert wird oder nicht. In Abbildung 4.29 sehen Sie auf der linken Seite, dass ggf. eine Bestätigungsmeldung angezeigt wird. Klickt der Benutzer nun auf dieses Menüelement, dann wird die rechte Seite der Abbildung sichtbar. Wir wollen zunächst `TwoComponentMenuItem` erstellen und uns dann mit `ToggleMenuItem` befassen.

#### 4.4.1 `TwoComponentMenuItem` erstellen

`TwoComponentMenuItem` ist ein Widget, das den Text in einem normalen Menüelement gefolgt von einem zweiten Widget zeigt (Abbildung 4.30). Dieses zweite Widget könnte beispielsweise Text sein, der eine zugehörige Tastenkombination (z. B. „Strg+O“) zum Ausführen des zugehörigen Befehls oder aber ein Bild anzeigt.



**Abbildung 4.30**

Schema von `TwoComponentMenuItem`

Sie implementieren `TwoComponentMenuItem` wie in Listing 4.18 gezeigt.

**Listing 4.18** Die Klasse `TwoComponentMenuItem`

```
package org.gwtbook.client;

import com.google.gwt.user.client.Command;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.MenuItem;
import com.google.gwt.user.client.ui.Widget;

public class TwoComponentMenuItem extends MenuItem{
    // ❶ Widget MenuItem erweitern

    protected HorizontalPanel theMenuItem = new HorizontalPanel();
    // ❷ Horizontal-Panel so erweitern, dass es die Komponenten aufnimmt

    public TwoComponentMenuItem(String theText,
                                Widget secondComponent,
                                Command theCommand){
```

```

super(theText, true, theCommand); ❸ Konstruktor des übergeordneten Widgets aufrufen
theMenuItem.add(new Label(theText+ " ")); ❹ Komponenten des neuen Widgets
theMenuItem.add(secondComponent); festlegen
this.setSecondComponent(secondComponent); ❺ Rechte Komponente als Widget festlegen
}

public void setSecondComponent(Widget newComponent){ ❻ Methode setSecond-
Component() definieren

    theMenuItem.remove(1);
    theMenuItem.add(newComponent);
    SimplePanel dummyContainer = new SimplePanel();
    dummyContainer.add(theMenuItem);
    String test = DOM.getInnerHTML(dummyContainer.getElement());
    this.setHTML(test);
}

public void setFirstComponent(String newComponent){ ❼ Methode
setFirstComponent()

    theMenuItem.remove(0);
    theMenuItem.insert(new Label(newComponent), 0);
    SimplePanel dummyContainer = new SimplePanel();
    dummyContainer.add(theMenuItem);
    String test = DOM.getInnerHTML(dummyContainer.getElement());
    this.setHTML(test);
}

public void setText(String theText){ ❸ Standardmethode setText()
    setFirstElement(theText);
}
}

```

TwoComponentMenuItem erweitert die Klasse MenuItem ❶ und damit auch alles, was diese Klasse zur Verfügung stellt – Sie können die Klasse also überall dort verwenden, wo Sie auch MenuItem einsetzen können. Um die Behandlung zweier Komponenten zu ermöglichen, erstellen Sie ein HorizontalPanel ❷, welches die beiden Komponenten aufnehmen wird. Das erste Element ist ein Label, das den Text enthält, das zweite das an die Klasse übergebene Widget.

Im Konstruktor des Widgets rufen Sie zunächst den Konstruktor von MenuItem auf ❸, um sicherzustellen, dass ein gültiges MenuItem erstellt wird, geben aber gleichzeitig an, dass Sie die Textkomponente als HTML-Code behandeln werden. Danach erstellen Sie unter Verwendung des für das Menü übergebenen Texts ein Label ❹ und fügen es dem HorizontalPanel hinzu, desgleichen die zweite Komponente direkt zum selben HorizontalPanel. Danach rufen Sie die Methode setSecondComponent() auf (❺). Beachten Sie, dass Sie die zweite Komponente hinzufügen müssen, bevor Sie die Methode aufrufen, um sie festzulegen, da die Methode andernfalls unterbrochen würde, denn sie entfernt – wie wir gleich sehen werden – zunächst die zweite Komponente.

Um die Darstellung der beiden Komponenten auf dem Bildschirm zu erstellen, treiben Sie nun allerlei Mystisches mit GWT und dem DOM ❸. Zunächst entfernen Sie eine ggf. vorhandene zweite Komponente aus dem HorizontalPanel und fügen dann die neue zweite Komponente hinzu. Das ist zunächst einfachstes GWT. Danach wollen Sie auf den HTML-Code von HorizontalPanel zugreifen; dies tun Sie durch Hantieren mit dem DOM. Sie fügen das HorizontalPanel in ein SimplePanel ein und rufen dann den inneren HTML-Code des SimplePanel ab. Ein ähnlicher Vorgang dient der Festlegung des ersten Elements in ❼; beachten Sie jedoch, dass der Code nun die erste Komponente (Index 0) ent-

fernt, wodurch die zweite Komponente zur ersten wird. Auf diese Weise wird impliziert, dass Sie eine neue erste Komponente vor der vorhandenen ersten Komponente einfügen müssen – Sie müssen also die Methode `insert()` (statt der Methode `add()`) verwenden.

Die Fertigstellung der Komponente besteht im Überschreiben der Methode `setText()`, weil Sie erforderlichen Text nicht direkt im Widget, sondern im `HorizontalPanel` ablegen müssen. Um den Text zu ändern, legen Sie wie in der Überschreibmethode ❸ gezeigt ein neues erstes Element fest.

Wie bei `pngImage` und `fileUpload` erstellen Sie auch hier das neue Widget über seinen Konstruktor:

```
TwoComponentMenuItem myMenuItem =
    new TwoComponentMenuItem("MenuItem",
        new Image("myMenuItemImage.jpg"),
        new Command(){
            public void execute(){
                Window.alert("Menu Clicked");
            }
        }
    );
```

Nachdem die Erstellung von `TwoComponentMenuItem` nun abgeschlossen ist, können wir daran gehen, es durch Erweiterung der Klasse zu spezialisieren: Wir erstellen das Dashboard-Widget `ToggleMenuItem`.

#### 4.4.2 ToggleMenuItem erstellen

Nachdem Sie das Widget `TwoComponentMenuItem` erstellt haben, können Sie es wie in Listing 4.19 gezeigt erweitern, um das Widget `ToggleMenuItem` zu entwickeln. Dieses Widget zeigt abhängig von einem internen Status eines von zwei verschiedenen Bildern. Beim Dashboard wollen Sie ein Häkchen im Menü anzeigen, wenn die betreffende Funktionalität aktiviert ist, andernfalls ein Kreuz. In anderen Szenarios sollte es ebenso einfach sein, abwechselnd ein Häkchen oder *kein* Bild (für die deaktivierte Funktionalität) anzuzeigen.

**Listing 4.19** Die Klasse `ToggleMenuItem`

```
package org.gwtbook.client;

import com.google.gwt.user.client.Command;
import com.google.gwt.user.client.ui.Image;

public class ToggleMenuItem extends TwoComponentMenuItem{ // ❶ TwoComponentMenuItem erweitern

    public class ToggleMenuItemStates{ // ❷ Widget-Status definieren
        static public final boolean ON = true;
        static public final boolean OFF = false;
    }

    Widget[] states; // ❸ Array zur Aufnahme der Umschalt-Widgets definieren
    boolean state = true; // ❹ Boolesche Variable zur Aufnahme des aktuellen Status definieren

    public ToggleMenuItem(String theText,
        Widget onState,
        Widget offState,
```

```

        Command command){
    super(theText, onState, command); ❸ Übergeordneten Konstruktor aufrufen
    states = new Widget[2];           ❹ Statusvariablen einrichten
    states[0] = onState;
    states[1] = offState;
}

public void toggle(){                ❺ Methode zum Umschalten der Widgets einrichten
    setSecondComponent(states[state?1:0]);
    state = !state;
}

public boolean getState(){           ❻ Aktuellen Status des Widgets holen
    return state;
}
}

```

Für `ToggleMenuItem` erweitern Sie `TwoComponentMenuItem` – das geht aus der Definition der Klasse hervor ❶. Durch die Vererbungshierarchie erbt dieses Widget `MenuItem` und kann überall dort eingesetzt werden, wo dies auch mit einem `MenuItem` möglich wäre.

Die zweite Komponente dieses Widgets ist eines von zwei verschiedenen Widgets (hierzu kommt im Dashboard das Widget `Image` zum Einsatz). Diese beiden Widgets werden im `widget`-Array abgelegt, das Sie bei ❹ erstellen. Um zu bestimmen, welches Widget standardmäßig angezeigt wird, setzen Sie den Statuswert dieses `ToggleMenuItem`-Widgets auf `true` ❺.

Das Erstellen des Widgets erfolgt durch den Konstruktor, der zunächst den übergeordneten Konstruktor aufruft, um sicherzustellen, dass dieses `ToggleMenuItem` sich ähnlich verhalten wird wie ein `TwoComponentMenuItem` ❸, und dann die zweite Komponente als Bild festlegt, welches den ersten Status repräsentiert. Danach erstellt er ein Array und speichert die beiden als Parameter übergebenen Widgets als Widgets für die beiden alternativen Zustände ❹.

Wir haben bereits ein wenig über die Zustände in diesem Widget gesagt: Bei ❺ definieren Sie eine innere Klasse, die die vorhandenen Zustände (`ON` und `OFF`) enthält. Das Umschalten des Zustands des Menüelements erfolgt mithilfe der Methode `toggle()` ❻, die aus dem Code des Benutzers heraus aufgerufen werden muss. Diese Methode legt die zweite Komponente des `ToggleMenuItem`-Widgets fest, die den Status darstellt, der gegenwärtig *nicht* aktiv ist, und ändert dann die interne Darstellung des Status. Den aktuellen Status des Widgets geben Sie durch Aufrufen der Methode `getState()` zurück, die bei ❸ definiert ist.

Sie könnten die Hierarchie der Menüelemente nun fortführen und viele verschiedene Typen erstellen und so all die Menüelemente kreieren, die Ihnen aus Desktopanwendungen bekannt sind. Wir wollen es aber bei diesen belassen. Nach der Entwicklung dieser Klasse verfügen Sie nun über alle grundlegenden Widgets, die wir benötigen, um die Beispielanwendung *Dashboard* zu erstellen. Allerdings müssen wir uns Panels und Ereignisse noch genauer ansehen.

---

## 4.5 Zusammenfassung

---

So endet der erste Teil unserer Reise durch die Welt der GWT-Grundlagen, der die Widgets behandelte. Sie haben gesehen, dass GWT bereits eine beachtliche Menge an Standard-Widgets bietet. Wo die erforderliche Funktionalität nicht vorhanden ist, können Sie sie durch Erstellung eigener Widgets relativ einfach nachbilden, und zwar, indem Sie entweder ein von Grund auf neues Widget erstellen oder ein vorhandenes Widget erweitern. (Wenn Sie eine komplexere Funktionalität benötigen, die besser implementiert sein muss als nur durch Verknüpfung von zwei oder mehr einfachen Widgets, sollten Sie zusammengesetzte Widgets, die wir in Kapitel 7 behandeln, in Betracht ziehen.)

Die Tatsache, dass Widgets zwei alternative Ansichten bieten – die Java-Objektdarstellung und die DOM-Darstellung –, kann anfangs ein wenig verwirrend sein. Allerdings haben Sie zu 99 Prozent ohnehin nur mit dem Java-Objekt zu tun. Einer der wenigen Fälle, in denen Sie an das Widget in der DOM-Ansicht denken sollten, liegt bei der Erstellung neuer Widgets vor. Es kann sogar gefährlich sein, sich zu sehr auf die DOM-Darstellung des Widgets einzulassen, da nicht gewährleistet ist, dass künftige GWT-Versionen dasselbe DOM-Konstrukt für ein bestimmtes Widget verwenden werden.

In der zweiten Hälfte des Kapitels haben Sie drei Widgets erstellt, die in der laufenden Anwendung *Dashboard* eingesetzt werden: `PNGImage` sowie `TwoComponentMenuItem` und `ToggleMenuItem`. Um beim Dashboard zu bleiben, sehen wir uns als Nächstes an, wie man Widgets in der Anwendung optisch anordnet. Dies geschieht mit den in Kapitel 5 beschriebenen Panels.