1 Einleitung

1.1 Was ist Vaadin?

Vaadin ist ein auf dem Google Web Toolkit (siehe [GWT]) basierendes Open-Source-Framework, das die einfache Erstellung auch komplexer browserbasierter Applikationen ermöglicht.

> Tatsächlich bringt sich Vaadin auch sehr stark in das GWT-Projekt ein: Es stellt ein Mitglied des Steuerungsausschusses und organisiert auch die Konferenz gwt.create – sieht sich also nicht als Konkurrenz, sondern in positiver Koexistenz mit GWT.

Vaadin abstrahiert hierbei für die normale Programmierung von HTML, JavaScript, AJAX und browserspezifischen Details und bietet ein javabasiertes Programmiermodell mit Widgets und Events an, das sehr stark an klassische UI-Programmierung in Java angelehnt ist.

Programmierung vollständig in Java

Tatsächlich müssen wir uns in der normalen Programmierung mit Vaadin keine Gedanken über verschiedene Datenmodelle auf Client und Server machen und haben keine Schwierigkeiten mit der Aktualisierung der jeweiligen Daten. Wir können unsere verschiedenen Widgets und ihr Layout mit Java programmieren und kommen mit HTML und JavaScript nicht in Berührung.

Steuerung des Layouts über CSS

Zusätzlich können die Details des Layouts getrennt von der Programmierung beeinflusst und über CSS applikationsweit konsistent gesteuert werden, so dass bei Bedarf auch Look & Feel der gesamten Applikation über *Themes* ausgetauscht werden kann.

Über einen Erweiterungsmechanismus besteht die Möglichkeit, Widgets von Drittanbietern einfach zu integrieren und zu programmieren. Das *Vaadin Directory* (siehe [Vaadin]) bietet aktuell mehr als 400 Komponenten und Widgets an, die die Funktionalität von Vaadin erweitern.

Hinter den Kulissen implementiert Vaadin neben dem Framework auf der Serverseite, das wir in der normalen Programmierung benutzen, eine *Client-Side Engine*, die im Browser läuft und sowohl die Darstellung der Benutzerschnittstelle als auch die Übermittlung der Benutzerinteraktionen an den Server übernimmt und sich dabei auch um browserspezifische Anpassungen kümmert. Diese Client-Side Engine ist in Java geschrieben und wird mit dem GWT-Compiler in JavaScript übersetzt.

Während wir bei der normalen Programmierung nicht viel mit der Client-Side Engine zu tun haben, wird diese interessant, wenn wir eigene Widgets für die Erweiterung von Vaadin programmieren. Dies ist allerdings ein Thema für Fortgeschrittene und wird in diesem Buch nicht weiter betrachtet.

1.2 Historie

Die Entwicklung an Vaadin wurde bereits im Jahr 2000 gestartet, damals als Adapter für das Framework Millstone der Firma *IT Mill Ltd.,* und die erste Version wurde 2002 veröffentlicht. Die Funktion umfasste damals Kommunikation mittels Ajax und eine eigene, proprietäre Render-Engine für die Darstellung von Widgets.

Ab 2006 wurde das Framework eigenständig kommerziell weiterentwickelt und 2007 umbenannt zu *IT Mill Toolkit*. Da sich im Lichte des Open-Source-Frameworks GWT die Weiterentwicklung einer eigenen Render-Engine nicht lohnt, wurde der Wechsel zu GWT als Basis für das Rendering gestartet. Ende 2007 wurde die Lizenz in eine Open-Source-Lizenz geändert.

2008 investierte einer der Gründer von MySQL (Monty Widenius) einen Teil seines Vermögens in die Firma und schuf damit die Grundlage für die Entwicklung der ersten Version des IT Mill Toolkit 5, die nach langer Betatestperiode 2009 erschien. Kurz nach Erscheinen wurde das Framework umbenannt in Vaadin (der finnische Name für ein weibliches Rentier) und eine Vorabversion der Version 6 veröffentlicht. Zusätzlich wurde auch IT Mill Ltd. umbenannt zu Vaadin Ltd., um klarzumachen, wie sehr die Firma hinter ihrem Framework steht.

Es dauerte bis März 2013, bis die Version 7 herauskam. Ende Juni 2013 kam die Version 7.1 heraus, die abgesehen von Fehlerkorrekturen als wichtigste Funktionalität *Server Push* enthielt. Hiermit wird der Server in die Lage versetzt, eigenständig Informationen auf dem Client zu aktualisieren, was das Programmiermodell noch flexibler macht.

1.3 Wofür ist Vaadin gut?

Vaadin ist ein typischer Vertreter eines *Rich Internet Application Frameworks* (*RIA* Framework), mit dessen Hilfe Webapplikationen implementiert werden, die so weit wie möglich einer klassischen Desktop-Applikation entsprechen. Hierbei zeigt sich als Trend, dass mehr und mehr weggegangen wird von proprietären Lösungen wie zum Beispiel Adobe Flash hin zu HTML5 und JavaScript. GWT und damit auch Vaadin setzen auf diesem Trend auf.

Mit immer größerer Leistungsfähigkeit der unterliegenden Browser und eines Frameworks wie Vaadin wird die Distanz zwischen klassischen Applikationen und Applikationen, die im Browser laufen, immer geringer, und Analysten sagen bereits seit 2007 voraus, dass die Grenzen zwischen diesen Applikationsarten immer mehr verschwimmen werden (siehe [Forrester 2007]).

Damit stellt sich weniger die Frage, was wir mit Vaadin machen können, als was sich nicht machen lässt. Die folgenden beiden Punkte sind problematisch:

- Verarbeitung großer Datenmengen auf dem Client
- vollständige Funktionalität ohne Netzzugang (Offline-Funktion für mobile Geräte)

Im ersten Fall stellt sich die Frage, warum die Datenverarbeitung auf dem Client durchgeführt werden muss und ob es nicht sinnvoller wäre, die Daten zum Server oder in eine Cloud-Infrastruktur zu bewegen.

Im zweiten Fall haben wir eine Situation, die das Programmiermodell von Vaadin ad absurdum führt. Wenn es hingegen um zeitlich beschränkte Offline-Funktionalität geht, dann gibt es zwei Aspekte: zum einen die Berechnung von Informationen auf dem Server. Dies kann man in den Griff bekommen, indem die entsprechenden Berechnungen lokal durchgeführt werden oder eine Meldung angezeigt wird, dass der Service aktuell nicht zur Verfügung steht. Der andere Aspekt ist die Eingabe von Daten. Hier besteht die Möglichkeit, die Daten lokal zu cachen und zum Server zu übertragen, sobald dieser wieder verfügbar ist.

Die Vaadin-Erweiterung TouchKit für mobile Geräte bietet Unterstützung für diese Vorgehensweisen. In jedem Fall ist es aber sinnvoll, den Nutzer auf die fehlende Verbindung hinzuweisen; das Touchkit ermöglicht, hierfür ein eigenes Theme zu verwenden, um dies sehr deutlich zu signalisieren.

1.4 Unsere Erfahrungen mit Vaadin

Wir verwenden Vaadin in vielen unserer Projekte und haben die verschiedensten Arten von Anwendungen bereits erfolgreich umgesetzt.

Es gibt prinzipiell keinen Typ Anwendung, bei dem wir sagen würden, dass Vaadin nicht verwendet werden kann – im Gegenteil sind die durch Vaadin zur Verfügung gestellten Abstraktionen und Funktionen so hilfreich, dass wir bei neuen Webapplikationen immer zur Verwendung von Vaadin tendieren. Dies gilt für Anwendungen im Intranet, im Internet, mit wenigen und vielen Benutzern, mit einfachen und komplexen Benutzerschnittstellen. Allerdings muss man bei komplexeren Benutzerschnittstellen darauf achten, dass die Komponenten nicht zu sehr verschachtelt werden (wozu man als unerfahrener Vaadin-Entwickler neigen kann), um die Applikation nicht unnötig zu bremsen.

Ein weiterer wichtiger Punkt ist, wie bei jeder Webapplikation, die Größe der Session. Vaadin legt die Benutzerschnittstellenobjekte (alle Objekte, die an den UI-Objekten hängen) in der Session ab und vergrößert diese damit natürlich (logischerweise umso mehr, je mehr verschachtelte Komponenten die Anwendung enthält). Nach dem Schließen des Browserfensters bleibt die Session so lange erhalten, bis sie vom System abgeräumt wird, und verbraucht damit weiterhin Speicher (sofern sich der Benutzer nicht explizit abmeldet). Wenn zusätzlich die Benutzer die gleiche Applikation mehrfach öffnen, dann legt Vaadin diese Objektbäume natürlich mehrfach in der Session ab. Dies ist aber nichts Ungewöhnliches und wird auch von anderen serverzentrierten Frameworks in gleicher Weise gelöst. In der praktischen Verwendung haben wir noch keine Probleme erlebt, die die Verwendung von Vaadin verhindert hätten, und auch die notwendige Größe der verwendeten Server ist im normalen Bereich. Ob allerdings eine Webapplikation wie Facebook mit Vaadin performant wäre, wagen wir zu bezweifeln.

Die eingebauten Kommunikationsmechanismen erlauben eine gute und reaktionsfähige Interaktion auch mit vergleichsweise niedrigen Datenraten des unterliegenden Netzes, solange die Latenzzeiten nicht zu hoch werden.

CenterDevice

Ein Beispiel, das die Verwendung von Vaadin mit vielen Nutzern und hohen nichtfunktionalen Anforderungen sehr gut demonstriert, ist die Cloud-Anwendung CenterDevice (siehe [CenterDevice]), die von Mitarbeitern der codecentric entwickelt wurde. CenterDevice ist eine Anwendung für Cloud Storage, in der Sie Ihre Daten in einer deutschen Cloud ablegen, automatisch verschlagworten und mit anderen teilen können. Hier haben wir eine komplexe Benutzerschnittstelle kombiniert mit sehr vielen Benutzern und großen Datenmengen, die vom und zum Nutzer transferiert werden müssen und für die auch

(zumindest für die gängigsten Formate) eine direkte Darstellung in der Applikation möglich sein muss.

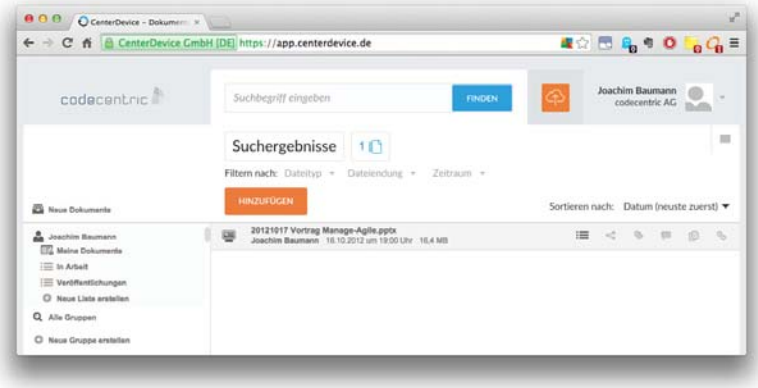

Abb. 1–1
Beispiel einer komplexen Vaadin-Applikation (CenterDevice)

Hier waren tatsächlich einige Iterationen notwendig, bis die Applikation die gewünschte Reaktivität hatte, auch und gerade beim Umgang mit großen Dateien. Aber das Ergebnis spricht für sich, und wir würden uns auch in diesem Fall mit den sehr hohen Anforderungen wieder für Vaadin entscheiden. Holen Sie sich bei Interesse einfach einen Test-Account und urteilen Sie selbst.

Einzig Offline-Applikationen, die ohne Verbindung mit dem Server uneingeschränkt funktionieren, sind mit Vaadin ohne zusätzliche Arbeit nicht möglich. Vaadin bietet zwar rudimentäre Unterstützung, aber für die Offline-Unterstützung ist trotzdem eine Menge an Handarbeit nötig, die die Vorteile von Vaadin einschränkt.

1.5 Weitergehende Informationen

Auf die Website von Vaadin gelangen Sie unter folgender URL:

http://www.vaadin.com

Hier gibt es nicht nur die jeweils aktuelle Version von Vaadin, Add-ons für die Entwicklung und die Integration in verschiedene Entwicklungsumgebungen, sondern auch Tutorials und eine Online-Version des »Book of Vaadin«, der Referenz für die Verwendung von Vaadin.

Außerdem finden Sie dort das Vaadin-Forum, das die direkte Kommunikation mit den Vaadin-Entwicklern ermöglicht. Falls Sie nicht den Umweg über die Vaadin-Website nehmen wollen, folgen Sie einfach der URL:

http://www.vaadin.com/forum

2 Erste Schritte mit Vaadin

In diesem Kapitel werden wir die Entwicklungsumgebung Eclipse einrichten, unser erstes Vaadin-Projekt aufsetzen und eine einfache Hello-World-Applikation schreiben. Wir werden sehen, wie wir Maven verwenden können, um Vaadin-Projekte zu bauen, und zum Schluss noch einen kurzen Blick auf andere Entwicklungsumgebungen werfen.

2.1 Einrichten der Entwicklungsumgebung Eclipse

Für das einfache Arbeiten mit Vaadin in Eclipse benötigen wir Java und Eclipse, das Eclipse-Plugin für Vaadin-Entwicklung, das von Vaadin selbst zur Verfügung gestellt wird, einen Servlet-Container (wir verwenden der Einfachheit halber Tomcat) und Maven als Build-Werkzeug für unsere Builds außerhalb von Eclipse.

Java und Eclipse

Die neueste Java-Version finden Sie auf der Website von Oracle (siehe [Javasoft]). Wählen Sie einfach die neueste SE-Version (Standard Edition) für Ihr Betriebssystem und Ihre Architektur (32 oder 64 Bit) und installieren Sie diese auf Ihrem System. Hierzu müssen Sie eventuell auch die Pfadinformation anpassen, so dass Ihnen Java auch auf der Kommandozeile zur Verfügung steht. Mit dem Aufruf `java –version` prüfen Sie, dass Java erfolgreich installiert ist.

```
~ > java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)
~ >
```

Listing 2–1
Java ist erfolgreich installiert.

Im nächsten Schritt laden wir Eclipse herunter (siehe [Eclipse]). Da wir Webapplikationen entwickeln wollen, wählen wir das Paket *Eclipse*

IDE for Java EE Developers, das alle benötigten Eclipse-eigenen Werkzeuge enthält. Laden Sie die Version herunter, die der Architektur Ihrer installierten Java-Version entspricht (32 oder 64 Bit). Packen Sie das Archiv aus und bewegen Sie das resultierende Verzeichnis `eclipse` an einen Ort Ihrer Wahl.

Starten Sie nun die im Eclipse-Verzeichnis zu findende Applikation. Beim Start fragt Eclipse nach dem Pfad zu einem Workspace-Verzeichnis, das mehrere Projekte zusammenfasst. Wählen Sie ein Verzeichnis (oder das voreingestellte) und sagen Sie ok. Wenn Eclipse die Willkommensseite präsentiert, dann ist Eclipse erfolgreich installiert.

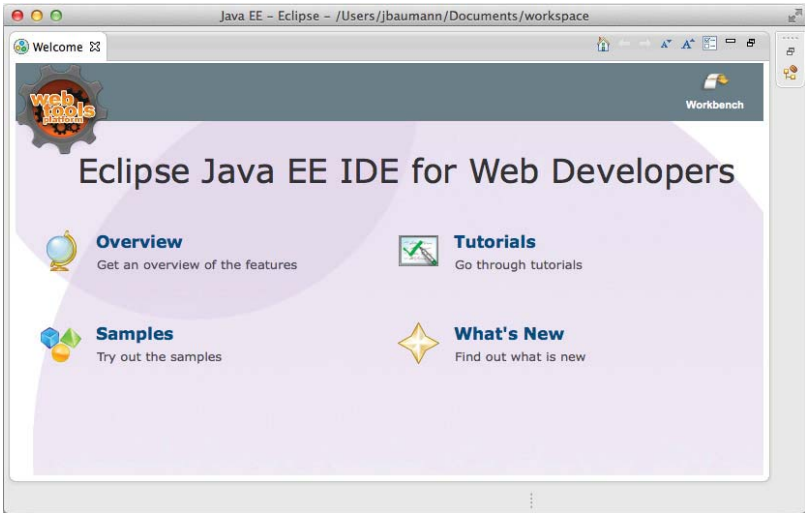

Abb. 2–1 *Willkommensseite der Eclipse-IDE*

Das Vaadin-Plugin

Nach der Installation von Eclipse können wir das Vaadin-Plugin installieren, das alle benötigten Vaadin-Bibliotheken sowie das Book of Vaadin, die offizielle englische Referenz, für das leichtere Nachschlagen mitbringt.

Mit dem Vaadin-Plugin können wir Vaadin-Projekte anlegen, neue Widgets oder Themes erzeugen, mit einem visuellen Editor, dem *Visual Designer*, Layouts definieren und unsere erzeugten Widgets debuggen. Deshalb empfehlen wir bei der Verwendung von Eclipse grundsätzlich, das Vaadin-Plugin für die Vaadin-Entwicklung zu verwenden.

Die Installation des Vaadin-Plugin

Das Vaadin-Plugin können Sie über den Eclipse-eigenen Marketplace installieren (hier können Hersteller ihre Plugins registrieren). Hierfür wählen Sie den Menüpunkt *Help → Eclipse Marketplace* aus und geben im Suchfeld den Text *Vaadin* ein. Beim nun aufgelisteten Vaadin-Plugin müssen Sie nur noch den Button *Install* anwählen, und

das Plugin wird installiert. Wenn Sie alle Fragen beantwortet haben und Eclipse neu gestartet ist, sollte das Plugin aktiv sein.

Sie können die korrekte Installation verifizieren, indem Sie den Menüpunkt *File → New → Other* anwählen. In der Liste der Assistenten für neue Artefakte sollten verschiedene Vaadin-Assistenten auftauchen.

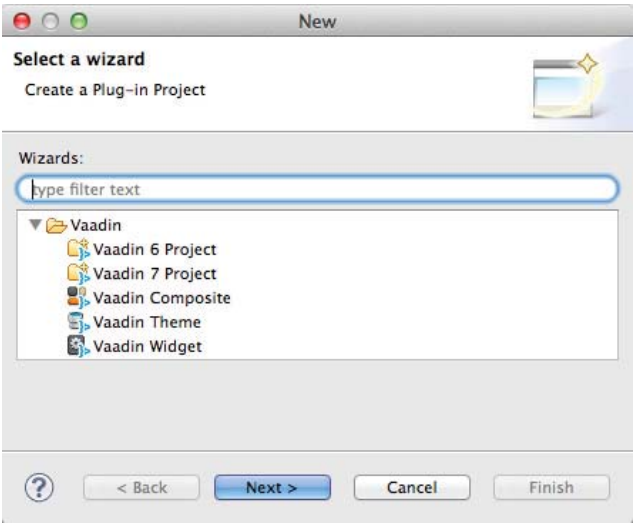

Abb. 2–2
Liste der in Eclipse zur Verfügung stehenden Vaadin-Assistenten

Weitere benötigte Werkzeuge

Installation des Tomcat

Um unsere Vaadin-Applikationen auszuführen, benötigen wir einen Servlet-Container. Im Rahmen des Buchs verwenden wir *Apache Tomcat* in der Version 7. Sie können das zugehörige Archiv für Ihr Betriebssystem und Ihre Architektur direkt von der Website herunterladen (siehe [Apache Tomcat]). Packen Sie das Archiv aus und legen Sie es an einem Ort Ihrer Wahl ab.

> Bei Windows-Systemen ist `C:\Dev\Java` eine gute Wahl, unter Unix-Systemen und Mac OS X können Sie `/opt/Java` oder `/usr/local/Java` nehmen, und unter Mac OS X ist auch `~/Library/Java` eine Möglichkeit. Zu guter Letzt können Sie den Tomcat auch einfach in Ihr Eclipse-Workspace-Verzeichnis legen.

Im nächsten Schritt konfigurieren wir Eclipse so, dass es den Tomcat kennt. Hierfür gibt es verschiedene Wege, wir gehen den Weg über den Assistenten. Wählen Sie den Menüpunkt *File → New → Other* aus. In der aufgeführten Liste gibt es unter dem Punkt `Server` den gleichnamigen Assistenten. In der folgenden List finden Sie unter `Apache` die Tom-

cat-Version, die Sie heruntergeladen haben, und geben im nach dem Klick auf den `Next`-Button erscheinenden Dialog den Pfad zu Ihrem Tomcat-Verzeichnis an. Der Klick auf den *Finish*-Button legt Ihre Serverkonfiguration an.

> Allgemein ist es vernünftig, eine spezifische Version von Java für den Tomcat auszuwählen, und hierbei nicht das Java-Runtime-Environment JRE, sondern die volle Installation (Standard Edition). Damit stellen Sie sicher, dass Ihrem Tomcat alle Java-Klassen zur Verfügung stehen, die er eventuell benötigt. Für unsere Vaadin-Beispiele brauchen wir dies nicht, aber wenn Sie zum Beispiel JavaServer Pages ausliefern möchten, sind Sie auf den Compiler angewiesen, der im JRE nicht enthalten ist, sondern nur im SDK zur Verfügung steht.

Wenn Sie jetzt den Reiter `Servers` anwählen, können Sie den Tomcat mit Ihrer Serverkonfiguration starten (grüner Start-Button). Wenn die Installation korrekt ist, dann sollte nach kurzer Zeit der Server als *Started, Synchronized* markiert sein.

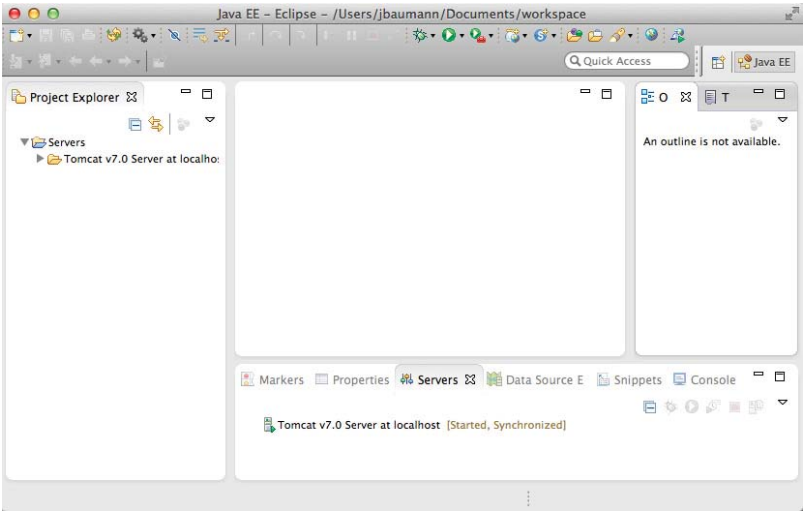

Abb. 2–3
Der gestartete Tomcat-Server in Eclipse

Installation von Maven

Für die Teile des Buchs, die sich mit der Erzeugung von Artefakten mit dem Build-Management-Werkzeug Maven beschäftigen, benötigen wir außerdem eine Maven-Installation und die zugehörige Konfiguration des Plugin `M2Eclipse` in Eclipse.

Auf der Website von Maven (siehe [Maven]) können Sie ein Archiv mit der aktuellen Version von Maven herunterladen. Packen Sie das Archiv aus, legen Sie das Verzeichnis an einem Ort Ihrer Wahl ab und

erweitern Sie den Pfad Ihrer verwendeten Shell oder Kommandozeile um das Unterverzeichnis `bin` Ihres Maven-Verzeichnisses.

> Sie können dieses Verzeichnis sehr gut neben das Tomcat-Verzeichnis legen, sind hier aber natürlich auch völlig frei in Ihrer Wahl. Die Pfaderweiterung machen Sie unter Windows mit der Benutzervariable `PATH`, die Sie über die Umgebungsvariablen erweitern können, unter Unix je nach verwendeter Shell in der zugehörigen Konfigurationsdatei. Dies geht unter Mac OS X auch, hier haben Sie aber zusätzlich die Möglichkeit, eine Textdatei, die den Pfad zu Ihrem `bin`-Verzeichnis enthält, im Verzeichnis `/etc/paths.d/` abzulegen. Dies erweitert den Pfad automatisch. Unter Mac OS X gibt es alternativ den Package Manager Homebrew, der sehr viele Programme, darunter auch Maven, sehr einfach installieren lässt (siehe [Homebrew]).

Da das Plugin M2Eclipse in Eclipse bereits enthalten ist, müssen wir ihm nur noch den Ort unserer Maven-Installation bekannt machen. Das Plugin bringt zwar auch eine eigene Maven-Version mit, aber diese ist vergleichsweise alt.

In den allgemeinen Einstellungen von Eclipse (erreichbar unter *Eclipse → Einstellungen*) gibt es einen eigenen Konfigurationsbereich namens *Maven*, unter dem sich verschiedene Unterpunkte befinden. Im Unterpunkt *Installations* können Sie Ihre eigene Maven-Installation eintragen. Klicken Sie auf den *Add*-Button, tragen Sie das Installationsverzeichnis Ihrer Maven-Version ein und klicken Sie auf den *Ok*-Button. Damit wird ab jetzt diese Maven-Version verwendet.

> Sollten Sie das Installationsverzeichnis bereits vergessen oder Maven über einen Package-Manager installiert haben, der den Installationsort nicht direkt ausgibt, geben Sie auf der Kommandozeile den Befehl `mvn –V` ein, der unter anderem das Installationsverzeichnis `Maven home:` auflistet.

Damit sind die Vorarbeiten für die Entwicklung mit Eclipse, dem Vaadin-Plugin, der Tomcat-Installation und Maven beendet, und wir können uns an dem ersten Projekt versuchen.

2.2 Erste Schritte

Wir beginnen damit, ein neues Vaadin-Projekt anzulegen. Über *File → New → Project* kommen wir zur Liste der Assistenten, aus der wir den Assistenten für ein Vaadin-7-Projekt auswählen. Wir klicken auf den *Next*-Button und kommen zur Konfiguration des Projekts. Hier sollte bereits unser Tomcat-Server eingetragen und eine Vaadin-7-Version

ausgewählt sein. Wir müssen nur noch den Projektnamen *VaadinHelloWorld* eingeben. Dieser Name taucht später in der URL unseres ersten Servlets auf, weshalb Sie hier einen nicht zu komplizierten Namen wählen sollten. Ein Klick auf den *Finish*-Button legt unser erstes Vaadin-Projekt an.

> Der Assistent erlaubt eine sehr weitgehende Konfiguration des späteren Projektes, nicht nur bezüglich der Java- und Vaadin-Version sowie der Version der Servlet-Spezifikation, sondern unter anderem auch der Zielarchitektur inklusive Google App-Engine oder einer Portlet-Umgebung. Auf den Folgeseiten, die wir für unser Beispiel überspringen, lassen sich fast alle Vaadin-spezifischen Aspekte der Applikation modifizieren, und auch die Generierung des Beispielquelltextes lässt sich problemlos abschalten.

Der Vaadin-Assistent fügt alle notwendigen Abhängigkeiten ein und lädt im Zweifelsfall sogar die Vaadin-Bibliotheken herunter. Zu guter Letzt wird auch ein einfaches Vaadin-Beispiel zur Verfügung gestellt, das den Test der korrekten Projektinitialisierung erlaubt. Wählen Sie hierfür im Kontextmenü der Java-Klasse den Menüpunkt *Run As* → *Run on Server* aus, und der Tomcat-Server wird gestartet, ein eingebauter Mini-Browser wird in Eclipse mit der richtigen URL geöffnet und eine einfache Vaadin-Applikation wird angezeigt. Nach der Prüfung können wir die vom Vaadin-Plugin erzeugte Beispielklasse (`VaadinhelloworldUI.java`) löschen.

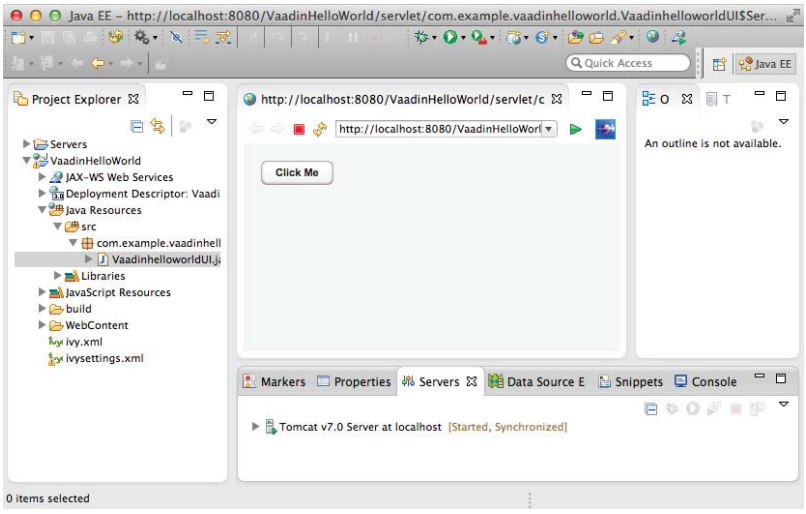

Abb. 2–4
Das erste Vaadin-Projekt in Eclipse

Unsere erste Applikation

Sehr kurz gefasst stellt eine Vaadin-Applikation eine Benutzeroberfläche im Browser dar. Die Inhalte der zugehörigen einzelnen Fenster oder Reiter werden durch User-Interface-Klassen (UI-Klassen), abgeleitet von einer Basisklasse `com.vaadin.ui.UI`, modelliert. Ausgehend von der Initialisierungsmethode `UI.init()` wird dazu ein Komponentenbaum aufgebaut, der zum Browser übertragen und dort dargestellt wird. Die notwendige Infrastruktur, damit diese UI-Klassen bei Aufrufen der jeweiligen URL korrekt identifiziert und angesprochen werden, wird von der Vaadin-Klasse `com.vaadin.server.VaadinServlet` implementiert. Diese kann für komplexere Fälle erweitert werden. Wir gehen auf diese Zusammenhänge später noch weiter ein, für unsere erste Applikation genügt diese Detailtiefe.

Das Vaadin-Servlet

Für einfachere Vaadin-Applikationen reicht es aus, wenn wir dem Servlet die zu verwendende UI-Klasse mitteilen. Dies tun wir über eine Annotation `@VaadinServletConfiguration`. Zusätzlich können wir noch über die Annotation `@WebServlet` angeben, unter welchem Namen unsere Applikation erreicht werden kann.

> Die Annotation `javax.servlet.annotation.WebServlet` gibt es seit der Servlet-Spezifikation 3.0. Mit ihr können die meisten Konfigurationseinstellungen, die früher in der `web.xml` durchgeführt wurden, direkt im Quelltext eingestellt werden (natürlich harmoniert Vaadin auch mit diesen früheren Versionen des Standards). Hierzu gehören auch die Initialisierungsparameter, die mit einer weiteren Annotation @WebInitParam gemacht werden können. Allerdings sind diese nicht typsicher, und Vaadin bietet in der Annotation `@VaadinServletConfiguration` die Möglichkeit, die Vaadin-spezifischen Initialisierungswerte typsicher zu definieren.

Um uns das Leben noch weiter zu erleichtern, bietet Vaadin die Möglichkeit, das Vaadin-Servlet als statische innere Klasse einer UI-Klasse zu definieren. In diesem Fall brauchen wir die UI-Klasse nicht mehr über die Annotation zu definieren. Vaadin verwendet automatisch die umgebende Klasse.

Unser Vaadin-Servlet sieht wie folgt aus:

```
@WebServlet("/*")
@VaadinServletConfiguration(ui = HelloWorld.class)
public static class Servlet extends VaadinServlet {
}
```

Listing 2–2
Das Vaadin-Servlet mit den notwendigen Annotationen

Wir beginnen mit der Annotation `@WebServlet`, der wir ein Muster mitgeben, das beliebige Namen für unsere Applikation erlaubt. Dann verwenden wir die Annotation `@VaadinServletConfiguration`, um als UI-

Klasse die Klasse `HelloWorld` zu spezifieren. Diese Zeile ist optional, sofern wir das nun folgende `VaadinServlet` als statische innere Klasse der UI-Klasse `HelloWorld` definieren.

Die UI-Klasse

In der UI-Klasse `HelloWorld` definieren wir unsere Benutzeroberfläche, die Komponenten und ihr Layout in der Methode `init()`, die beim ersten Aufruf der Applikation ausgeführt wird. Als Parameter wird ein `VaadinRequest` übergeben, der den spezifischeren Request-Typ (`ServletRequest` oder `PortletRequest`) kapselt und die Initparameter zur Verfügung stellt.

Listing 2–3
Unsere erste Vaadin-Applikation

```
package de.vaadinbuch.einfuehrung;

import ...

public class HelloWorld extends UI {

       @WebServlet("/*")
    public static class Servlet extends VaadinServlet {
    }
    protected void init(final VaadinRequest request) {
        final HorizontalLayout layout = new HorizontalLayout();
        setContent(layout);
        final TextField name = new TextField
                                    ("Geben Sie bitte Ihren Namen ein");
        layout.addComponent(name);
            name.addTextChangeListener(new TextChangeListener() {
            public void textChange(final TextChangeEvent event) {
                final String name = event.getText();
                Notification.show("Hallo, " + name);
            }
        });
    }
}
```

Nach der Package-Deklaration folgen die benötigten Import-Statements.

Nun definieren wir die Klasse HelloWorld, die wie beschrieben von `com.vaadin.ui.UI` abgeleitet ist. In dieser definieren wir unser Vaadin-Servlet als statische innere Klasse. Damit können wir auf die Annotation zur Festlegung der UI-Klasse verzichten und benötigen nur die Annotation zur Festlegung des Zugriffsmusters.

Die Methode init()

Jetzt kommen wir zur eigentlichen Festlegung unserer Benutzerschnittstelle, die wir in der Methode `init()` durchführen. Dies wird von Vaadin so vorgegeben, da die UIs erst dann erzeugt werden, wenn der erste Zugriff auf sie erfolgt, und damit zum Zeitpunkt der Erzeugung des Objekts nicht grundsätzlich alle benötigten Informationen zur Erzeugung vorhanden sind. Das übergebene Objekt vom Typ `VaadinRequest` enthält dabei die Parameter dieses ersten Zugriffs.

In unserer `init()`-Methode erzeugen wir im ersten Schritt ein Objekt vom Typ `com.vaadin.ui.HorizontalLayout`, das wir mit Aufruf der Methode `setContent()` zum Inhalt unserer Benutzerschnittstelle machen. Ein Layout ist eine `Container`-Komponente, die weitere Komponenten unserer Benutzerschnittstelle aufnimmt und nach bestimmten Vorgaben anordnet. Im Falle des `HorizontalLayout` werden alle enthaltenen Komponenten nebeneinander dargestellt.

Im nächsten Schritt erzeugen wir ein `com.vaadin.ui.TextField` mit einer Beschriftung und fügen es mit dem Aufruf der Methode `addComponent()` dem Layout hinzu. Dieses erlaubt uns beliebige Eingaben, die dann in Folge verarbeitet werden können.

Unser Layout ist damit fertig, aber wir wollen noch eine (wenn auch minimale) Reaktion auf unsere Eingaben erhalten. Wir verwenden die minimale Variante und registrieren einen `TextChangeListener`, der bei jeder Änderung des Eingabefeldes aufgerufen wird. Die zugehörige Methode `textChange()` ist hierbei sehr einfach; sie extrahiert den aktuellen Wert aus dem übergebenen Event und ruft die Methode `show()` der Klasse `com.vaadin.ui.Notification` auf. Diese Methode wird von Vaadin bereitgestellt, um kurzzeitig anzuzeigende Meldungen für den Benutzer darzustellen.

Um diese Klasse in Eclipse zu erzeugen, wählen Sie den Wizard für das Erzeugen einer neuen Klasse an (zum Beispiel über das Kontextmenü des Verzeichnisses *Java Resources/src*), tragen als Package `de.vaadinbuch.einfuehrung`, als Klassennamen `HelloWorld` und als Elternklasse `com.vaadin.ui.UI` ein, fügen die statische innere Klasse und den Inhalt der `init()`-Methode ein und rufen *Organize Imports* auf, um die fehlenden Import-Befehle einfügen zu lassen (wählen Sie aus der vorgeschlagenen Liste immer die Vaadin-Implementierung).

Wählen Sie jetzt aus dem Kontextmenü der `HelloWorld`-Klasse den Menüpunkt *Run As* → *Run on Server* aus, der Tomcat-Server wird gestartet und unsere Applikation angezeigt. Wenn Sie im Textfeld etwas eingeben, wird die entsprechende Meldung ausgegeben.

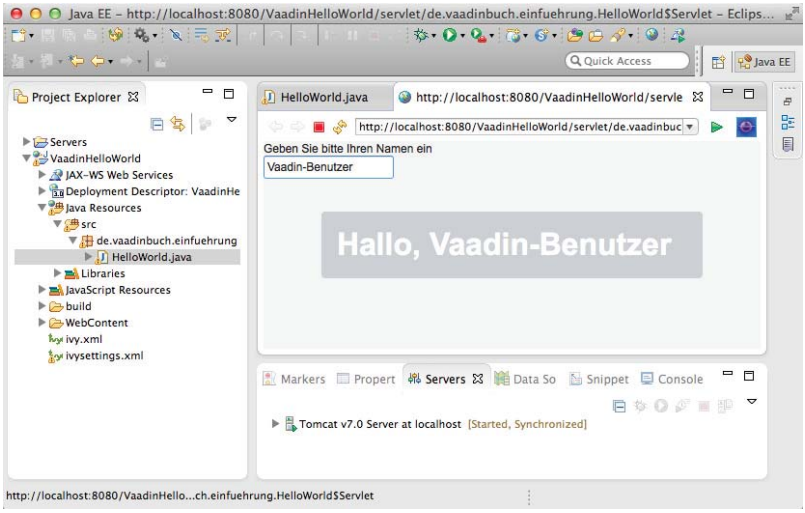

Abb. 2–5
Unsere erste Vaadin-Applikation in Aktion

Debugging mit Eclipse

Sie können Ihre Vaadin-Applikation mit Eclipse sehr gut debuggen. Entfernen Sie zuerst die Deklaration `final` des `String`-Objekts in der Methode `textChange()` und setzen Sie dann in der darauffolgenden Zeile einen Breakpoint. Danach wählen Sie aus dem Kontextmenü der `HelloWorld`-Klasse den Menüpunkt *Debug As → Debug on Server* aus, der Tomcat-Server wird gestartet und unsere Applikation angezeigt.

Wenn Sie jetzt eine Eingabe machen, wechselt Eclipse in den Debug-Modus, und Sie können den Wert des `String`-Objekts über den *Variables-View* einsehen und ändern. Wenn Sie Ihre Applikation weiterlaufen lassen, werden Sie den von Ihnen geänderten Wert in der Anzeige sehen.

> Interessant ist es hierbei auch, einige Zeit zu warten. Der Clientteil der Vaadin-Applikation verfügt für Serveraufrufe über einen Timeout, nach dessen Erreichen dem Benutzer eine Fehlermeldung angezeigt wird, die auf den Kommunikationsfehler mit dem Server hinweist. Dies macht auf der einen Seite das Kommunikationsverhalten stabiler, stellt aber für das Debugging eine kleine Einschränkung dar.

Zusätzlich können Sie auch auf der Clientseite einige Informationen erhalten, indem Sie an die Applikations-URL die Zeichenfolge `?debug` anhängen. Dies sorgt dafür, dass Vaadin ein Debug- und Log-Fenster öffnet, das uns einen tiefen Einblick in die Zusammenarbeit von Client und Server geben kann. Auch wenn unsere momentanen Kenntnisse

noch nicht für die Interpretation der Details genügen, so ist es doch interessant, diese Zusammenarbeit im Log zu beobachten.

2.3 Verwendung von Maven für Vaadin-Projekte in Eclipse

Um Maven für Eclipse-Projekte zu verwenden, gibt es viele Wege. Der einfachste für unsere Zwecke ist die Erzeugung eines neuen Maven-Projektes mit dem von Vaadin bereitgestellten Archetyp für Vaadin-Projekte (Archetypen sind Projektvorlagen für Maven, wir werden später noch auf die Details eingehen).

Wir generieren ein neues Maven-Projekt über den Menüpunkt *File → New → Maven Project*. Wir belassen die Voreinstellungen und wählen den *Next*-Button. Auf der nun folgenden Seite können wir einen Archetyp auswählen. Allerdings ist die Wahrscheinlichkeit hoch, dass der Vaadin-Archetyp noch nicht in der Liste auftaucht (tippen Sie im Filterfeld Vaadin ein, um dies zu prüfen). Um den Vaadin-Archetyp der Liste hinzuzufügen, wählen Sie den Button *Add Archetype...* .

Es erscheint ein Dialog, in dem Sie die Details des Archetyps angeben müssen. Für den benötigten Archetyp lauten die Informationen:

Name	Inhalt
Archetype Group Id	com.vaadin
Archetype Artifact Id	vaadin-archetype-application
Archetype Version	LATEST
Repository URL	Kann freibleiben (Voreinstellung ist Maven Central)

Tab. 2–1
Maven-Archetyp: Benötigte Informationen

Wenn Sie die Informationen eingetragen haben, wählen Sie den *Ok*-Button. Damit wird der Archetyp heruntergeladen, eingetragen und ist verwendbar. Wählen Sie den Archteyp aus und klicken Sie den *Next*-Button.

Im nächsten Dialog fragt Eclipse Details für die Anlage unseres Projekts ab. In der Liste der Eigenschaften steht die Angabe eines eigenen Theme, die Sie stehen lassen und für den Moment ignorieren können.

Wählen Sie hier die folgenden Einstellungen, um unser Beispiel aus Unsere erste Vaadin-Applikationdirekt übernehmen zu können:

Tab. 2–2
Maven-Projekt: Benötigte Informationen

Name	Inhalt
Group Id	de.vaadinbuch
Artifact Id	einfuehrung
Version	0.0.1-SNAPSHOT (sollte voreingestellt sein)
Package	de.vaadinbuch.einfuehrung (sollte automatisch ausgefüllt werden)

Wählen Sie den *Finish*-Button, und das Projekt wird erstellt. Navigieren Sie zum Quelltext (*Java Resources → src/main/java → de.vaadinbuch.einfuehrung*), und Sie sehen, dass der Maven-Archetyp eine Beispielklasse mit zugehöriger XML-Datei angelegt hat. Löschen Sie diese und fügen Sie unsere Klasse aus Unsere erste Vaadin-Applikationein.

Um unser Beispiel zu starten, wählen Sie aus dem Kontextmenü der `HelloWorld`-Klasse den Eintrag *Run as → Run on Server*, wählen den Tomcat als Server, und unsere Applikation startet, diesmal erzeugt über Maven.

2.4 Andere Entwicklungsumgebungen

Nicht alle Entwickler möchten oder können Eclipse als Entwicklungsumgebung verwenden. Hierfür gibt es auch für andere Entwicklungsumgebungen Integrationen, die die Entwicklung mit Vaadin erleichtern.

IntelliJ IDEA

Für IDEA gibt es seit 2012 ein externes Plugin für die Vaadin-Entwicklung. Dieses bietet, da es eine Portierung des Eclipse-Plugin ist, eine ähnliche Unterstützung für die Entwicklung von Vaadin-Applikationen, Widgetsets, Themes und die Verwendung des Vaadin Designers an.

Seit der Version 13 bietet IDEA direkte Unterstützung für die Vaadin-Entwicklung in der Ultimate Edition. In der Community Edition gibt es diese Unterstützung nicht. In beiden Versionen steht aber wie in Eclipse die Maven-Unterstützung zur Verfügung, die für viele Anwendungsfälle die interessantere Variante sein kann.

Das Vaadin-Plugin

Wenn Sie die Entwicklungsumgebung in der Ultimate Version neu herunterladen und das erste Mal starten, sollte das Vaadin-Plugin bereits aktiviert sein. Prüfen Sie dies, indem Sie das *Preferences*-Fenster öffnen, im Suchfeld *Vaadin* eingeben und dann unter den Plugin-Einstellungen verifizieren, dass die Vaadin-Unterstützung aktiviert ist.

Um das Vaadin-Plugin zu verwenden, erzeugen Sie ein neues Projekt vom Typ Java. Auf der nun folgenden Seite wählen Sie zum einen

das Vaadin-Framework aus. Da das Vaadin-Plugin keine Vaadin-Version mitbringt, müssen Sie die gewünschte Version herunterladen. IDEA liefert freundlicherweise gleich den zugehörigen Link. Zum anderen benötigen Sie noch einen Application Server. Setzen Sie also auch bei diesem Eintrag einen Haken, und falls Sie noch keinen Application Server konfiguriert haben, klicken Sie auf *New*, wählen *Tomcat Server* aus und geben das Verzeichnis Ihrer Tomcat-Installation an. Wählen Sie den *Finish*-Button an, und Ihr Projekt wird erzeugt. Unter dem Ordner `External Libraries` sollten Sie sowohl Ihren Application Server als auch die Vaadin-Bibliotheken sehen.

Erzeugen Sie jetzt eine neue Klasse `de.vaadinbuch.einfuehrung.HelloWorld` mit dem Inhalt von Unsere erste Vaadin-Applikation, indem Sie im Kontextmenü des Verzeichnisses `src` den Eintrag *New* → *Java Class* anwählen.

Öffnen Sie zum Schluss die Datei `web.xml` im Ordner `web` → `WEB-INF`. Wenn das Plugin dort Einträge für `Servlet` und `Servlet-Mapping` eingefügt hat, löschen Sie diese einfach. Wir konfigurieren diese Information in unserem Beispiel vollständig über die entsprechenden Annotationen.

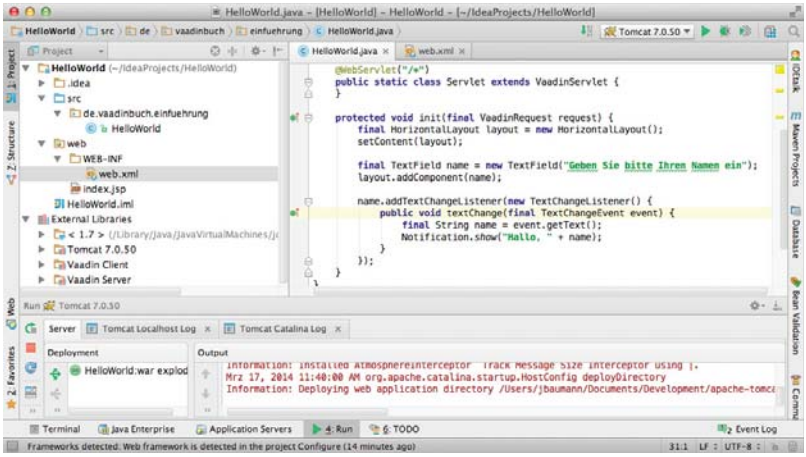

Abb. 2–6
Unsere erste Applikation mit IntelliJ IDEA

Wählen Sie nun aus dem Menü `Run` den Eintrag `Run Tomcat` (der dargestellte Name ist der von Ihnen bei der Konfiguration des Application Servers vergebene). IDEA startet den Application Server mit Ihrer Vaadin-Applikation und öffnet ein Browserfenster zur Anzeige.

Die Maven-Unterstützung

Wenn Sie die Community-Version von IDEA verwenden oder aber die Maven-gestützte Erzeugung von Projekten in der Ultimate-Version nutzen wollen, wählen Sie *File* → *New Project* und selektieren *Maven* als Typ. Geben Sie als Projektnamen *VaadinHelloWorld* an und klicken Sie den *Next*-Button. Im nun angezeigten Dialog können wir

GroupId, *ArtifactID* und *Version* angeben (verwenden Sie die Werte aus Maven-Projekt: Benötigte Informationen) sowie den Maven-Archetyp auswählen. Wählen Sie hierfür *Create from Archetype* aus und, falls der Vaadin-Archetyp noch nicht aufgeführt ist, *Add Archetype...* . Hier geben Sie die Informationen aus Maven-Archetyp: Benötigte Informationenein, und IDEA lädt den Archetyp. Klicken Sie auf *Next* (der Vaadin-Archetyp sollte ausgewählt sein), prüfen Sie die angezeigten Informationen und lassen Sie mit Klick auf den *Finish*-Button das Projekt erzeugen.

IDEA benötigt im nächsten Schritt den Import der Maven-Informationen. Hierzu sollte ein Hinweis dargestellt werden, den Sie anklicken können, um dies durchzuführen (wählen Sie am besten *Auto-Import*). Sie können aber auch im Kontextmenü des erzeugten Moduls *Maven → Reimport* anwählen.

Löschen Sie wie zuvor die erzeugte Beispielklasse und die zugehörige XML-Datei unter `src/main/vaadin` und erzeugen Sie wie mit dem Vaadin-Plugin unsere Beispielklasse `de.vaadinbuch.einfuehrung.HelloWorld`.

Jetzt benötigen wir noch eine Konfiguration, um den Tomcat mit unserer Applikation zu starten. Wählen Sie hierzu den Menüeintrag *Run → Edit Configurations...*, klicken Sie im sich öffnenden Fenster auf das Pluszeichen und wählen Sie *Tomcat Server → Local*. Auf dem Reiter Deployment klicken Sie auf das Pluszeichen und dann *Artifact...*, um *VaadinHelloWorld:war exploded* auszuwählen. Geben Sie als Namen VaadinHelloWorld ein und klicken Sie auf den *Ok*-Button.

Sie können jetzt mit Run → VaadinHelloWorld den Tomcat starten, und kurz danach wird unsere Beispielapplikation im Browser dargestellt.

NetBeans

NetBeans verfügt seit der Version 7.3 über ein Plugin für die Vaadin-Entwicklung. Das Plugin bietet Assistenten für die Projekterzeugung, eine Integration des Vaadin-eigenen Add-on-Verzeichnisses und verschiedene Vorlagen für die Erleichterung der Programmierung an. Wählen Sie die Java-EE-Variante von NetBeans für Ihre Experimente.

Zur Installation des Plugin wählen Sie im Menü *Tools* den Menüpunkt *Plugins*, um die Plugin-Verwaltung anzuzeigen. Wechseln Sie auf den Reiter der verfügbaren Plugins (*Available Plugins*) und geben Sie als Suchzeichenkette Vaadin ein. Setzen Sie den Haken und klicken Sie den *Install*-Button.

NetBeans verwendet Maven für die Erzeugung des Projekts.

Nach der Installation können Sie ein neues Projekt erzeugen (*File → New Project*) und als Vorlage ein *Vaadin-Web-Application-Project* auswählen. NetBeans verwendet Maven zur Erzeugung der Projekt-

struktur, Sie müssen also die entsprechenden Angaben für Projektnamen, Group-Id und bei Bedarf Package-Namen machen.

Ähnlich wie beim Eclipse-Plugin wird auch hier die vollständige Projektstruktur erzeugt, allerdings hier von vornherein auf Maven basierend. Das Plugin erstellt dabei die gleiche Beispielklasse, die wir zum Testen der Installation verwenden können.

Wenn Sie noch nie mit NetBeans gearbeitet haben, sollten Sie zumindest noch den Tomcat-Server installieren, um die gleiche Umgebung zu haben (alternativ können Sie auch Jetty oder Glassfish verwenden). Wählen Sie hierzu *Tools* → *Server*, dann den Button *Add Server*, wählen Sie den *Tomcat-Server* aus und geben Sie den Pfad zu Ihrer Installation an (denken Sie daran, einen Benutzer und ein Passwort zu setzen zur Kommunikation von Tomcat und NetBeans).

Nun klicken Sie auf den Button *Run Project*, und Ihr Projekt wird mit Maven gebaut, im Server deployt und es wird ein Browserfenster mit unserer Applikation geöffnet.

Im nächsten Schritt können Sie jetzt unsere `HelloWorld`-Klasse implementieren und erhalten ein ähnliches Ergebnis wie in der folgenden Abbildung.

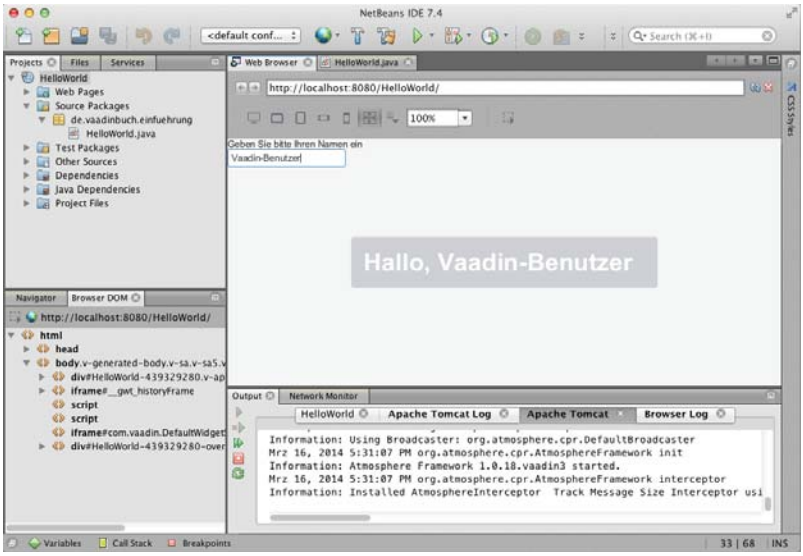

Abb. 2–7
Unsere erste Applikation mit NetBeans