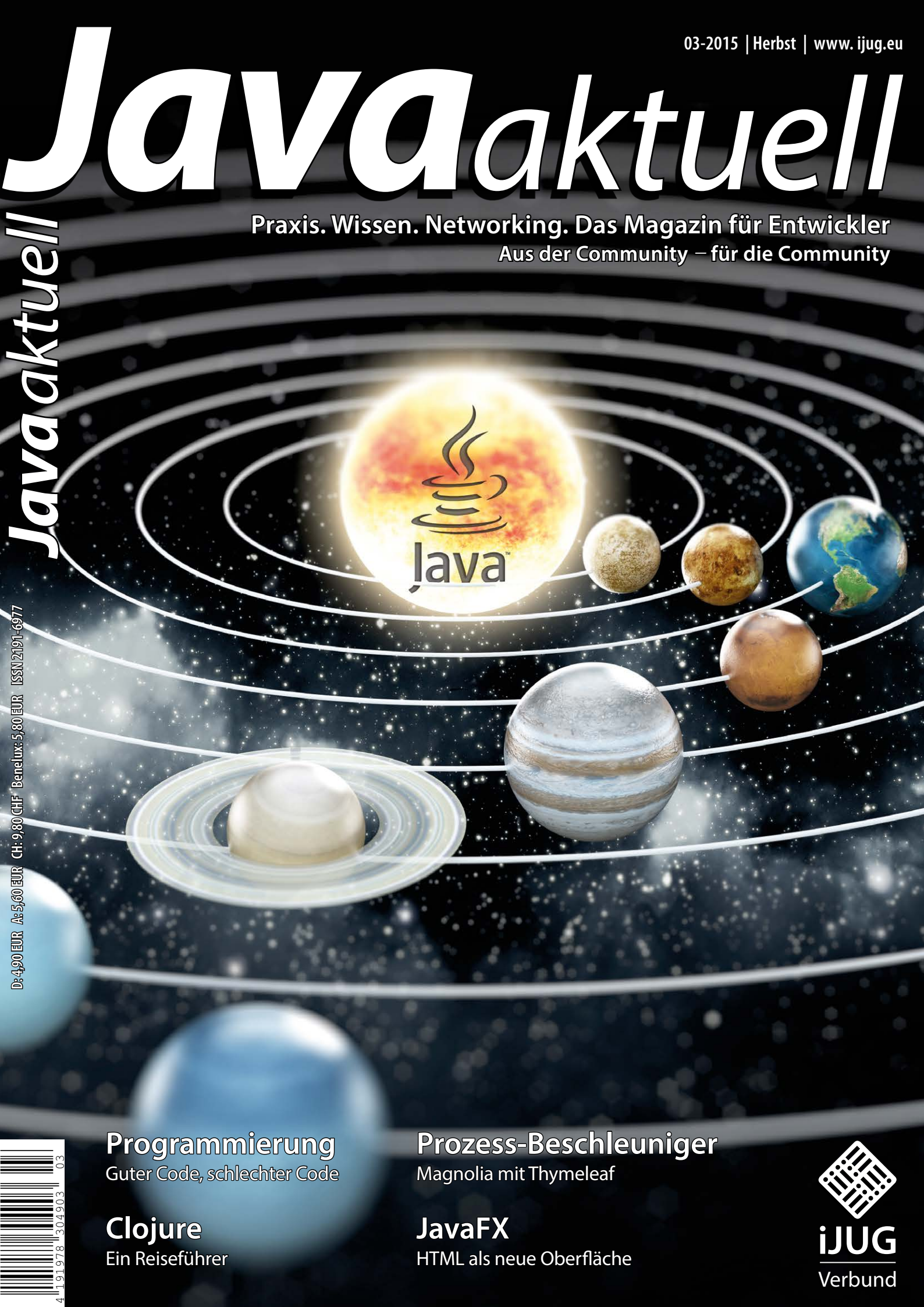
03-2015 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell



D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Programmierung
Guter Code, schlechter Code

Clojure
Ein Reiseführer

Prozess-Beschleuniger
Magnolia mit Thymeleaf

JavaFX
HTML als neue Oberfläche

iJUG Verbund

Kunstprojekt im JavaLand 2015

Seit Java 1.5 erlaubt die Java Virtual Machine die Registrierung sogenannter „Java-Agenten"

Bei Mgnolia arbeiten Web-Entwickler und CMS-Experten mit ein und demselben Quellcode

Ein Heim-Automatisierungs-Projekt

Write once – App anywhere

Axel Marx, Tricept Informationssysteme AG

Die Entwicklung mobiler Anwendungen ist ein stetig an Bedeutung zunehmender Bereich. Vor allem Cross-Platform-Ansätze sind für die Entwicklergemeinde interessant, um mit (hoffentlich) geringem Aufwand Anwendungen für ein möglichst breites Gerätespektrum anbieten zu können. Dieser Artikel zeigt Möglichkeiten, mobile Anwendungen soweit möglich mit Java-Know-how zu entwickeln, ohne sich tiefer mit JavaScript- oder HTML5-Programmierung auseinandersetzen zu müssen, und richtet sich an Java-Entwickler, für die das Schlagwort „mobile Entwicklung" mehr bedeutet als die bloße Gestaltung von GUIs oder die Erstellung mobiler Websites.

Der Autor ist seit vielen Jahren in der Entwicklung von Client-Server-Systemen tätig und beschäftigt sich dabei mit dem Entwurf und der Implementierung sowohl der Oberflächen als auch der Fachlogik mithilfe von Java-Technologien. Ausgehend von diesem Hintergrund sowie der wachsenden Bedeutung mobiler Anwendungen und der Frage „Nativer oder Cross-Platform-Ansatz bei der Entwicklung?" mit ihren unterschiedlichen Möglichkeiten der Umsetzung, stellte sich die Frage, inwieweit (und ob) Plattformunabhängige Entwicklung mit Java überhaupt möglich ist.

Wer sich als Java-Entwickler mit dem Gedanken trägt, seine Java-Kenntnisse zur Entwicklung mobiler Anwendungen einzusetzen und dabei möglichst ohne tiefere Kenntnisse in den Standard-Techniken der mobilen Entwicklung wie HTML5, CSS oder JavaScript auszukommen zu wollen, der sollte sich zuerst ein-

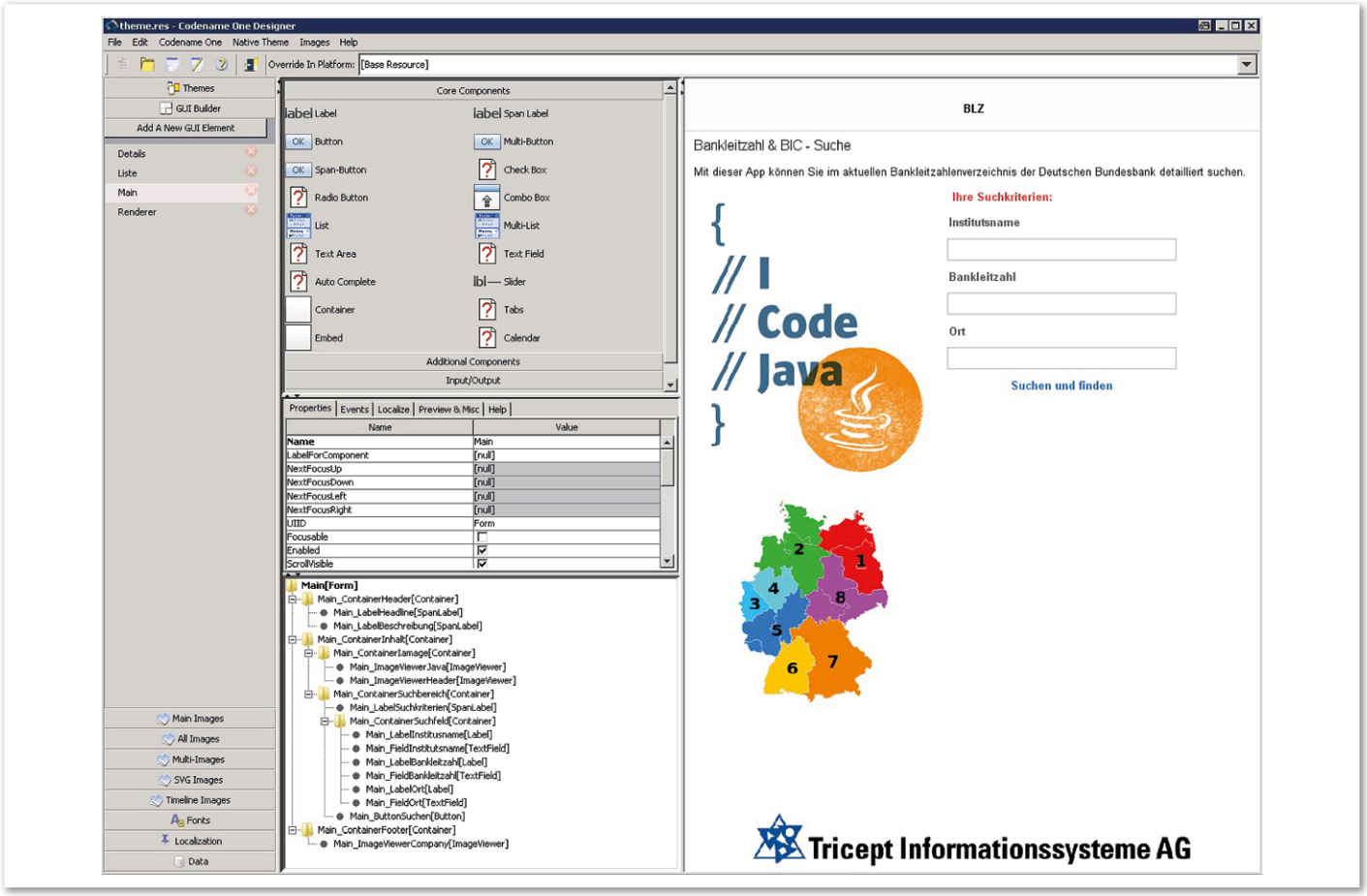


Abbildung 1: Die App in der Entwicklungsumgebung

```
140510001Sparkasse Mecklenburg-Nordwest   23951Wismar, Meckl      Spk Mecklenburg-Nordwest   51499NOLADE21WIS20048818U000000000
140510002Sparkasse Mecklenburg-Nordwest   19217Rehna             Spk Mecklenburg-Nordwest   51499      20048244U000000000
140510002Sparkasse Mecklenburg-Nordwest   23996Bad Kleinen       Spk Mecklenburg-Nordwest   51499      20048824U000000000
140510002Sparkasse Mecklenburg-Nordwest   23946Ostseebad Boltenhagen Spk Mecklenburg-Nordwest 51499      20048825U000000000
140510002Sparkasse Mecklenburg-Nordwest   23942Dassow            Spk Mecklenburg-Nordwest   51499      20048826U000000000
140510002Sparkasse Mecklenburg-Nordwest   23972Dorf Mecklenburg  Spk Mecklenburg-Nordwest   51499      20048827U000000000
140510002Sparkasse Mecklenburg-Nordwest   19205Gadebusch         Spk Mecklenburg-Nordwest   51499      20048828U000000000
```

Abbildung 2: Auszug aus der XML-Datei mit den Bank-Informationen

mal einen Überblick über den aktuellen Stand der Technik und die am Markt verfügbaren Werkzeuge und ihre Eigenarten verschaffen.

Abhängig von den zur Wahl stehenden Werkzeugen gibt es verschiedene Ansätze, sich dem Thema zu nähern. Während mittlerweile fast jeder Anbieter die gängigen Entwicklungsumgebungen wie Eclipse, NetBeans und IntelliJ IDEA mehr oder weniger gut unterstützt, liegen die Unterschiede dabei vor allem im Aufbau und in der Struktur der erzeugten App (kompilierter Binärcode zur Installation auf dem Zielgerät oder auf einen Server ausgelagerte Fach- und Darstellungslogik mit einem Viewer auf dem Mobilgerät für das Rendering der GUI) sowie in den verwendeten Techniken zur Umsetzung von UI und Fachlogik (ohne oder mit unterschiedlich hohen Anteilen an HTML5, CSS und JavaScript). Zunächst ein Überblick über die betrachteten Entwicklungswerkzeuge.

Codename One

Das Tool erlaubt die Entwicklung von Standalone-Clients in Java, ohne dass weitere Arbeiten mit HTML5, CSS oder JavaScript erfolgen müssen, und ist als Plug-in für alle gängigen Entwicklungsumgebungen wie Eclipse, NetBeans oder IntelliJ IDEA erhältlich. Sowohl Fachlogik als auch Oberflächenelemente werden in Java entwickelt, wobei für die Erstellung der UI neben der Verwendung von Java 5 SE auch ein eigenes GUI-Builder-Tool zur Verfügung steht. Ein gesonderter Build-Server generiert dann je nach Ziel-Betriebssystem eine native App, die direkt auf dem Endgerät installiert oder über den Apple App Store sowie den Google Play Store bereitgestellt werden kann. Der Java-Code wird dabei über den Device-Simulator von Codename One überwacht.

Tabris

Im Gegensatz zu Codename One bekommt man bei Tabris keine für ein bestimmtes Betriebssystem kompilierte native App. Stattdessen entwickelt man eine Server-Anwendung in Java, die auf dem Device durch einen von Tabris mitgelieferten Viewer generisch visualisiert wird. Die komplette Oberflächensteuerung liegt dabei auf dem Server. Tabris erweitert die Ecplise Remote Application Platform (RAP) und lässt die in Java implementierte Fachlogik in einem Servlet ausführen.

Auf dem Endgerät erfolgt lediglich das Rendering des mithilfe des Java SWT erstellten GUI durch den Viewer. Device und Server kommunizieren hierbei via HTTPS

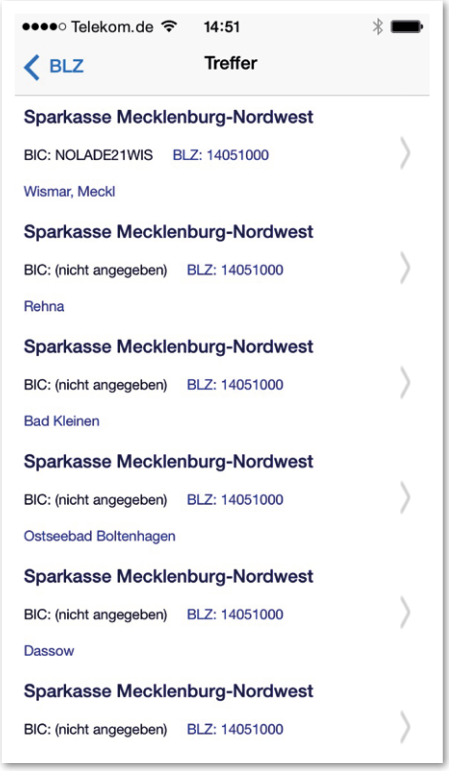


Abbildung 3: Suchergebnis, das dem Suchkriterium „Institutsname=Nord" entspricht

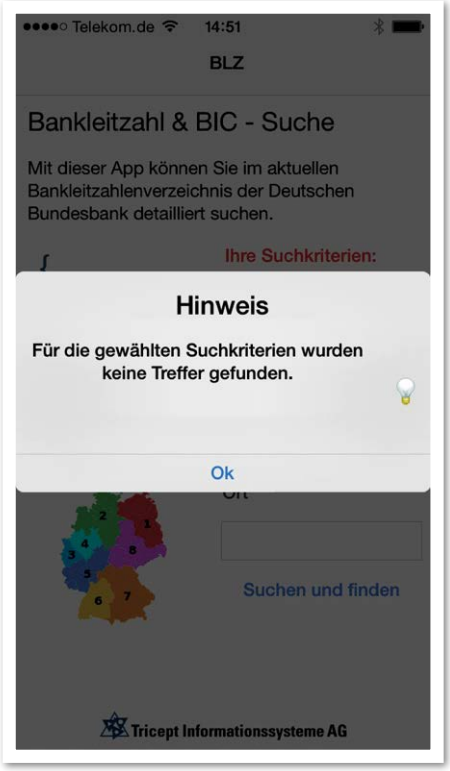


Abbildung 4: Meldung bei fehlgeschlagener Suche

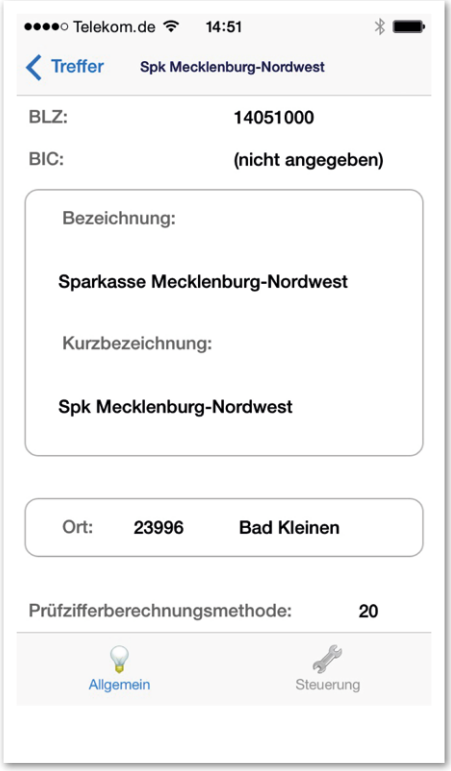


Abbildung 5: Seite „Allgemein"

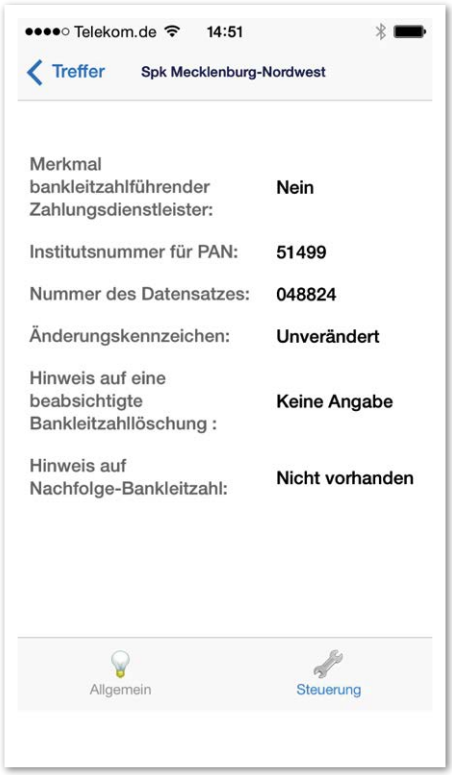


Abbildung 6: Seite „Steuerung"

Abbildung 7: Lage der „BLZ.xml" im Ressourcen-Verzeichnis für iOS

```
// Android, Simulator
com.codename1.io.BufferedInputStream stream =
  new com.codename1.io.BufferedInputStream(
    com.codename1.io.FileSystemStorage.getInstance().
      openInputStream("/BLZ.xml"));

// iOS
java.io.InputStream stream =
  com.codename1.ui.Display.getInstance().
    getResourceAsStream(getClass(), "/BLZ.xml");

java.io.InputStreamReader streamReader =  new java.io.InputStreamReader(stream);

com.codename1.xml.XMLParser  parser = new com.codename1.xml.XMLParser();

this.start(parser.parse(streamReader));      // public void start(com.codename1.xml.Element e) { ... }
```

Abbildung 8: Java-Code für Android und iOS zur Verwendung externer Ressourcen

```
protected void onAction_ButtonSuchen(com.codename1.ui.Component newCompo-
nent, com.codename1.ui.events.ActionEvent newEvent) {
if (this.find_FieldInstitutsname() != null) {
this.setData_institutsname(
this.find_FieldInstitutsname().getText());
}

// Code für BLZ hier analog
  // Code für Ort hier analog

     int treffer = 0;

for (int i = 0; i < this.getInstitutListe().size(); i++) {
de.tricept.codenameone.blz.Institut institut = this.getInstitutListe().
get(i);

            if (this.getData_institutsname() != null) {
if (institut.getBezeichnung().toUpperCase()
.indexOf(this.getData_institutsname()
.toUpperCase()) < 0) {
                        continue;
                }
        }

            // Code für BLZ und Ort auch hier analog zum
// Institutsnamen

treffer++;
} //for

if (treffer == 0) {
// Keine Treffer
            com.codename1.ui.Image image = null;
try {
com.codename1.ui.util.Resources res = com.codename1.ui.util.Resources.
open("/theme.res");
                image = res.getImage("lightbulb_48.png");
} catch (java.io.IOException ex) {
            // do nothing
     }

com.codename1.ui.Dialog.show(
this.convert("Hinweis"),
            this.convert("Für die gewählten Suchkriterien wurden keine
Treffer gefunden."),
com.codename1.ui.Dialog.TYPE_INFO, image, this.convert("Ok"),
null);
} else {
// Liste anzeigen
this.showForm("Liste", null);
}
}
```

Listing 1

und JSON miteinander, wobei der Viewer Benutzer-Eingaben an den Server sendet und von diesem im Gegenzug Anweisungen zum Rendering der Oberflächen-Elemente erhält und diese umsetzt.

Oracle ADF mobile und Oracle MAF

Eine weitere Möglichkeit stellen das Framework „ADF mobile" beziehungsweise dessen Nachfolger Mobile Application Framework (MAF) dar. Nachdem ADF mobile nur für den JDeveloper 11 verfügbar war und lediglich Unterstützung für Java 1.4 ME bot, ist es mit dem MAF nun möglich, neben dem JDeveloper ab Version 12.1.3 auch die Eclipse IDE zusammen mit dem Oracle Enterprise Pack For Eclipse 12.1.3.1 zu verwenden und Unterstützung für Java 8 zu erhalten.

ADF mobile und MAF unterstützen sowohl Browser-basierte Anwendungen als auch die Entwicklung von Apps, bei denen die Fachlogik in einer eigenen mitinstallierten JVM direkt auf dem Endgerät ausgeführt wird. Die Daten werden zur Visualisierung an die Darstellungsschicht übermittelt, die dann das Rendering mithilfe von HTML5 und JavaScript übernimmt. Die einzelnen Anwendungskomponenten lassen sich dabei auf unterschiedliche Weise realisieren:

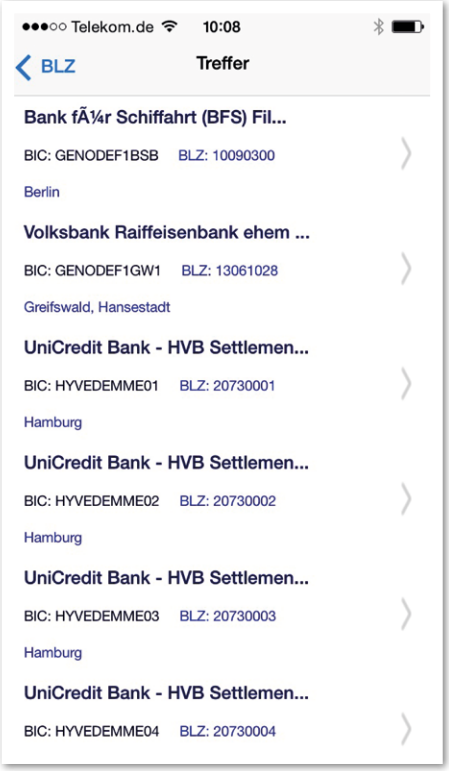


Abbildung 9: Fehlerhafte Darstellung der Umlaute im Namen des ersten Such-Ergebnisses

- Deklarativ unter Verwendung von HTML und JavaScript
- Lokales HTML
- Remote URL (Server-HTML)

Die App „Bankleitzahlensuche" in Java

Um ein Gefühl für die App-Entwicklung mit Java zu bekommen, hat der Autor als Beispiel eine Anwendung mit Codename „One" zur Suche von Bankdaten entwickelt. Die Entscheidung für Codename „One" als Entwicklungswerkzeug war hierbei hauptsächlich dadurch bedingt, dass sich uns hier die Möglichkeit bot, sowohl GUI als auch Fachlogik wie gewünscht komplett in Java zu entwickeln. Zum Installationsumfang der App gehört eine aktuelle Bankleitzahlen-Datei der Deutschen Bundesbank im XML-Format, die die Bank-Informationen zur Suche bereitstellt. *Abbildung 1 und 2* zeigen den Startbildschirm der App in der Codename-One-IDE sowie einen formatierten Auszug aus der BLZ-Datei mit den in der App angezeigten Informationen.

Die Funktionsweise der App ist denkbar einfach und intuitiv. Abhängig von den eingegebenen Such-Parametern liefert die Suche ein für den Benutzer aufbereitetes Ergebnis zurück *(siehe Abbildung 3)*, das in Form einer Übersicht angezeigt wird.

```
protected boolean onMainCommand1() {
    this.formSettings =
    (com.codename1.ui.Form) createContainer(fetchResourceFile(), "Settings");

    com.codename1.ui.Command[] cmds =
    new com.codename1.ui.Command[] {
      // Die Zeichenkette "Übernehmen" wird falsch dargestellt.
      new com.codename1.ui.Command( ("Übernehmen"),
        userclasses.Const.COMMAND_ID_OK),
      new com.codename1.ui.Command( ("Abbrechen"),
        userclasses.Const.COMMAND_ID_CANCEL) };

    formSettings.setDefaultCommand(cmds[0]);
    ...
}
```

Abbildung 10: Umlaute korrekt (grün) beziehungsweise fehlerhaft dargestellt (rot)

Schlägt die Suche fehl, wird dem Benutzer eine entsprechende Meldung ausgegeben *(siehe Abbildung 4)*.

Nach Auswahl eines Suchergebnisses wird eine Seite mit allgemeinen Angaben zum gewählten Kreditinstitut angezeigt. Eine weitere Seite enthält Detail-Informationen zur Steuerung des Zahlungsverkehrs (siehe *Abbildung 5 und 6)*. Hierbei wandelt die Fachlogik der App die in der Datei enthaltenen und teilweise kryptischen Werte in für den Anwender lesbaren Klartext um.

Der Java-Code

Listing 1 zeigt den Java-Code, der die Suche durchführt. Dabei wird jeder Eintrag der XML-Datei auf jedes angegebene Suchkriterium (Name, BLZ, Ort) geprüft und dem Suchergebnis nur im Falle einer entsprechenden Übereinstimmung hinzugefügt. Bei einer erfolglosen Suche wird die Nachricht aus *Abbildung 4* erzeugt und angezeigt.

Auffälligkeiten während der Entwicklung

Wie zu erwarten, läuft bei der Entwicklung natürlich nicht immer alles sofort nach Plan. Abhängig von der Ziel-Plattform muss an verschiedenen Stellen eingegriffen werden, um externe Ressourcen wie etwa die XML-Datei in unserem Beispiel einzubinden. Hier sind manuelle Anpassungen im Java-Code notwendig, um den Unterschieden zwischen den Betriebssystemen Android und iOS Rechnung zu tragen.

Während die Datei für Android an beliebiger Stelle im Dateisystem liegen kann, um mit Java eingebunden und verwendet zu werden, so ist für eine iOS-App zu beachten, dass verschachtelte Verzeichnis-Strukturen hier nicht zulässig sind. In diesem Fall müssen alle Ressourcen direkt im Root-Verzeichnis der App liegen, damit sie für die App verfügbar sind *(siehe Abbildung 7)*. *Abbildung 8* zeigt den unterschiedlichen Java-Code, mit

```
// Wird vom XML-Parser aufgerufen
private java.lang.String getValue(com.codename1.xml.Element element) {
    return this.convert(element.getText());
}


private java.lang.String convert(java.lang.String text) {
    try {
        // Konvertierung in Unicode
        return new java.lang.String(text.getBytes(), "UTF-8");
    } catch (java.io.UnsupportedEncodingException ex) {
        return text;
    }
}
```

Abbildung 11: Konvertierungsroutine für den XML-Parser

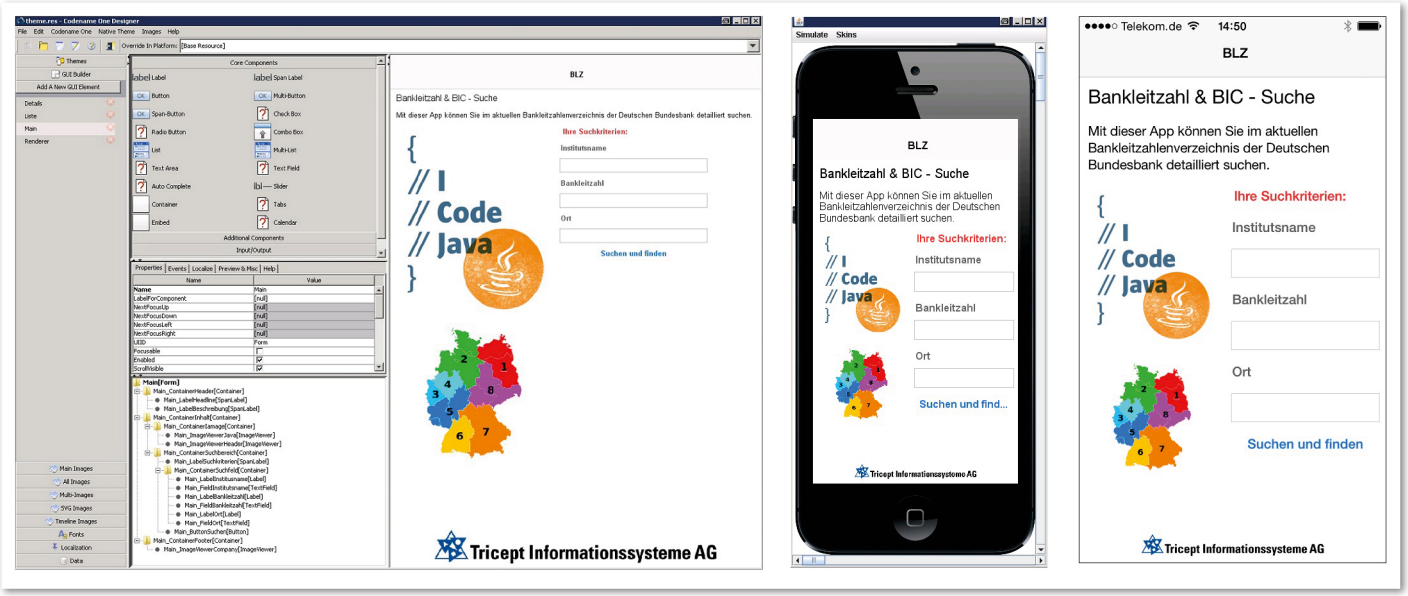


Abbildung 12: App im GUI-Builder, im Simulator und auf dem Gerät

dem die Bankleitzahlen-Datei unter Android beziehungsweise iOS angesprochen wird.

Ein weiteres Problem bei der Entwicklung ist das unterschiedliche Verhalten im Umgang mit Umlauten. Während eine iOS-App in Java-Strings verwendete Umlaute und solche, die über den GUI-Editor direkt in die Properties geschrieben werden, korrekt anzeigt, werden Umlaute aus der XML-Datei fehlerhaft dargestellt *(siehe Abbildung 9 und 10)*.

Abhilfe schafft hier die explizite Konvertierung der Texte in UTF-8. Die in *Abbildung 11* vorgestellte Konvertierungsroutine wird auch in *Listing 1* verwendet, um am Ende den Text des Nachrichtenfensters korrekt auszugeben. Neben den gerade erwähnten Punkten ist es genauso unerlässlich, während des gesamten Entwicklungsprozesses immer wieder das „Look&Feel" der App zu überprüfen. So variiert das Erscheinungsbild mitunter recht stark, wenn man sich die App in der Entwicklungsumgebung, im Simulator oder als fertige Version auf dem physischen Gerät selbst anschaut. Auch zwischen Android und iOS sind Unterschiede zu beobachten und entsprechend zu berücksichtigen.

Auftretende Abweichungen zwischen dem gewünschten Endlayout und der Ansicht im GUI-Builder darf man nicht aus dem Auge verlieren und muss diese immer wieder manuell in zum Teil recht anstrengender Kleinarbeit korrigieren. *Abbildung 12* zeigt, wie sich das Layout der Beispiel-Anwendung von Fall zu Fall unterscheidet.

Fazit

Abschließend lässt sich sagen, dass es bereits gute Ansätze und Möglichkeiten gibt, mobile Anwendungen in Java zu entwickeln. Man sollte dabei aber immer im Hinterkopf behalten, dass man immer noch Unzulänglichkeiten und Hindernisse in Kauf nehmen muss. Da der gesamte Bereich einem stetigen Wandel und schnell fortschreitender Weiterentwicklung unterworfen ist, verbessern sich die Werkzeuge allerdings permanent. Dies stimmt zuversichtlich im Hinblick darauf, dass auch künftig der Frustrationsfaktor sinken und der Motivationsfaktor steigen wird und die bequeme Entwicklung mobiler Anwendungen mit Java sich hier einen festen Platz erarbeitet. Bis es jedoch soweit ist, heißt es weiterhin: Write once – test everywhere!

Axel Marx
axel.marx@tricept.de

Axel Marx studierte Mathematik und Informatik an der TU Braunschweig und ist seit mehr als fünfzehn Jahren in der Anwendungsentwicklung tätig. Dabei sammelte er Erfahrungen auf dem Gebiet klassischer Client-Server-Systeme und in der Koordination und Betreuung von Software-Migrationsprojekten im In- und Ausland. Zurzeit betreut er für die Tricept Informationssysteme AG Kunden im Java-Umfeld, hauptsächlich aus den Bereichen „Banking" und „Automotive". Seine besonderen Interessen liegen in der Datenbank- und Mobile-Entwicklung.

http://ja.ijug.eu/15/3/3

Java für Mac: Ärger über Adware und wie man sie umgeht

Laut Berichten von heise und zdnet reichert Oracle Java für den Mac inzwischen mit einer Adware an. User und Community sind verärgert. Allerdings kann man dies am Oracle JDK 8 Update 40 umgehen. Dafür muss der „Java Control Panel" aufgemacht werden.

Unter „Advanced" und „Miscellaneous" ist die Einstellung „Suppress sponsor offers when installing or updating Java" zu finden. Damit kann mögliche Werbung reduziert werden.