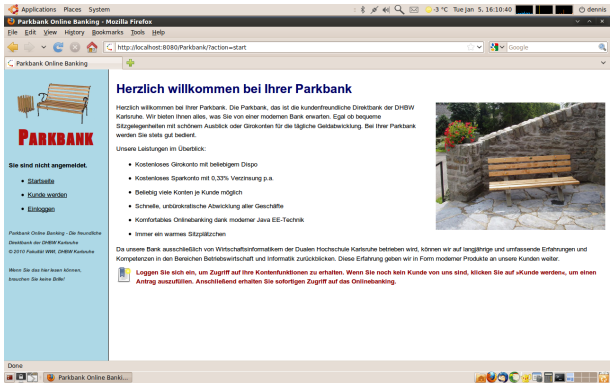
— Gruppenarbeit 4 (P): Parkbank —

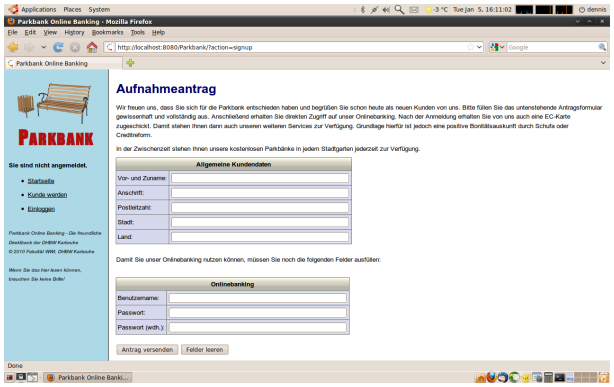
## **Aufgabe 1:** Alle **(Einleitung und Hinweise)**

Das Kultusministerium des Landes Baden-Württemberg hat beschlossen, dass die Duale Hochschule in Kooperation mit der Landesbank Baden-Württemberg und der Bad Bank of America ein eigenes Geldinstitut für die Mitglieder der Hochschule eröffnen wird. Mit einem großen Festakt wurde die Bank auf den Namen Parkbank getauft und die Bankleitzahl 660123987 vergeben. Nun soll am Standort Karlsruhe ein Pilotprojekt laufen, so dass die Bank als Onlinebank ihre ersten Geschäfte aufnehmen kann. Dabei wurde folgender Leistungsumfang für das Onlinebanking vereinbart:

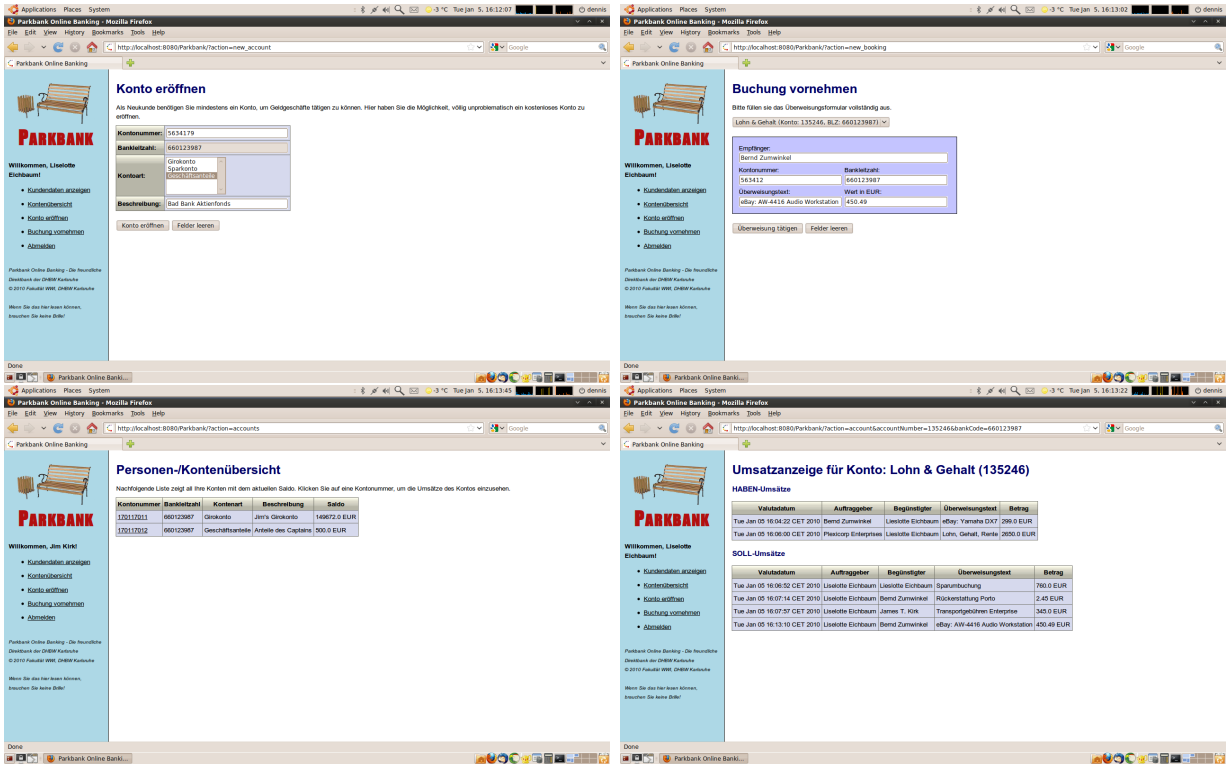
- Neukunden können sich online selbstständig registrieren
- Jeder Kunde kann eine beliebige Anzahl von Konten online eröffnen
- Zwischen den Konten der Bank können beliebige Geldtransfers durchgeführt werden
- Kunden können online ihre Kundendaten einsehen
- Außerdem steht eine Kontenübersicht mit einer Auflistung aller Konten zur Verfügung
- Für jedes Konto kann eine Liste der durchgeführten Transaktionen angezeigt werden

Weil die Architekturverantwortlichen damit rechnen, dass die Onlineanwendung im Laufe der Zeit an vielen Funktionen hinzugewinnen wird, wurde schon früh entschieden, dass die Anwendung auf Basis der neusten Java Enterprise Edition umgesetzt werden soll. Die Webanwendung wird mit einfachen Servlets und Java Server Pages realisiert, weil sich die Anwendung somit später auf einen separaten Webcontainer auslagern lässt[1]. Die dahinterliegenden Funktionen sollen dann mit Enterprise Java Beans und der Java Persistence API umgesetzt werden, wobei die grundlegenden Dienste auch per Webservice verfügbar gemacht werden sollen.




[1] Aussage des amerikansichen CIO: *As a stock exchange we don't like crashes.*




**Vorbereitung:** Laden Sie sich von Moodle die `Parkbank`-Aufgabe herunter. Dieses Archiv beinhaltet in den zwei Ordnern namens `Server` und und `WS-Client` die vorprogrammierten Anwendungen, wobei der Ordner `WS-Client` zunächst leer ist. Nachfolgend sehen Sie die komplette Verzeichnisstruktur der Aufgabe mit allen Dateien und Unterverzeichnissen:

```
.
|-- Server
|   |-- build.xml
|   |-- lib_compile
|   |   |-- javaee-16.jar
|   |   `-- servlet-api.jar
|   |-- lib_server
|   `-- sources
|       |-- config
|       |   `-- classes
|       |       `-- META-INF
|       |           |-- beans.xml
|       |           `-- persistence.xml
|       |-- java
|       |   `-- de
|       |       `-- dhbw
|       |           `-- parkbank
|       |               |-- AccountController.java
|       |               |-- CharsetFilter.java
|       |               |-- ejb
|       |               |   |-- AccountBean.java
```

```
|       |               |   |-- AccountLookupException.java
|       |               |   |-- AuthenticationException.java
|       |               |   |-- CreateAccountException.java
|       |               |   |-- CreateBookingException.java
|       |               |   |-- LoginBean.java
|       |               |   `-- SignUpException.java
|       |               |-- FrontServlet.java
|       |               |-- jpa
|       |               |   |-- Account.java
|       |               |   |-- AccountPK.java
|       |               |   |-- AccountType.java
|       |               |   |-- Booking.java
|       |               |   |-- Credentials.java
|       |               |   `-- User.java
|       |               `-- LoginController.java
|       |-- manifest
|       `-- web
|           |-- img
|           |   |-- dialog-error.png
|           |   |-- dialog-information.png
|           |   |-- home.png
|           |   |-- log-out.png
|           |   |-- mauer-bank.jpg
|           |   |-- new.png
|           |   |-- parkbank.png
|           |   `-- tbl_head.png
|           |-- inc_account.jsp
|           |-- inc_account.jsp.INAKTIV
|           |-- inc_accounts.jsp
|           |-- inc_accounts.jsp.INAKTIV
|           |-- inc_default.jsp
|           |-- inc_login.jsp
|           |-- inc_logout.jsp
|           |-- inc_new_account.jsp
|           |-- inc_new_booking.jsp
|           |-- inc_new_booking.jsp.INAKTIV
|           |-- inc_signup.jsp
|           |-- inc_userdata.jsp
|           |-- inc_userdata.jsp.INAKTIV
|           |-- template.jsp
|           `-- template.jsp.INAKTIV
`-- WS-Client
```

Die Verzeichnisse `lib_compile` und `lib_server` beinhalten verschiedene JAR-Bibliotheken, welche zum Compilieren und zum Ausführen der Anwendung benötigt werden. Die eigentlichen Quellcodes der Anwendung finden Sie im Verzeichnis `sources`. Hinzu kommen die Verzeichnisse `build` und

`dist`, welche von Ant automatisch erzeugt werden. Unterhalb von `dist` finden Sie die von Ant erzeugten Archivdateien.
Das Quellcodeverzeichnis gliedert sich in die drei Unterverzeichnisse `config`, `java` und `web`. In `config` befinden sich alle Konfigurationsdateien, welche für das Deployment der Anwendung benötigt werden. `java` beinhaltet die Javaquellcodes der enthaltenen Enterprise Java Beans und Persistent Entities, wobei sich diese in unterschiedlichen Paketen befinden. Der Ordner `web` beinhaltet alle Webdateien wie JSP- oder HTML-Dateien, die unverändert in das erzeugte Webarchiv aufgenommen werden.

**Anwendung kompilieren:** Um die Anwendung kompilieren zu können, benötigen Sie Apache Ant[2] auf Ihrem Computer. Es steuert den Kompiliervorgang und die Erzeugung des zu deployenden Webarchives der Anwendung. Wenn Sie Ant installieren, achten Sie darauf, dass die ausführbare Datei `ant` in Ihrer PATH-Umgebungsvariable ist, so dass Sie das Kommando aus allen Verzeichnissen heraus ausführen können. Die möglichen Aufrufe von `ant` können Sie dann mit folgendem Befehl sehen:

```
$ ant -projecthelp
Buildfile: Aufgaben/pr4-Parkbank/Server/build.xml

        Dies ist die Parkbank-Aufgabe zu EJB und JPA.

Main targets:

 build-war  Erzeugt ein deploybares Webarchiv
 clean      Löscht alle zuvor kompilierten Klassendateien und Webarchive
 compile    Compiliert alle Javaklassen
Default target: build-war
```

Das heißt, würden Sie nur den Befehl `ant` ausführen, würde Ant das Ziel `build-war` ausführen und somit die Anwendung kompilieren und ein Webarchiv für den Applikationsserver erzeugen. Wollen Sie ein anderes Ziel ausführen, müssen Sie dieses beim Aufruf mitgeben. Zum Beispiel: `ant clean`
Nachdem Sie die Anwendung kompiliert haben, finden Sie im Verzeichnis `dist` eine Datei namens `dhbw-parkbank.war`. Diese müssen Sie in einem Applikationsserver wie JBoss 6 deployen.

**Deployment in einem Applikationsserver**: Falls noch nicht geschehen, besuchen Sie die Seite `http://www.jboss.org/jbossas/downloads.html` und laden sich die neuste Final-Version des JBoss Applikationsserver herunter. Es handelt sich dabei um eine ca. 180 MiB große ZIP-Datei, die Sie einfach irgendwo auf Ihrem Rechner entpacken können.
Bevor Sie den Applikationsserver das erste mal starten, müssen Sie erst die Konfiguration anpassen, damit der Webcontainer unicodefähig wird. Öffnen Sie daher die Datei `jbossweb.sar` aus dem Verzeichnis `/server/default/deploy/jbossweb.sar/` in einem Texteditor. Ungefähr in Zeile 12 finden Sie ein `<Connector protocol="HTTP/1.1" .../>`-Element, das Sie durch folgenden Inhalt ersetzen müssen:

```
<Connector protocol="HTTP/1.1" port="8080" address="${jboss.bind.address}"
           connectionTimeout="20000" redirectPort="8443"
           useBodyEncodingForURI="true"/>
```

[2] http://ant.apache.org

Anschließend wechseln Sie in das Wurzelverzeichnis von JBoss und führen folgende Befehle aus, um den Server zu starten.

```
WINDOWS:
    $ cd bin
    $ run

LINUX:
    $ cd bin
    $ ./run.sh
```

Warten Sie nun, bis der Server hochgefahren ist und Sie am Ende des Protokolls folgenden Eintrag sehen: `Started in 1m:5s:666ms`
Damit steht der Server zur Verfügung und die Aufgabe kann deployed werden. Hierfür kopieren Sie einfach die Datei `dhbw-parkbank.war` aus dem `dist`-Verzeichnis der Aufgabe in folgendes JBoss-Verzeichnis: `server/default/deploy`. Im Kommandozeilenfenster von JBoss sehen Sie daraufhin mehrere Protokolleinträge vorbei ziehen. Dort sollte kein Eintrag mit `[ERROR]` beginnen.

Unter folgenden Adressen können Sie die Anwendung aufrufen:
`http://localhost:8080/dhbw-parkbank/index`
`http://localhost:8080/dhbw-parkbank/LoginBean?wsdl`
`http://localhost:8080/dhbw-parkbank/AccountBean?wsdl`

## **Aufgabe 2:** Alle **(Persistent Entities)**

Vervollständigen Sie als erstes die Implementierung der Klasse `User`, welche gleichzeitig einen Kunden und einen Systembenutzer modelliert. Führen Sie folgende Änderungen an der Klasse durch:

1. Machen Sie die Klasse zu einer persistenten Klasse.
2. Legen Sie fest, dass die zugrunde liegende Tabelle `USER_TABLE` heißen soll, da das Wort `USER` in einigen DBMS ein geschütztes Schlüsselwort ist.
3. Definieren Sie die folgenden Attribute mitsamt Setter- und Getter-Methoden:
    - `String username`
    - `String password`
    - `String fullName`
    - `String street`
    - `int zip`
    - `String city`
    - `String country`
    - `List<Account> accounts`
4. Machen Sie das Feld `username` zum Schlüsselfeld.

5. Machen Sie aus `accounts` eine unidirektionale 1:n-Beziehung. Achten Sie dabei darauf, der Annotation den Wert `fetch=FetchType.EAGER` mitzugeben, um spätere Probleme zu vermeiden.

6. Stellen Sie sicher, dass jedes Attribut mit einem sinnvollen Startwert vorbelegt wird. Für Stringfelder ist dies ein leerer String, für Ganzzahlen ist es `-1`, die Liste sollten Sie als `ArrayList<Account>` vorbelegen.

7. Fügen Sie der Klasse noch eine Methode namens `public Credentials getCredentials()` hinzu. Diese Methode soll mit `@Transient` ausgezeichnet sein und ein neues `Credentials`-Objekt mit Benutzername und Passwort des Benutzers zurückgeben.

Fügen Sie der Klasse außerdem eine benannte Abfrage namens `findUserByUsernameAndPassword` hinzu. Diese wird von der `LoginBean` verwendet, um festzustellen, ob ein Benutzer sich einloggen kann. Setzen Sie hierfür folgende Annotation vor die Klasse:

```
@NamedQueries({
  @NamedQuery(
    name  = "findUserByUsernameAndPassword",
    query = "SELECT u FROM User u WHERE u.username = :username"
          + "AND u.password = :password")
})
```

Aktivieren Sie nun den Quellcode der beiden Methoden `authenticate()` und `signUp()` der Klasse `LoginBean`, indem Sie die jeweils gekennzeichneten Kommentarzeilen entfernen. Machen Sie dasselbe mit den Methoden `actionSignup()` und `actionNewBooking()` des `FrontServlet`. Lassen Sie jedoch den Quellcode von `actionNewAccount()` auskommentiert, bis Sie die Kontoklasse umgesetzt haben.
Aktivieren Sie nun folgende Java Server Pages, indem Sie sie im Verzeichnis `/Server/sources/web` löschen und durch die gleichnamigen Dateien mit der Endung `*.INAKTIV` ersetzen.

- `template.jsp`
- `inc_userdata.jsp`

Probieren Sie die Anwendung aus! Versuchen Sie, einen neuen Benutzer anzulegen und sich mit diesem Benutzer einzuloggen. Probieren Sie dabei auch die Menüfunktionen für eingeloggte Benutzer aus. Viele Funktionen funktionieren schon. Da Sie aber noch keine Konten anlegen können, stehen Ihnen einige Funktionen noch nicht zur Verfügung.

Programmieren Sie nun die Klasse `Account` aus, indem Sie folgende Änderungen an ihr vornehmen:

1. Legen Sie folgende Attribute mitsamt Settern und Gettern an:
   - `AccountType type`
   - `String description`
   - `double balance`

2. Belegen Sie alle Attribute sinnvoll vor. Die Integer-Attribute belegen Sie mit `-1` vor, die String-Attribite mit einem leeren String und die Double-Attribute mit `0.0`. Der ursprüngliche Kontentyp soll `AccountType.GIRO` sein und die Listen belegen Sie mit je einer leeren `ArrayList` vor.

3. Verwenden Sie die Annotation `@Enumerated`, um das Attribut `type` als Aufzählungswert zu kennzeichnen.

Aktivieren Sie nun auch den Quellcode der Methode `actionNewAccount()` des `FrontServlet`, indem Sie die übrigen, gekennzeichneten Kommentare entfernen. Aktivieren Sie ebenso folgende Java Server Pages, indem Sie sie im Verzeichnis `/Server/sources/web` löschen und durch die gleichnamigen Dateien mit der Endung `*.INAKTIV` ersetzen.

- `inc_account.jsp`
- `inc_new_booking.jsp`

Sie haben es fast geschafft! Es fehlt nurnoch die Klasse `Booking`, welche eine Buchung mit zwei beteiligten Konten darstellt. Nehmen Sie folgende Änderungen an der Klasse vor:

1. Entfernen Sie das bisherige Schlüsselfeld `int id`, indem Sie alle Zeilen zwischen den entsprechenden Kommentaren löschen.

2. Legen Sie stattdessen folgende Attribute mitsamt Settern und Gettern an:

   - `int id`
   - `String sender`
   - `String receiver`
   - `double value`
   - `String description`
   - `Date timestamp`
   - `AccountPK accountFrom`
   - `AccountPK accountTo`

3. Das Feld `id` ist das Schlüsselfeld der Entität. Der Wert soll von Java automatisch generiert werden.

4. Das Feld `timestamp` soll ein Zeitfeld sein, das sowohl Datum als auch Uhrzeit beinhaltet.

5. Benutzen Sie folgende Annotationen, um die beiden Felder `accountFrom` und `accountTo` zu definieren:

```
@AttributeOverrides({
    @AttributeOverride(name="bankCode",
                       column=@Column(name="FROM_BANK_CODE")),
    @AttributeOverride(name="accountNumber",
                       column=@Column(name="FROM_ACCOUNT_NUMBER"))
})
@Embedded
```

```
...

@AttributeOverrides({
    @AttributeOverride(name="bankCode",
                       column=@Column(name="TO_BANK_CODE")),
    @AttributeOverride(name="accountNumber",
                       column=@Column(name="TO_ACCOUNT_NUMBER"))
})
@Embedded
...
```

## **Aufgabe 3:** Alle **(Enterprise Java Beans)**

Wenn Sie die Anwendung nun compilieren und deployen, stehen Ihnen immer noch nicht mehr Funktionen zur Verfügung. Das liegt daran, dass Sie die Methode `createAccount()` der Klasse `AccountBean` zur Anlage eines neuen Kontos noch entwickeln müssen. Programmieren Sie folgenden Algorithmus:

1. Zunächst sollen die Daten verprobt werden. Wenn eine der folgenden Bedingungen verletzt ist, soll eine `CreateAccountException` mit einer Fehlermeldung geworfen werden:

   `throw new CreateAccountException("Fehlermeldung ...")`

   - Der Parameter `newAccount` darf nicht `null` sein
   - Die Kontonummer darf nicht kleiner 1000 sein
   - Die Beschreibung darf nicht leer (Methode `isEmpty()`) sein

2. Wenn keine Bankleitzahl vorgegeben wurde, setzen Sie diese auf 660123987.

3. Prüfen Sie, ob bereits ein Konto mit derselben Kontonummer und Bankleitzahl existiert. Hierfür erzeugen Sie ein Objekt vom Typ `AccountPK` und übergeben ihm die zu prüfenden Schlüsselwerte. Nutzen Sie dann die Methoe `find()` des `EntityManager`, um nach einer persistenten Instanz zu suchen. Wenn die Rückgabe nicht `null` ist, werfen Sie eine Exception mit einer Fehlermeldung.

4. Vergessen Sie nicht, das Konto auch dem Benutzer zuzuordnen. Verwenden Sie hierfür die Variable `user` und nehmen Sie das Konto in die Liste `accounts` des Benutzers auf. Speichern Sie diese Änderung durch Aufrufen von `this.em.merge(user)`. Für das neue Konto müssen Sie hingegeen die Methode `this.em.persist(newAccount)` aufrufen.

Aktivieren Sie nun auch die letzte Java Server Page namens `inc_account.jsp`, indem Sie die Originaldatei löschen und durch die Datei `inc_account.jsp.INAKTIV` ersetzen. Deployen Sie die Anwendung anschließend ein weiteres mal. Jetzt sollten Sie in der Lage sein, ein neues Konto anzulegen und dieses in der Kontenliste anzuzeigen. Sie können aber noch keine Überweisungen tätigen und somit auch nicht die Liste der Überweisungen anzeigen.

Damit die in der Anwendung getätigten Überweisungen nun auch ausgeführt werden können, müssen Sie noch die Methode `createNewBooking()` der Klasse `AccountBean` vervollständigen. Aktivieren

Sie zunächst den bereits vorhandenen Quellcode, indem Sie die entsprechend gekennzeichneten Kommentare entfernen. Anschließend ergänzen Sie den fehlenden Code gegen Ende der Methode wie folgt:

1. Erwerben Sie eine Schreibsperre auf `srcAccount` und `dstAccount`:
   ```
   this.em.lock(srcAccount, LockModeType.WRITE);
   this.em.lock(dstAccount, LockModeType.WRITE);
   ```
2. Erzeugen Sie ein neues Objekt vom Typ `Booking` und füllen Sie es mit den Überweisungsdaten.
3. Subtrahieren Sie den Überweisungsbetrag vom Saldo des Quellkontos.
4. Addieren Sie den Überweisungsbetrag zum Saldo des Zielkontos.
5. Fügen Sie die neue Buchung zu den `negativeBookings` des Quellkontos und den `positiveBookings` des Zielkontos hinzu.
6. Speichern Sie die neue Überweisung durch `em.persist()`.
7. Speichern Sie die beteiligten Konten durch `em.merge()`.

Das war's! Sie haben die Onlinebankinganwendung gerade noch fristgerecht fertiggestellt. Deployen Sie die Anwendung und probieren Sie die verbleibenden Funktionen aus. Legen Sie dabei auch verschiedene Kunden mit mehreren Konten an. Machen Sie mehrere Überweisungen zwischen den einzelnen Konten und schauen Sie sich die Kontenübersicht und die jeweiligen Kontendetails dazu an.

## **Aufgabe 4:** Alle **(Webservices und Client)**

Nachdem die Onlineanwendung nun funktioniert, wollen Sie Ihren Anwendern natürlich auch die Möglichkeit geben, andere Programme zur Erledigung ihrer Geldgeschäfte zu nutzen. Da Sie aber nicht wissen können, welche Anwendungen hier in Frage kommen, müssen Sie eine Schnittstelle anbieten, die mit möglichst vielen Programmiersprachen auf möglichst vielen Betriebssystemen angesprochen werden kann. Das heißt, Sie müssen Ihren Enterprise Java Beans nun die Möglichkeit verleihen, per Webservice aufgerufen zu werden.

Nehmen Sie folgende Änderungen an der `LoginBean` und der `AccountBean` vor:

1. Erweitern Sie die Klassen mit allen benötigten Annotationen so, dass sie gültige Webservice-Implementierungen darstellen.
2. Erweitern Sie auch alle enthaltenen Methoden mit den entsprechenden Annotationen, um diese per Webservice aufrufbar zu machen. Dabei beinhaltet die Klasse `LoginBean` zwei Methoden mit dem Namen `authenticate`. Da dies in Webservices nicht erlaubt ist, müssen Sie eine der beiden Methoden mit der Annotation `@WebMethod(exclude=true)` ausstatten.
3. Sorgen Sie dafür, dass der Rückgabewert der Methode `authenticate` in der Klasse `LoginBean` unter dem Namen `user` zurückgegeben wird.
4. Der Rückgabewert der Methode `lookup` in der Klasse `AccountBean` soll entsprechend `account` heißen.

5. Sämtliche Methodenparameter sollen im Webservice dieselben Namen besitzen wie in den Javaklassen.

Deployen Sie nun die angepasste Anwendung und rufen Sie folgende Adressen in Ihrem Browser auf, um die `WSDL`-Beschreibungen der Webservices anzuschauen. Alternativ können Sie die `WSDL`-Beschreibungen auch in ein Programm wie SoapUI[3] einlesen, um die Webservices auszuprobieren.

```
http://localhost:8080/dhbw-parkbank/LoginBean?wsdl
http://localhost:8080/dhbw-parkbank/AccountBean?wsdl
```

Zum Schluss schreiben Sie noch ein kleines Clientprogramm, welches die neuen Webservices aufrufen kann. Der entsprechende Ordner `WS-Client` ist in der Aufgabe bereits angelegt. Das Programm soll nicht alle Operationen der Webservices verwenden. Stattdessen soll es nur folgende Funktionen bieten: Beim Programmstart muss man seinen Benutzernamen und sein Passwort eingeben, um sich am Server zu authentifizieren. Anschließend kann man in einem Menü auswählen, ob man sich eine Übersicht der eigenen Konten anzeigen lassen will, oder ob man eine neue Überweisung beauftragen will. Ein typischer Ablauf würde daher so aussehen:

```
Parkbank Terminalclient
=======================

Benutzer: susi
Passwort: sorglos

Hauptmenü
---------

[K] Kontenübersicht
[U] Überweisung
[E] Ende

Auswahl: K

Personen/Kontenübersicht
------------------------

Kontonummer:  123456
Bankleitzahl: 660123987
Kontenart:    Girokonto
Bezeichnung:  Bargeld lacht
Saldo:        5113.0 EUR
```

[3] http://www.soapui.org/

```
Hauptmenü
---------

[K] Kontenübersicht
[U] Überweisung
[E] Ende

Auswahl: U

Neue Überweisung
----------------

Eigene Kontonummer:  123456
Name des Empfängers: Rainer Krawall
Fremde Kontonummer:  4534731
Fremde Bankleitzahl: 660123987
Betrag in EURO:      42.84
Überweisungstext:    Vielen Danke für die Blumen!
...
```

Beobachten Sie auch, was in der Onlineanwendung passiert, wenn Sie mit dem Webservice-Client eine Überweisung tätigen.