Dokumenation

E-Bank-System

Praktikumsaufgabe - Objektorientierte Softwaretechnik

Valerius Weigandt - 8200844; Carl Tuemmler - 8200841
9/25/2011

INHALTSVERZEICHNIS

INSTALLATIONSSCHRITTE

Der Quellcode sowie eine deployable WAR Datei ist unter `https://github.com/carlt/OOSWT-Bank-Server` zu finden.

Eine Installation dauert nicht sehr lange; es muss lediglich eine Datenbank installiert werden, und die Anwendung auf einem Server deployed werden. Es wird davon ausgegangen dass der Server lokal (`localhost`) läuft. Weiterhin muss der Server Java Server Faces 2.0 unterstützen.

1. Die Datenbank "Berkeley DB XML" ist notwendig. Diese ist kostenlos bei Oracle (Windows Installer oder zip) verfügbar.
2. Die Datei `E-Bank-System.war` muss auf dem Server deployed werden. Bei einem Glassfish Server ist dies über die Adminkonsole (standardmäßig `http://localhost:4848`) möglich.
3. Beim deployen müssen die `db.jar` und die `dbxml.jar` aus dem Ordner `<Berkeley DB XML Installationsverzeichnis>/jars` ebenso mit angegeben werden. Alternativ kann man die beiden Dateien in das Verzeichnis `<Glassfish Installationsverzeichis>/glassfish/domains/domain1/lib` kopieren.
4. Die Anwendung ist nun unter `http://localhost:8080/E-Bank-System` erreichbar.

WICHTIG: Da das Webinterface CSS3 verwendet, sollte Firefox / Chrome / Safari / Opera als Browser verwendet werden.

DER LOGIN

Auf dieser Webseite erhält der Benutzer die Möglichkeit sich entweder in das System einzuloggen oder sich zu registrieren.

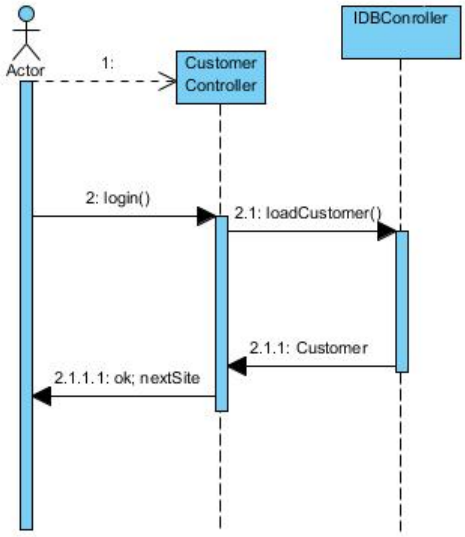
Um sich anzumelden muss der Benutzer die ihm zuvor gegebene Zugangsdaten unter Kundennummer und Passwort eintragen und auf Anmelden klicken. Falls die Zugangsdaten in irgendeiner Art ungültig seine sollten wird der Benutzer gleich darauf hingewiesen und erhält die Möglichkeit sich zu verbessern. Im unteren rechten Teil gibt es für den Benutzer die Möglichkeit sich zu registrieren.

Um zum Verwaltungsbereich zu kommen muss als Kontonummer "`admin`" und als Passwort ebenfalls "`admin`" angegeben werden.

ABARBEITUNG EINER ANMELDUNG

Die Klasse `CustomerContoller` wir für jeden Kunden neu instanziiert und kümmert sich die Verwaltung des aktuellen Kunden. Also um die Benutzerdaten und den Login Status.

Im Sequenzdiagramm 1 ist die Abarbeitung einer Login Anfrage dargestellt. Dabei gibt der Kunde im Login Dialog seine Anmeldedaten ein und ruft mit dem Anmelden Button die Funktion `login()` auf. Hier im Schritt 2 dargestellt. Im Schritt 2.1 holt der `CustomerContoller` über den `IDBContoller` die Benutzerdaten aus der Datenbank. Dabei wird in Schritt 2.1.1 der passende Kunde an den `CustomerContoller` zurückgeliefert, fass einer existiert. Wenn alles in Ordnung ist leitet der `CustomerContoller` den Kunden auf die nächste Seite weiter. Falls Fehler beim Login Festgestellt wurden, verweilt der Kunde weiterhin auf der Login Seite bis er korrekte Anmeldedaten angibt.



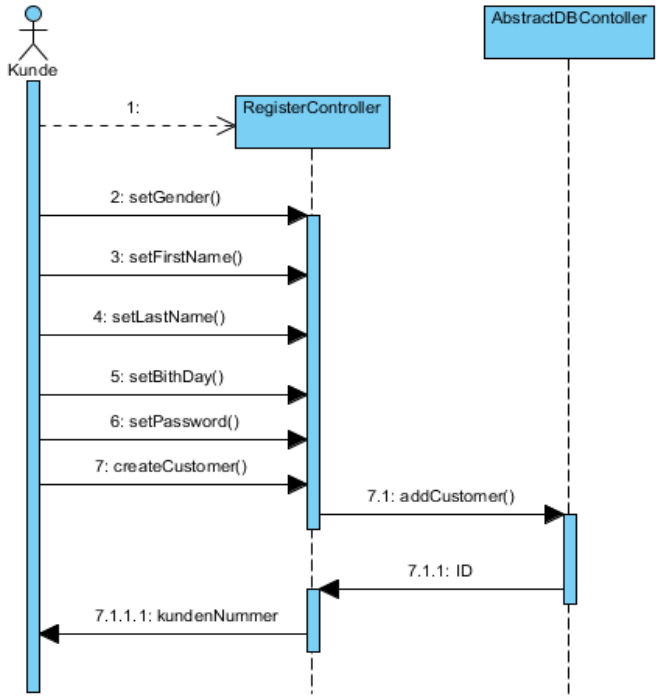
Sequenzdiagramm 1 Anmeldung

DIE REGISTRIERUNG

In dem Registrieren Dialog erhält ein Kunde die Möglichkeit sich zu Registrieren. Dabei werden Daten wie die Anrede, Vorname, Nachname, Geburtstag und ein Passwort abgefragt. Falls alle Angaben zufriedenstellen angegeben wurden kann sich der Kunde über den Registrieren Button am System Anmelden. Bei einer erfolgreichen Anmeldung wird dem Kunden eine Kundennummer angezeigt. Mit der Kundenummer und dem vorher gewählten Password kann sich der Kunde im Login Dialog nun anmelden.

ABARBEITUNG EINER REGISTRIERUNG

Im Sequenzdiagramm 2 wird die Abarbeitung einer Registrierung dargestellt. Dabei wird sofort eine Instanz des `RegisterContoller` erzeugt, sobald ein Kunde den Registrieren Dialog aufruft. Anschließend gibt er in Schritt 2-6 seine Daten ein. Anschließen wird über die `createCustomer()` Funktion des `RegisterControllers` ein neuer Kunde in der Datenbank angelegt. Dies geschieht in 7.1. Als Antwort bekommt der Register-Controller die `ID` des neuen Kunden. Zuletzt wird die `Kundennummer` dem Kunden bekannt gegeben.



Sequenzdiagramm 2 Registrierung

DIE KONTENÜBERSICHT

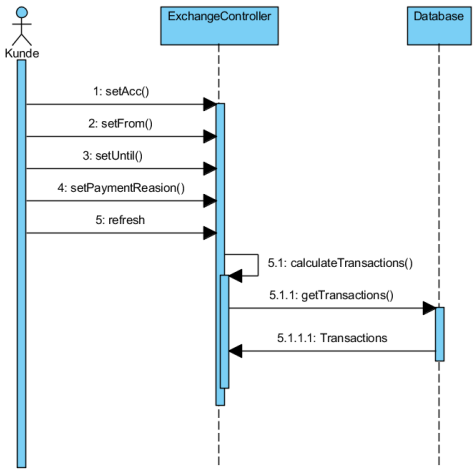
In der Kontenübersicht bekommt der Kunde eine schnelle Übersicht über seine Konten. Dabei wir in Tabellenform die Kontonummer sowie der Kontostand dargestellt. Zusätzlich kann man über die Aktionen schnell auf die Konten zugreifen. Dabei kann man für ein gewünschtes Konto die Umsätze abfragen oder Überweisungen ausführen. Alternativ sind diese Funktionen auch jederzeit über die globale Menu leiste zu erreichen. Zusätzlich bekommt der Kunde die Möglichkeit jederzeit auch ein neues Konto zu eröffnen. Jedoch sollte dieser Schritt stehst bedacht erfolgen, da bereits angelegte Konten nicht mehr gelöscht werden können.

DIE UMSÄTZE

Bei den Umsätzen kann der Kunde in seine Kontobewegungen Einblicken. Dabei kann der Kunde zu einem gewählten Konto noch Kriterien angeben was angezeigt werden soll. Als Kriterium kann man angeben von wann bis wann getätigte Buchungen angezeigt werden soll und man kann ebenso nach einem bestimmten Verwendungszweck filtern. Falls Ergebnisse zur Verfügung stehen werden diese in absteigender Reinfolge angezeigt, also stehen neuere Buchungen weiter oben. Alle Zeitangaben sind in GMT+0.

ABARBEITUNG EINER ANFRAGE

Die Abarbeitung einer nach den gewollten Kriterien gefilterten Anfrage ist in Sequenzdiagramm 3 dargestellt. Dabei gibt der Kunde in Schritt 1-4 seine Kriterien an. In Schritt 5 wird dann die gewünschte Anfrage gestartet. Dafür wird die Funktion `calculateTranstions()` in 5.1 aufgerufen. Diese besorgt sich in 5.1 alle Buchungen aus der Datenbank und Filtert dann diese nach den gewünschten Kriterien. Sobald die Abarbeitung zu Ende ist werden die Ergebnisse dargestellt.



Sequenzdiagramm 3

DER VERWALTUNGSBEREICH

In dem Verwaltungsbereich können Logs eingesehen werden, Überweisungen getätigt werden sowie Zinseinstellungen geändert werden. Um zum Verwaltungsbereich zu gelangen muss im Logindialog als ID "`admin`" und als Passwort ebenfalls "`admin`" verwendet werden.

LOGANSICHT

Nach dem anmelden zum Verwaltungsbereich wird man zur Logansicht weitergeleitet. Hier können die Logs angesehen und gefiltert werden. Durch das freilassen der "Von" und "Bis" Felder und klicken des "Filtern" Buttons werden alle Logeinträge angezeigt. Filtereinstellungen müssen in Form von "DD.MM.YYYY" eingetragen werden. Zeitangaben sind in GMT+0.

ÜBERWEISUNGSSICHT

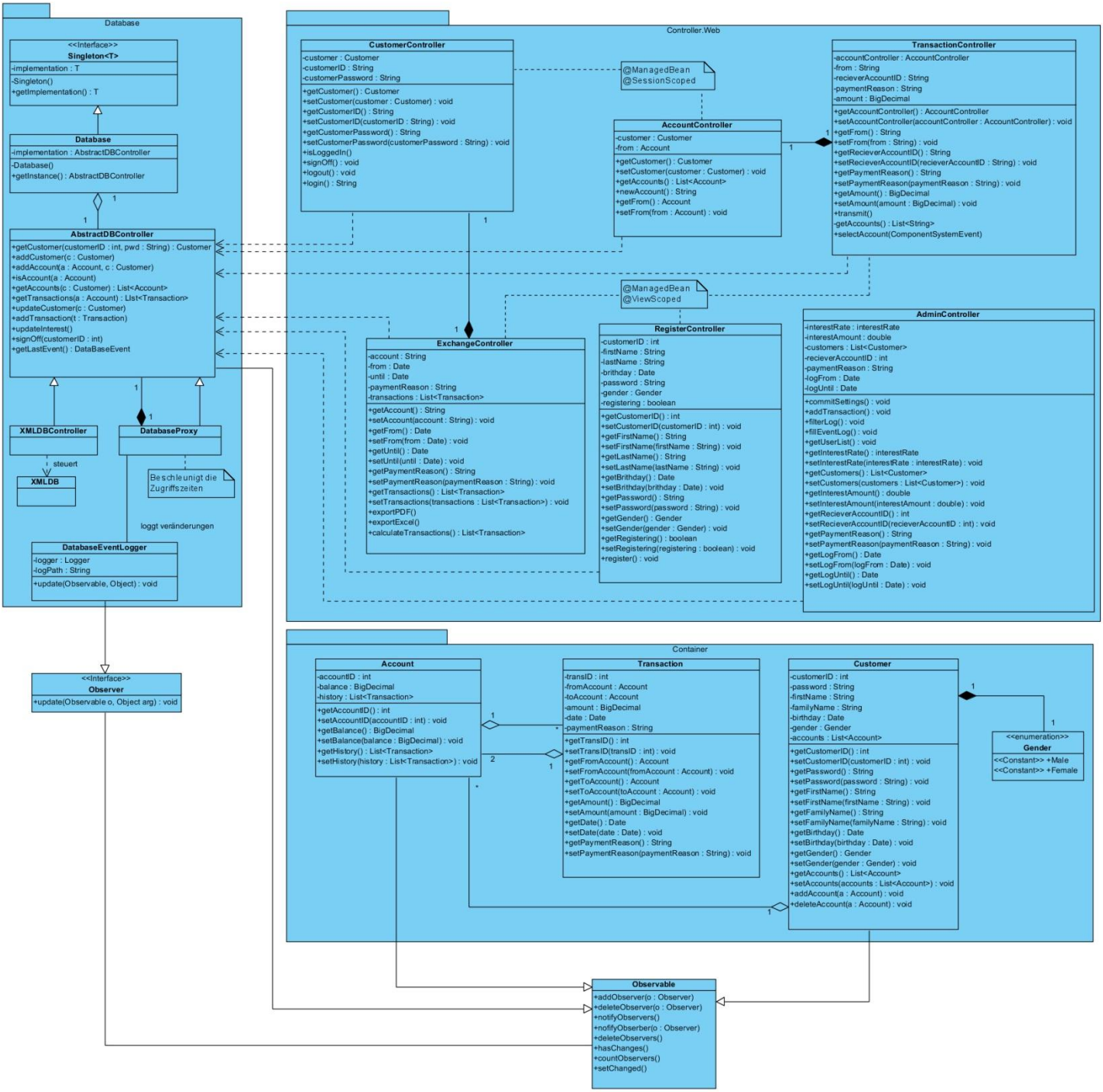
Eine weitere Funktion des Verwaltungsbereichs ist Überweisungen von der Bank an beliebige Konten zu betätigen. Es muss die Kontonummer des Empfängers, ein Verwendungszweck sowie ein Betrag eingetragen werden. Diese Überweisungen werden bei den Kunden als vom System kommend angezeigt.

Um ein Konten eines Nutzers einfacher zu finden, werden unterhalb des Überweisungsdialogs alle Kunden und ihre Konten aufgelistet.

ZINSEINSTELLUNGEN

Im Zinseinstellungspanel können die Zinsen sowie die Zins-rate zu welcher Zinsen überwiesen werden eingestellt werden. Die Zinsen müssen in Prozent angegeben werden. Mögliche Einstellungen zur Zins-rate sind jährliche, monatliche, wöchentliche, tägliche und stündliche Zinsen.

DAS KLASSENDIAGRAMM



Klassendiagramm 1 : Übersicht über die Packages sowie die Klassen

Eine große Version des Klassendiagramms 1 ist hier zu finden.

Im Klassendiagramm 1 wird der Aufbau des Bank-Systems dargestellt. Es ist im Grunde in drei große Packages eingeteilt:

- Im `Database` Package liegen die Klassen für die Verwaltung der Datenbank zur Verfügung.
- Im `Controller.Web` Package liegen die Kommunikationsschnittstellen zwischen dem Frontend und Backend, da das Design an das Model View Controller Pattern angelegt ist. Die View ist in diesem Fall die Webseite selber. Die Models liefert die Datenbank.
- Im Container Package liegen die grundlegenden Strukturen mit denen die meistens Klassen aus dem `Database` und dem `Controller.Web` Package arbeiten.

Es existieren noch weitere Behelfspackages, die jedoch nicht weiter wichtig sind. Daher wird im Weiteren nicht weiter auf sie eingegangen.

DAS DATABASE PACKAGE

Das Package setzt sich aus fünf Klassen zusammen.

- Database
  - Ist eine Klasse die das Singleton Pattern einsetzt. Dadurch ist der leichte Austausch der Datenbank möglich.
  - Die `getInstance()` Methode liefert eine Instanz des `AbstractDBContollers`.
- AbstractDBController
  - Ist eine abstrakte Klasse die von der `java.util.Observable` Klasse erbt, sodass hier Listener registriert werden können um auf Datanbankevents zu lauschen.
  - Stellt ein Interface für die Kommunikation mit der Datenbank bereit, was von allen erbenden Klassen implementiert werden muss.
- DatabaseProxy
  - Ist eine Implementierung des `AbstractDBControllers` und setzt das Proxy Pattern ein.
  - Der Proxy dient selber als Cache um die Zugriffszeiten auf die Datenbank zu beschleunigen, damit Anfragen von gerade aktiven Kunden nicht jedes Mal umständlich aus der Datenbank geladen werden müssen, sondern bequem im RAM gehalten werden können, solange der Kunde eingeloggt ist.
  - Da der `DatabaseProxy` das Proxy Pattern einsetzt braucht er natürlich eine Implementierung des `AbsractDBControllers` mit der er arbeiten kann.
- XMLDBController
  - Ist eine Implementierung des `AbstractDBControllers`.
  - Handhabt die Kommunikation mit der `Berkeley DB XML`.
- DatabaseEventLogger
  - Ist ein `Observer` und lauscht auf die Events des `DatabaseProxy`s und schreibt diese in eine Logdatei.

Weitere Einzelheiten zur Datenbank sind im Kapitel „Die Datenbank und Weiterentwicklung“ zu finden.

DAS CONTROLLER.WEB PACKAGE

Wie im Klassendiagramm modelliert, setzt sich dieses Package aus sechs Klassen zusammen. Diese dienen Handhaben die Kommunikation zwischen der View und der Datenbank.

- RegisterController
  - Registriert neue Kunden
- CustomerController
  - Kümmert sich um das An- und Abmelden von Kunden
  - Kümmert sich um Änderung in den Stammdaten
- AccountController
  - Kann neue Konten für ein Kunden anlegen
  - Stellt vorhandene Konten bereit
- ExchangeController
  - Stellt für ein gewähltes Konto die getätigten Überweisung zusammen
  - Erlaubt das Filtern von Überweisungen nach Datum oder Verwendungszweck
- TransactionController
  - Kümmert sich um Überweisungen
  - Mach Fehler Analyse wie z.B. ob des Empfängerkonto vorhanden ist oder ob genug Geld bereit liegt
  - Gibt Fehlermeldungen zurück, falls Angaben fehlerhaft sind
- AdminController
  - Fast administrative Aufgaben zusammen
  - Einstellen von Zinsen pro Stunde/Tag/Monat/Jahr
  - Einsicht in das Log-File im Front-End
  - Bietet die Möglichkeit an Überweisungen von der Bank zu einem bestimmten Konto durchzuführen. (Simulation einer Einzahlung am durch den Mitarbeiter)
  - Einsicht aller registrierten Kunden sowie deren Kontonummern

DAS CONTAINER PACKAGE

Hier liegen die Entitäten die für die Datenbank und die Controller gebraucht werden.

- Customer
  - Übliche Angaben zu einem Kunden die gebraucht werden wie Vorname, Nachname, Geburtsdatum, Geschlecht usw. …
  - Führt eine Konten Liste
  - Erbt von `java.util.Observable`, sodass er auch gegeben falls Events senden kann. Es wird jedoch nicht weiter Gebrauch davon gemacht.
- Account
  - Modelliert die üblichen Angaben eines Kontos wie Kontonummer und Kontostand
  - Hält eine Liste von getätigten Überweisungen
  - Erbt von `java.util.Observable`, sodass die Klasse auch gegeben falls Events senden kann.
- Transaction
  - Enthält Angaben von welchem zu welchem Konto eine Überweisung stattfinden soll
  - Höhe der Überweisung
  - Grund der Überweisung
  - Datum der Überweisung

VERWENDETE DESIGN PATTERNS

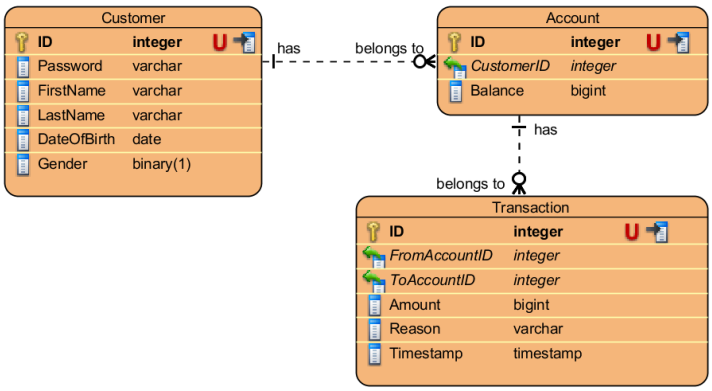
Die Design Patterns wurden für bei der Datenbankschnittstelle angewendet:

- Das Singleton Pattern
  - Die Klasse `Datenbase` verwendet dieses Pattern. Der Konstruktor ist `private` sodass keine Instanz erstellt werden kann. Um eine Instanz zu erhalten muss stattdessen die Methode `getInstance()` verwendet werden.
- Das Proxy Pattern
  - Die Klasse `DatabaseProxy` verwendet dieses Pattern. Der Proxy dient selber als Cache um die Zugriffszeiten auf die Datenbank zu beschleunigen, damit Anfragen von gerade aktiven Kunden nicht jedes Mal umständlich aus der Datenbank geladen werden müssen, sondern bequem im RAM gehalten werden können, solange der Kunde eingeloggt ist.
- Den Observer Pattern
  - Alle Klassen die `AbstractDBController` implementieren sind `Observable`. Eine dieser Klassen ist das `DatabaseProxy`; es werden Events bei Datenbankänderungen an alle `Observer` geschickt. Eine dieser `Observer` ist die Klasse `DatabaseEventLogger`, welche die bekommenen Events in eine Logdatei schreibt.
- Das Bridge Pattern
  - Die Klasse `Database` verwendet dieses Pattern. Es wird eine konkrete Implementation der Klasse `AbstractDBController` in dieser Klasse gehalten.

DIE DATENBANK UND WEITERENTWICKLUNG

Standardmäßig wird `Berkeley DB XML` als Datenbank verwendet. Diese Datenbank arbeitet mit XML-Dokumenten und verwendet als Anfragesprache XQuery. Es sind drei Tabellen zur Speicherung der Daten notwendig:

- eine Kundentabelle welche alle Daten der Kunden sowie das Passwort als SHA-256 hash enthält,
- eine Kontentabelle welche alle Konten enthält,
- und eine Überweisungstabelle welche alle Überweisungen enthält.



Die Datenbank ist als Singleton implementiert. Um diese zu verwenden reicht lediglich ein Aufruf von `Database.getInstance()`.

AUSTAUSCH DER DATENBANK

Die Datenbank ist leicht austauschbar. Es muss nur die Klasse `AbstractDBController` implementiert werden. Weiterhin muss in der Klasse `Database` in der Methode `getInstance()` die variable `concreteDatabase` auf die neue Datenbank gesetzt werden. Die neue Datenbank profitiert automatisch vom Proxy, welches als Cache dient. Es werden auch weiterhin Logs erstellt.

WEITERE MÖGLICHKEITEN

Da jede Klasse welche `AbstractDBController` implementiert `Observable` sind, können `Observer` geschrieben werden welche auf `DatabaseEvent`s reagieren. Standardmäßig sendet das `DatabaseProxy` folgende Events:

- wenn sich ein Kunde anmeldet (`DatabaseEventType.CUSTOMER_LOGGED_IN`),
- wenn sich ein Kunde registriert (`DatabaseEventType.CUSTOMER_CREATED`),
- wenn ein Kunde seine Daten ändert (`DatabaseEventType.CUSTOMER_CHANGED`),
- wenn ein Konto eröffnet wird (`DatabaseEventType.ACCOUNT_CREATED`),
- wenn eine Überweisung in Auftrag gegeben wird (`DatabaseEventType.TRANSACTION_PROCESSED`),
- wenn die Zinsen für jedes Konto zugewiesen werden (`DatabaseEventType.TRANSACTION_INTEREST`).

Ein fertiger `Observer` ist der `DatabaseEventLogger`; dieser erstellt bei jedem Event ein Logeintrag. Diese sind im Verwaltungsbereich einsehbar.