

Online Banking System

Dorancelis Cordones(x16130936, Waleed Sajid(x01520989), Ahmad



April 24, 2017

HDIP in SCIENCE IN computing (cloud computing

NCIRL

Contents

[Introduction: 2](#_Toc480856401)

[Technical Background: 3](#_Toc480856402)

[REST: 3](#_Toc480856403)

[JAX-RS: 3](#_Toc480856404)

[Jersey : 3](#_Toc480856405)

[Database Design 4](#_Toc480856406)

[Database Schema 4](#_Toc480856407)

[ER-Diagram 6](#_Toc480856408)

[Flow Diagram Bank System 6](#_Toc480856409)

[Use Case Diagram: 7](#_Toc480856410)

[Implementation 8](#_Toc480856411)

[1) HomePage : 8](#_Toc480856412)

[2) Signup for the Customers: 8](#_Toc480856413)

[3) Lists the Customers: 9](#_Toc480856414)

[4) Lists the Accounts: 9](#_Toc480856415)

[5) Lists a particularAccount: 10](#_Toc480856416)

[6) Lists the Transactions: 11](#_Toc480856417)

[7) Lodgement : 12](#_Toc480856418)

[**[1] GANFIELD, K.**Creating an Enterprise Application Using Maven. URL : *Netbeans.org*. N.p., 2017. Web. 24 Apr. 2017. 14](#_Toc480856419)

[[2]"IBM Knowledge Center". Ibm.com. N.p., 2017. Web. 17 Apr. 2017. 14](#_Toc480856420)

[[3] "Developing REST Apis With JAX-RS - Java Brains". Javabrains.io. N.p., 2017. Web. 16 Apr. 2017. 14](#_Toc480856421)

[[4] Lars Vogel, 2016 vogella GmbH. "REST With Java (JAX-RS) Using Jersey - Tutorial". Vogella.com. N.p., 2017. Web. 24 Apr. 2017. 14](#_Toc480856422)

[[5] Restful Web Services Tutorial". www.tutorialspoint.com. N.p., 2017. Web. 13 Apr. 2017. 14](#_Toc480856423)

# Introduction:

Advances in the field of computer science have led to massive automation in many of the sectors. One of the predominant sectors that had great impact because of these advances is financial sector. Over the last decade many of the banks has inclined more towards making the business automated. This was demonstrated when most on the banks published online banking system.

The main objective of this project is to provide an online platform in the banking sector that publishes RESTful web services in order to facilitate customers and bank administrative person to check the important entities such as customers, accounts, transaction details. The system allows and should allow customers to view their personnel accounts.

This project projects the facilities provided generally by any banking system such as checking accounts, lodging money, and withdrawing money, viewing transactions, and viewing its customers.

When it comes to banking sector, confidentiality plays a vital role. Hence security is a great concern.

# Technical Background:

Online Banking System projects implements RESTful web services using Java**(***Jersey / JAX-RS).*

## REST:

REST is an architectural style, based on web-standards and the HTTP protocol. In a REST based architecture everything is a resource. A resource is accessed via a common interface based on the HTTP standard methods. REST permits resources to have different representations, e.g., text, XML, JSON etc.

There are 4 major HTTP methods namely GET, POST, PUT, DELETE.

## JAX-RS:

Java defines REST support via the Java Specification Request (JSR) 311. This specification is called JAX-RS (The Java API for RESTful Web Services). JAX-RS uses annotations to define the REST relevance of Java classes.

## Jersey:

Jersey is the reference implementation for the JSR 311 specification.

The Jersey implementation provides a library to implement Restful web services in a Java servlet container.

This is order to implement this project, following state-of-art were taken into consideration:

1. Java JDK version (1.7 and above)
2. Jersey version(2.x)
3. Netbeans
4. Maven
5. MySQL.
6. Derby

## Database Design

In order to implement the database for the online banking system, MySQL is taken into consideration. To design database, total 4 tables were implemented as follows:

1. Customer
2. Accounts
3. Transactions
4. Login

## Database Schema

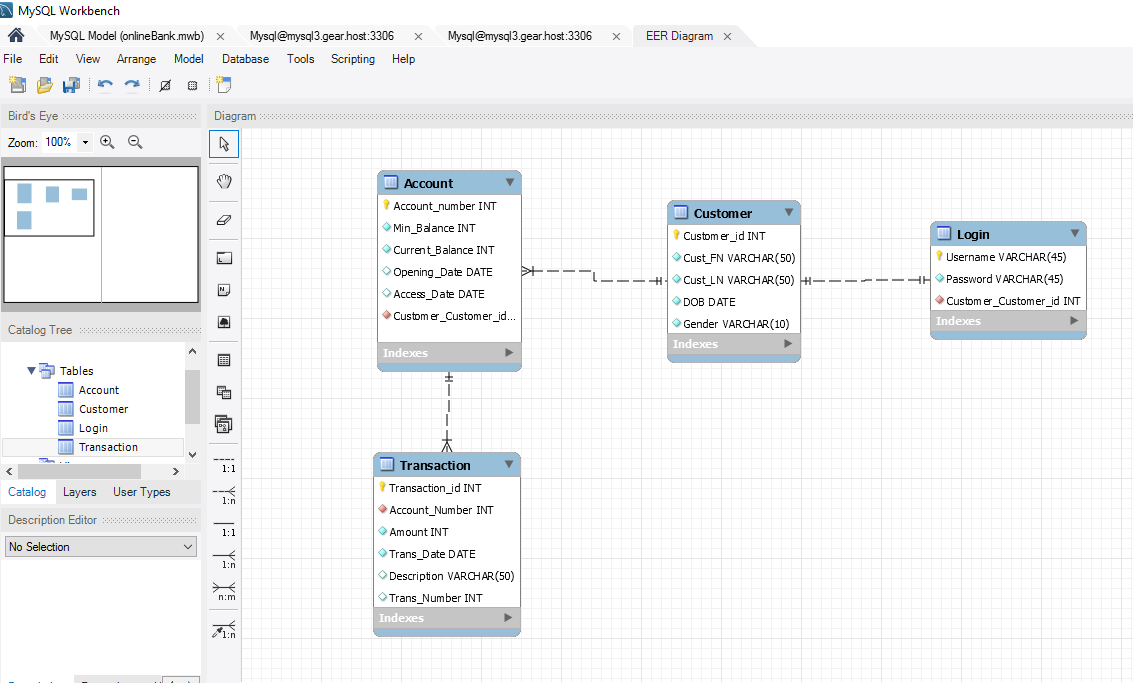
|  |  |  |  |
| --- | --- | --- | --- |
| **Customer** | | | |
| **Column name** | **Data type** | **Null able** | **Primary key** |
| Customer\_id | INT | No | Yes |
| Name | Varchar(50) | No | No |
| Address | Varchar(50) | No | No |

|  |  |  |  |
| --- | --- | --- | --- |
| **Login** | | | |
| **Column name** | **Data type** | **Null able** | **Primary key** |
| Username | Varchar(45) | No | Yes |
| Password | Varchar(45) | No | No |
| Customer\_Customer\_id(FK) | INT | No | No |

|  |  |  |  |
| --- | --- | --- | --- |
| **Account** | | | |
| **Column name** | **Data type** | **Null able** | **Primary key** |
| Account\_number | INT | No | Yes |
| Current \_Balance | INT | No | No |
| Customer\_Customer\_id(FK) | INT | NO | No |

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | | | |
| **Column name** | **Data type** | **Null able** | **Primary key** |
| Transaction\_id | INT | NO | Yes |
| Account\_Number(FK) | INT | NO | No |
| Amount | INT | NO | No |

## ER-Diagram



## Flow Diagram Bank System

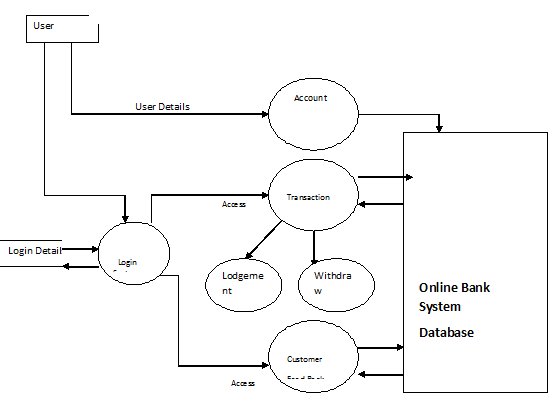
****

Figure 1: Flow Diagram of Bank System

## Use Case Diagram:

Database

BankSystemm

Customer

# 

# Implementation

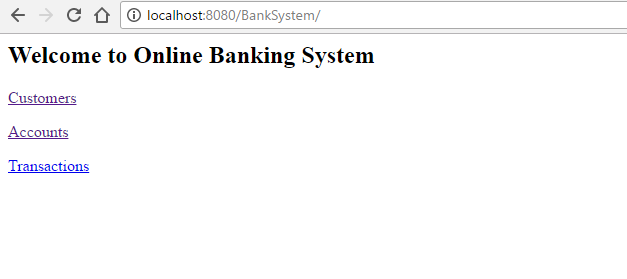
While designing the Restful API the Design Principles Separation of Concerns were followed where the implementation was categorized into 3 parts for each API: Model, Resource and Service.

RestfulAPIs:

## 1**) Homepage:**

Description: The homepage of the developed banking system where an administrative person can access the Customers, Accounts and Transactions.

URL: <http://localhost:8080/BankSystem>



## 2) Signup for the Customers:

**Description –** Login feature provided to the customer to check their details and perform tasks like lodgment, withdrawal.

PATH: <http://localhost:8080/BankSystem/webapi/home>

HTTP VERB: GET

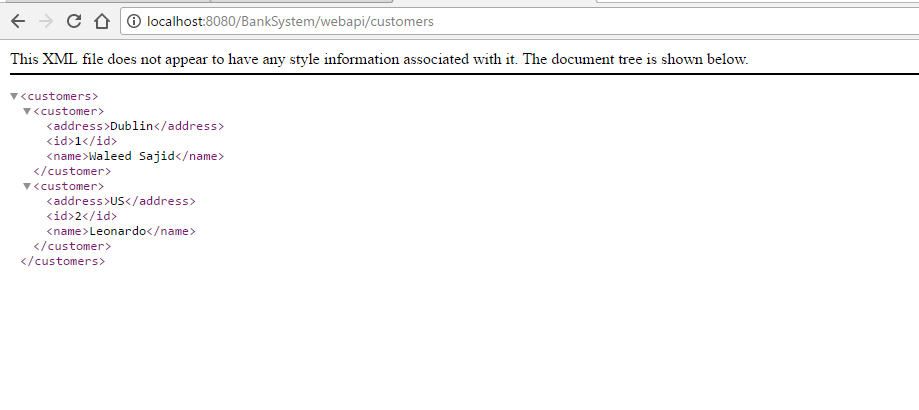
## 3) Lists the Customers:

Description- This feature lists the number of customers registered into the system.

HTTP VERB: GET

URL:<http://localhost:8080/BankSystem/webapi/customers/>

Resource name : Customers



## 4) Lists the Accounts:

Description- This feature lists the number of accounts registered into the system.

HTTP VERB: GET

URL:<http://localhost:8080/BankSystem/webapi/accounts/>

Resource name : Accounts

Resource content:



## 5) Lists a particular Account:

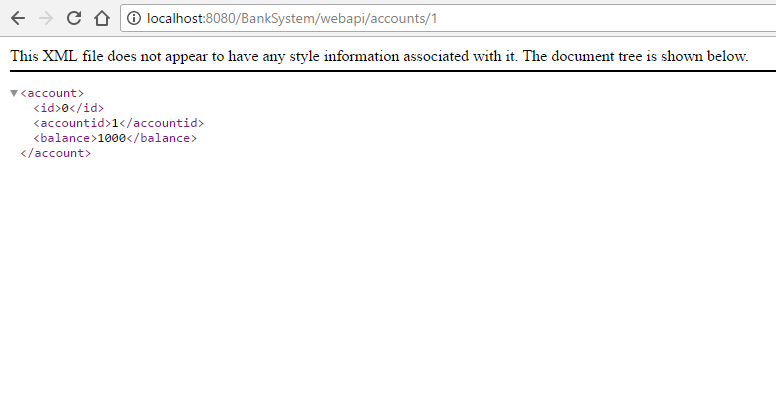
Description- This feature lists a particular account registered into the system.

HTTP VERB: GET

URL:<http://localhost:8080/BankSystem/webapi/accounts/>{accountid}

Resource name: Accounts

Resource content:



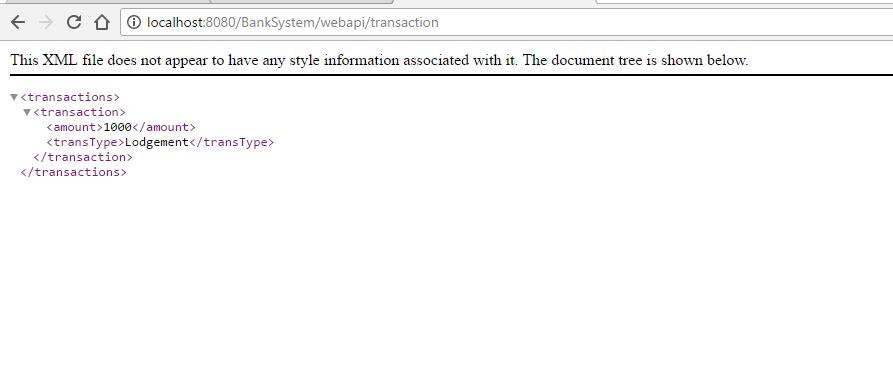
## 6) Lists the Transactions:

Description- This feature lists the number of transactions performed into the system. There are two types of transactions namely, Lodgement, Withdraw.

HTTP VERB: GET

URL:<http://localhost:8080/BankSystem/webapi/transaction/>

Resource name:Transactions

Resource content :

## 7) Lodgement:

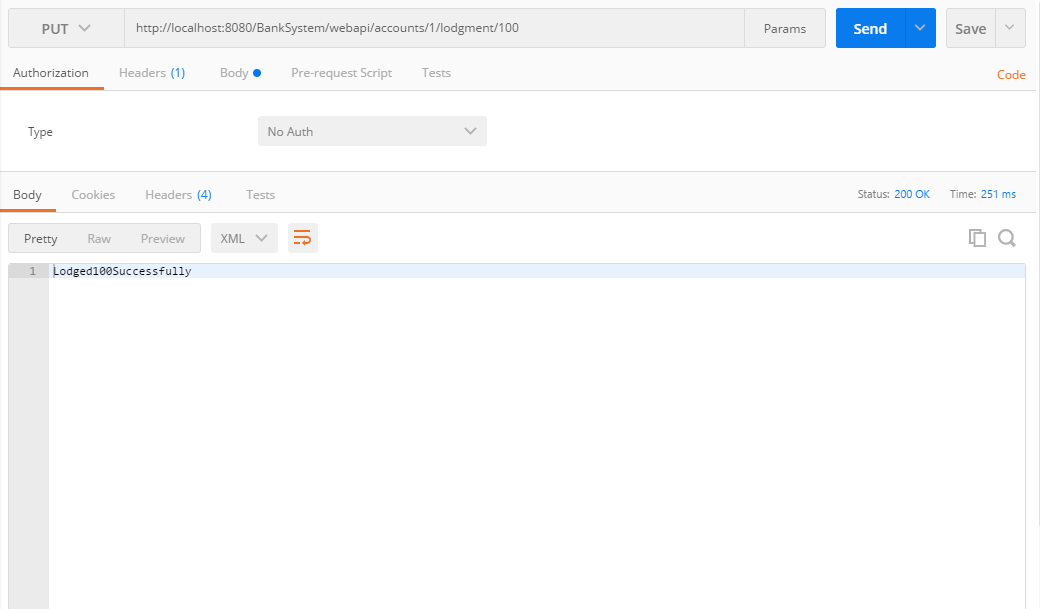
Description- This feature performs lodgement of amount into particular bank account.

HTTP VERB:PUT

URL:<http://localhost:8080/BankSystem/webapi/accounts/1/lodgment/100>

Resource name: Accounts

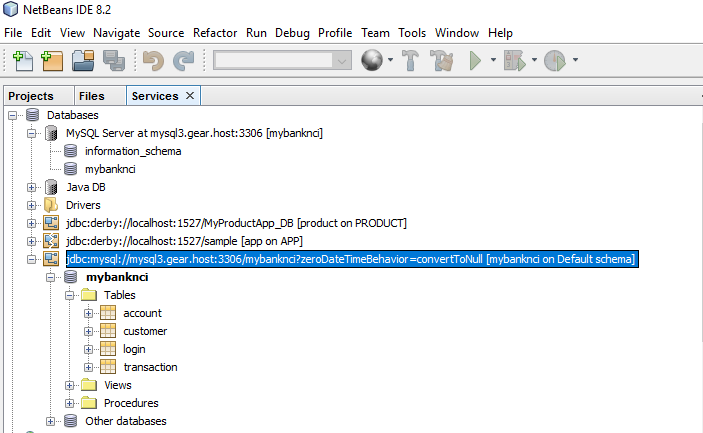
Resource content:



Database is implemented using JPA technology and following as the screenshots for the configuration done in persistence.xml

Online cloud server Gear Host is used to sae data. Connection details are provided in the screen shot as well as a text file in the submission zip file.





**Conclusion and enhancements:**

As mentioned in the above sections, we end favoured to develop a website for a banking system. This project was developed based upon the project specification and the requirements of the module. While developing this project the main focus was to implement an online banking system with the perspective of bank administrative and customers. This system was developed by publishing the Restful web services using latest technologies like JAX-RS , Jersey, JPA , MySQL. This system enlists and creating customers, accounts, transactions and performs tasks like lodgement and withdrawal of money.

For enhancing this projects operations such as deleting ,updating customers, accounts, transactions can be implemented.

**References:**

**[1] GANFIELD, K.**Creating an Enterprise Application Using Maven. URL : *Netbeans.org*. N.p., 2017. Web. 24 Apr. 2017.

### [2]"IBM Knowledge Center". Ibm.com. N.p., 2017. Web. 17 Apr. 2017.

### [3] "Developing REST Apis With JAX-RS - Java Brains". Javabrains.io. N.p., 2017. Web. 16 Apr. 2017.

### [4] Lars Vogel, 2016 vogella GmbH. "REST With Java (JAX-RS) Using Jersey - Tutorial". Vogella.com. N.p., 2017. Web. 24 Apr. 2017.

### [5] Restful Web Services Tutorial". www.tutorialspoint.com. N.p., 2017. Web. 13 Apr. 2017.

### [6] Week9.Lab JPA - JAVA Persistence API.pdf”.

### [7] Week 8 Lab Restful Web Services 3”. https://moodle.ncirl.ie/pluginfile.php/394514/mod\_resource/content/0/RESTful%20Web%20Services%20III.pdf.

Project Code :

**Resource :**

**Customers.java**

/\*

\* @reference Week9.LAB JPA - JAVA Persistence API.pdf \*/

**package** com.banksystem.BankSystem;

**import** java.util.List;

**import** javax.ws.rs.Consumes;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.POST;

**import**javax.ws.rs.PUT;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.PathParam;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.MediaType;

**import** com.banksystem.BankSystem.bean.Account;

**import** com.banksystem.BankSystem.bean.AccountService;

**import** com.banksystem.BankSystem.bean.Customer;

**import** com.banksystem.BankSystem.bean.CustomerService;

**import** javax.ws.rs.DELETE;

@Path("/customers")

**publicclass** Customers {

CustomerService customerservice= **new** CustomerService();

AccountService service= **new** AccountService();

// This method is called if XML is request

@GET

@Produces(MediaType.***TEXT\_XML***)

**public** List<Customer> getCustomers() {

**return**customerservice.getAllMessages();

}

// This method is called if HTML is request

@GET

@Path("{CustomerId}")

@Produces(MediaType.***TEXT\_XML***)

**public** Customer updateCustomerInfo(@PathParam("CustomerId") **long**CustomerId)

{

//Code to update customer record using customerid

**return**customerservice.getCustomer(CustomerId);

}

@GET

@Path("{CustomerId}/ACC")

@Produces(MediaType.***TEXT\_XML***)

**public** List<Account> getAccount(@PathParam("CustomerId") **long**CustomerId) {

**return**service.getAccountidbyCust(CustomerId);

}

@POST

@Consumes({MediaType.***APPLICATION\_XML***, MediaType.***APPLICATION\_JSON***})

@Produces({MediaType.***APPLICATION\_XML***, MediaType.***APPLICATION\_JSON***})

**public** Customer saveBook(Customer b) {

**return**customerservice.createCustomer(b);

}

@DELETE

@Path("/{id}")

@Produces({MediaType.***APPLICATION\_XML***, MediaType.***APPLICATION\_JSON***})

**publicvoid** removeBook(@PathParam("id") **int**id) {

customerservice.deleteCustomer(id);

}

}

**Accounts.java**

**package** com.banksystem.BankSystem;

**import** java.util.List;

**import**javax.ws.rs.Consumes;

**import** javax.ws.rs.GET;

**import**javax.ws.rs.POST;

**import** javax.ws.rs.PUT;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.PathParam;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.MediaType;

**import** com.banksystem.BankSystem.bean.Account;

**import** com.banksystem.BankSystem.bean.AccountService;

**import** com.banksystem.BankSystem.bean.Customer;

**import**com.banksystem.BankSystem.bean.CustomerService;

**import**com.banksystem.BankSystem.bean.Transaction;

@Path("/accounts")

**publicclass** Accounts {

AccountService customerservice= **new** AccountService();

// This method is called if XML is request

@GET

@Produces(MediaType.***TEXT\_XML***)

**public** List<Account> getAccount() {

**return**customerservice.getAllAccounts();

}

@GET

@Path("{AccountId}")

@Produces(MediaType.***TEXT\_XML***)

**public** Customer updateCustomerInfo(@PathParam("AccountId") **long**AccountId)

{

//Code to update customer record using customerid

**return**customerservice.getAccount(AccountId);

}

@PUT

@Path("{AccountId}/lodgment/{amount}")

@Produces(MediaType.***TEXT\_XML***)

**public** String updateCustomerLodgeInfo(@PathParam("AccountId") **long**AccountId,@PathParam("amount") **int**amount)

{

//Code to update customer record using customerid

**if**(customerservice.getAccountLodge(AccountId,amount))

{

**return**"Lodged"+amount+"Successfully";

}

**return**"Failed to Lodge the amount";

}

@PUT

@Path("{AccountId}/withdraw/{amount}")

@Produces(MediaType.***TEXT\_XML***)

**public** String updateCustomerWithdrawInfo(@PathParam("AccountId") **long**AccountId,@PathParam("amount") **int**amount)

{

//Code to update customer record using customerid

**if**(customerservice.getAccountWithdraw(AccountId,amount))

{

**return**"Withdaw"+amount+"Successfully";

}

**return**"Failed to Withdraw the amount. Check the balance";

}

**Transactions.java**

**package** com.banksystem.BankSystem;

**import** java.util.List;

**import**javax.ws.rs.Consumes;

**import** javax.ws.rs.GET;

**import**javax.ws.rs.POST;

**import**javax.ws.rs.PUT;

**import** javax.ws.rs.Path;

**import**javax.ws.rs.PathParam;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.MediaType;

**import**com.banksystem.BankSystem.bean.Account;

**import**com.banksystem.BankSystem.bean.AccountService;

**import**com.banksystem.BankSystem.bean.Customer;

**import**com.banksystem.BankSystem.bean.CustomerService;

**import** com.banksystem.BankSystem.bean.Transaction;

**import** com.banksystem.BankSystem.bean.TransactionService;

**import**com.banksystem.BankSystem.bean.Transaction;

@Path("/transaction")

**publicclass** Transactions {

TransactionService customerservice= **new** TransactionService();

// This method is called if XML is request

@GET

@Produces(MediaType.***TEXT\_XML***)

**public** List<Transaction> getTransaction() {

**return**customerservice.getAllTransactions();

}

}

**Service**

**CustomerService.java**

/\*

\* Week9.LAB JPA - JAVA Persistence API.pdf

\* \*/

**package** com.banksystem.BankSystem.bean;

**import**java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.EntityTransaction;

**import** javax.persistence.Persistence;

**import** javax.persistence.TypedQuery;

**import** javax.persistence.criteria.CriteriaBuilder;

**import** javax.persistence.criteria.CriteriaQuery;

**import** javax.persistence.criteria.Root;

**publicclass** CustomerService {

**private** Map<Long, Customer>customer = DatabaseClass.*getCustomers*();

**private** EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("Customer");

**private** EntityManager em = emf.createEntityManager();

**private** EntityTransaction tx = em.getTransaction();

**public** CustomerService() {

// customer.put(1L, new Customer(1, "WaleedSajid", "Dublin","11"));

// customer.put(2L, new Customer(2, "Leonardo", "US","22"));

}

**public** List<Customer> getAllMessages() {

**return** allEntries();

}

/\*

\* @reference

\* Week9.LAB JPA - JAVA Persistence API.pdf\*/

**public** List<Customer> allEntries() {

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<Customer>cq = cb.createQuery(Customer.**class**);

Root<Customer>rootEntry = cq.from(Customer.**class**);

CriteriaQuery<Customer>all = cq.select(rootEntry);

TypedQuery<Customer>allQuery = em.createQuery(all);

**return**allQuery.getResultList();

}

**public** Customer getCustomer(**long**id) {

**return**customer.get(id);

}

**public** Customer createCustomer(Customer b) {

System.***out***.println("inCreateCustomer");

Customer test = em.find(Customer.**class**, b.getId());

System.***out***.println("inCreateCustomer");

**if** (test == **null**) {

tx.begin();

em.persist(b);

tx.commit();

em.close();

}

**return**b;

}

**publicvoid** deleteCustomer(**int**id) {

Customer test = em.find(Customer.**class**, id);

**if** (test !=**null**) {

tx.begin();

em.remove(test);

tx.commit();

em.close();

}

}

}

**AccountService.java**

**package** com.banksystem.BankSystem.bean;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**publicclass** AccountService {

**private** Map<Long, Account>acc = DatabaseClass.*getAccounts*();

**private** Map<Long, Transaction>trans = DatabaseClass.*getTransactions*();

**public** AccountService() {

acc.put(1L, **new** Account(1,1000));

acc.put(2L, **new** Account(2,2000));

acc.put(3L, **new** Account(2,2000,1));

trans.put(1L, **new** Transaction(1L, 1000, "Lodgement"));

}

**public** List<Transaction> getAllTransactions() {

//Map<Long, Account>acc

**returnnew** ArrayList<Transaction>(trans.values());

}

**public** List<Account> getAllAccounts() {

//Map<Long, Account>acc

**returnnew** ArrayList<Account>(acc.values());

}

**public** List<Account> getAccountidbyCust(**long**id)

{

ArrayList<Account>accountbycust = **new** ArrayList<Account>();

**for** (**long**key : acc.keySet()) {

//

**if**(( acc.get(key).equals(id)))

{

accountbycust.add(accountbycust.size()+1, (Account)(acc.get(key)));

}

}

**return**accountbycust;

}

**public** Account getAccount(**long**id) {

**return**acc.get(id);

}

**publicboolean** getAccountLodge(**long**accountId, **int**amount) {

// **TODO** Auto-generated method stub

**int**balance=acc.get(accountId).getBalance();

balance=balance+amount;

acc.get(accountId).setBalance(balance);

System.***out***.println("balance"+acc.get(accountId).getBalance());

trans.put(1L, **new** Transaction(accountId,amount,"lodge"));

**returntrue**;

}

**publicboolean** getAccountWithdraw(**long**accountId, **int**amount) {

// **TODO** Auto-generated method stub

**int**balance=acc.get(accountId).getBalance();

balance-=amount;

**if**(balance>0)

{

acc.get(accountId).setBalance(balance);

trans.put(2L, **new** Transaction(accountId,amount,"withdraw"));

**returntrue**;

}

**else**

**returnfalse**;

}

}

**TransactionService.java**

**package** com.banksystem.BankSystem.bean;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**publicclass** TransactionService {

**private** Map<Long, Transaction>trans = DatabaseClass.*getTransactions*();

**public** TransactionService() {

trans.put(1L, **new** Transaction(1L, 1000, "Lodgement"));

}

**public** List<Transaction> getAllTransactions() {

//Map<Long, Account>acc

**returnnew** ArrayList<Transaction>(trans.values());

}

}

Model

**Account.java**

**package** com.banksystem.BankSystem.bean;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

**publicclass**Account**extends** Customer{

**privateint**Accountid;

**privateint**Balance;

**public** Account() {

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@return** the id

\*/

**public** Account(**int**id, **int**Balance) {

// **TODO** Auto-generated constructor stub

**this**.Accountid=id;

**this**.Balance=Balance;

}

**public** Account(**int**id, **int**Balance, **long**custid) {

// **TODO** Auto-generated constructor stub

**this**.Accountid=id;

**this**.Balance=Balance;

**super**.id= custid;

}

/\*\*

\* **@return** the accountid

\*/

**publicint** getAccountid() {

**return**Accountid;

}

/\*\*

\* **@param** accountid the accountid to set

\*/

**publicvoid** setAccountid(**int**accountid) {

Accountid = accountid;

}

/\*\*

\* **@return** the name

\*/

**publicint** getBalance() {

**return**Balance;

}

**publicvoid** setBalance(**int**balance) {

Balance = balance;

}

}

**Transaction.java**

**package** com.banksystem.BankSystem.bean;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

**publicclass** Transaction {

**privateint**Amount;

**privatelong**Accountid;

**private** String TransType;

**public** Transaction() {

// **TODO** Auto-generated constructor stub

}

**public** Transaction(**long**accountId2,**int**Amount,String string) {

// **TODO** Auto-generated constructor stub

**this**.Accountid=accountId2;

**this**.Amount=Amount;

**this**.TransType=string;

}

/\*\*

\* **@return** the amount

\*/

**publicint** getAmount() {

**return**Amount;

}

/\*\*

\* **@param** amount the amount to set

\*/

**publicvoid** setAmount(**int**amount) {

Amount = amount;

}

**public** String getTransType() {

**return**TransType;

}

**publicvoid** setTransType(String transType) {

TransType = transType;

}

/\*\*

\* **@return** the accountid

\*/

**publiclong** getAccountid() {

**return**Accountid;

}

/\*\*

\* **@param** accountid the accountid to set

\*/

**publicvoid** setAccountid(**int**accountid) {

Accountid = accountid;

}

}

**Customer.java**

**/\***

**\* Week9.LAB JPA - JAVA Persistence API.pdf**

**\* \*/**

**package com.banksystem.BankSystem.bean;**

**import javax.xml.bind.annotation.XmlElement;**

**import javax.xml.bind.annotation.XmlRootElement;**

**import java.io.Serializable;**

**import javax.persistence.Entity;**

**import javax.persistence.GeneratedValue;**

**import javax.persistence.GenerationType;**

**import javax.persistence.Id;**

**import javax.persistence.Table;**

**@Entity**

**@Table**

**@XmlRootElement**

**public class Customer implements Serializable {**

**@Id**

**protected long id;**

**protected String name;**

**protected String addresses;**

**public Customer() {}**

**public Customer(int i, String name, String addresses,String Accountid) {**

**this.id = i;**

**this.name = name;**

**this.addresses = addresses;**

**}**

**public Customer(int id) {**

**this.id = id;**

**}**

**public long getId() {**

**return id;**

**}**

**public void setId(long id) {**

**this.id = id;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**@XmlElement(name="address")**

**public String getAddresses() {**

**return addresses;**

**}**

**public void setAddresses(String addresses) {**

**this.addresses = addresses;**

**}**

**}**