Ryan Pepe

Professor Arias

Software Development I

26 April 2019

50/50

Project Writeup

Abstract

The digital banking application is composed of two GUI classes, “BankingApp.java,” and “UserGui.java.” In order for the application to function properly, the BankingApp class must be executed instead of the UserGui class. This system allows a user to create an account, log in to the system, create multiple checking and banking accounts, and perform operations with these accounts. I used Hibernate and Java annotations to make queries into my MySQL Workbench database in order to fetch and update data fields. Overall, this project required much more work than I anticipated, but I am glad to have been able to gain a firm understanding in software development as well as database design.

Introduction

When I first started at Marist College, I had no previous knowledge in any programming language, so I always felt that I was falling behind many of my classmates. However, I sought to increase my understanding of computer science as a whole by putting in as much effort as I could possibly afford. Since I never worked with databases before, it was difficult to figure out the necessary requirements to create a system that operated in conjunction to a database. In addition, I had no previous experience with creating a gui in Java, so implementing gui mechanics and incorporating database queries into the user interface was a tall task. Ultimately, I attempted to incorporate elements from the entirety of the course, including aspects of polymorphism, encapsulation, inheritance, abstract classes, and exception handling.

Detailed System Description

There are four main classes in the digital banking system: AccountHolder, Account, CheckingAccount, and SavingsAccount. There is a heirarchy between these classes as follows: AccountHolder is at the top, followed by the abstract class Account, followed by its two subclasses CheckingAccount and SavingsAccount. Each of these classes are properly annotated for the structure of the database that I created, where each data field corresponds to an attribute in the database schema.

Requirements

This project is addressing the problem of creating bank accounts digitally and making transactions through a desktop application. In addition, it is addressing the problem of keeping track of an influx of information that is being thrown through the system. Each piece of data will be stored into a database in order for the bank administrators to monitor account activity and see user information.

Literature Survey

Chase bank has created applications to transfer funds between accounts and view account activity; however, they do not allow one to create accounts and delete accounts. I seek to allow the user to create accounts remotely and generate a unique account number that will be stored in the database. Many modern banks are trying to implement similar features as the world begins to become more digitized.

User Manual

The bank application will have a visually appealing GUI which will make the process of performing bank actions fairly easy. On the first GUI, there are two tabs: Login and Create Account. The login tab will allow the user to access a user Gui specific to one's personal login information. The create account tab will simply allow a user to create a bank account user entry and upload it to the database. Once the user has access to an instance of the UserGui class, he will be able to create bank accounts, withdraw funds, deposit funds, retrieve account information, and transfer funds between two accounts owned by the user.

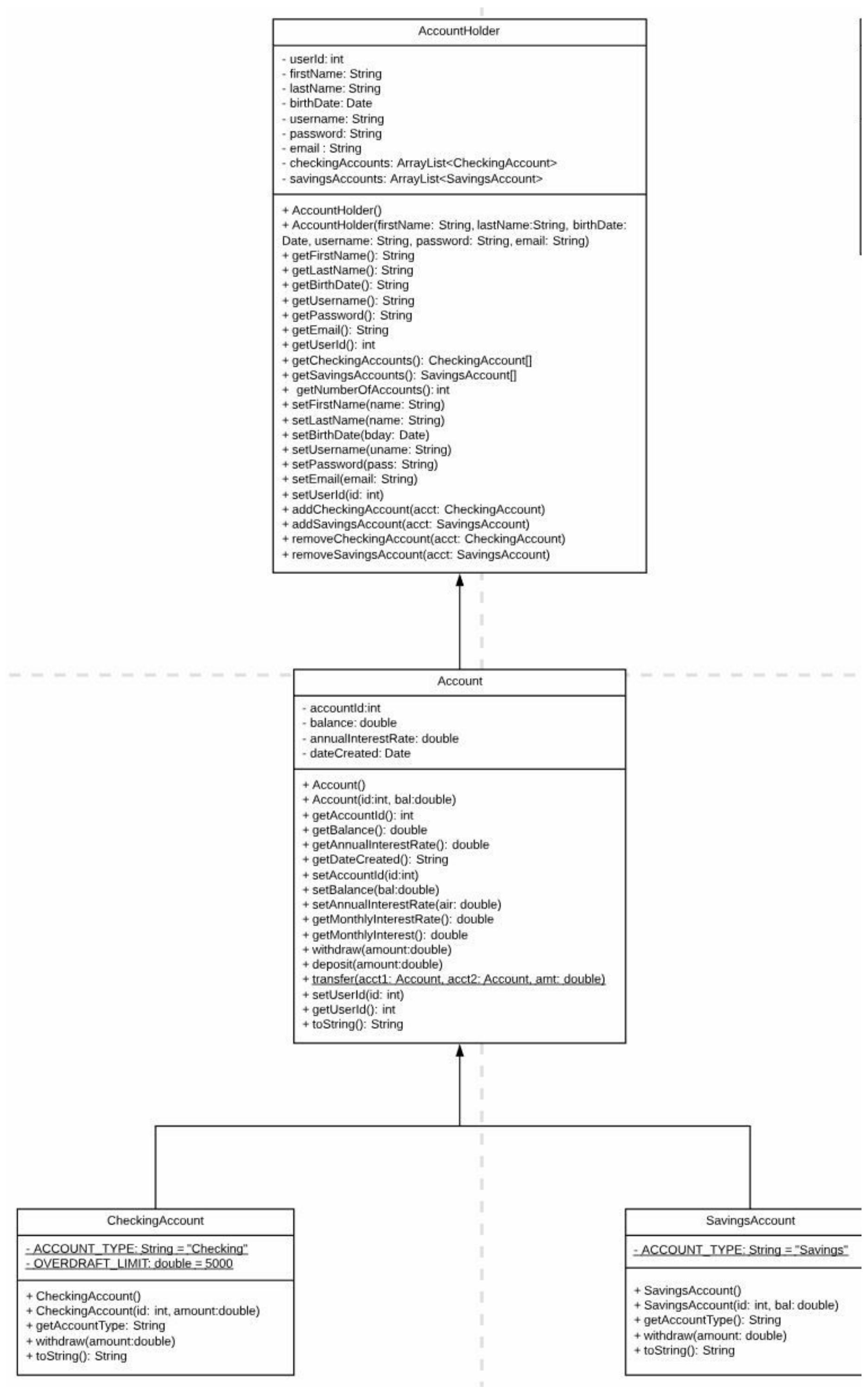
Conclusion

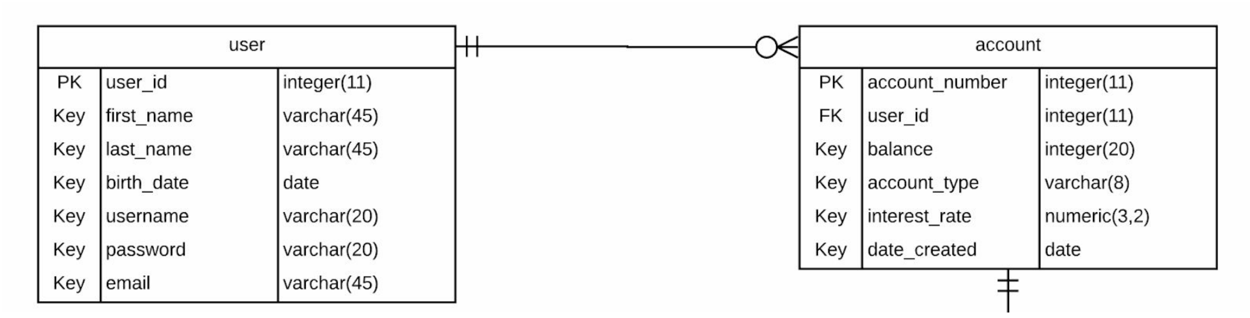
The digital banking system truly is a testament to the amount of dedication that I invested in this class. I did not expect for this project to take so much time and energy to complete, but now that I have finished, I know that the skills that I have earned will help me further my education at Marist College. Learning how to incorporate multiple classes, GUIs, and database tables is something that I now know that I am fully capable of completing in the future. This system accomplished my goal of having a fully exception-handled GUI with the ability to send and pull information to and from a database.

References/Bibliography

1. **Luv2Code Hibernate Tutorials**
   a. **https://www.youtube.com/playlist?list=PLEAQNNR8IlB7fNkRsUgzrR346i-UqE5CG**
2. **MySQL Workbench Manual**
   a. **https://dev.mysql.com/doc/workbench/en/**
3. **Hibernate ORM Final User Guide**
   a. **https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html**

UML CLASS DIAGRAM


AccountHolder
- userId: int
- firstName: String
- lastName: String
- birthDate: Date
- username: String
- password: String
- email : String
- checkingAccounts: ArrayList<CheckingAccount>
- savingsAccounts: ArrayList<SavingsAccount>
+ AccountHolder()
+ AccountHolder(firstName: String, lastName:String, birthDate: Date, username: String, password: String, email: String)
+ getFirstName(): String
+ getLastName(): String
+ getBirthDate(): String
+ getUsername(): String
+ getPassword(): String
+ getEmail(): String
+ getUserId(): int
+ getCheckingAccounts(): CheckingAccount[]
+ getSavingsAccounts(): SavingsAccount[]
+ getNumberOfAccounts(): int
+ setFirstName(name: String)
+ setLastName(name: String)
+ setBirthDate(bday: Date)
+ setUsername(uname: String)
+ setPassword(pass: String)
+ setEmail(email: String)
+ setUserId(id: int)
+ addCheckingAccount(acct: CheckingAccount)
+ addSavingsAccount(acct: SavingsAccount)
+ removeCheckingAccount(acct: CheckingAccount)
+ removeSavingsAccount(acct: SavingsAccount)
Account
- accountId:int
- balance: double
- annualInterestRate: double
- dateCreated: Date
+ Account()
+ Account(id:int, bal:double)
+ getAccountId(): int
+ getBalance(): double
+ getAnnualInterestRate(): double
+ getDateCreated(): String
+ setAccountId(id:int)
+ setBalance(bal:double)
+ setAnnualInterestRate(air: double)
+ getMonthlyInterestRate(): double
+ getMonthlyInterest(): double
+ withdraw(amount:double)
+ deposit(amount:double)
+ transfer(acct1: Account, acct2: Account, amt: double)
+ setUserId(id: int)
+ getUserId(): int
+ toString(): String
CheckingAccount
- ACCOUNT_TYPE: String = "Checking"
- OVERDRAFT_LIMIT: double = 5000
+ CheckingAccount()
+ CheckingAccount(id: int, amount:double)
+ getAccountType: String
+ withdraw(amount:double)
+ toString(): String
SavingsAccount
- ACCOUNT_TYPE: String = "Savings"
+ SavingsAccount()
+ SavingsAccount(id: int, bal: double)
+ getAccountType(): String
+ withdraw(amount: double)
+ toString(): String



user		
PK	user_id	integer(11)
Key	first_name	varchar(45)
Key	last_name	varchar(45)
Key	birth_date	date
Key	username	varchar(20)
Key	password	varchar(20)
Key	email	varchar(45)

account		
PK	account_number	integer(11)
FK	user_id	integer(11)
Key	balance	integer(20)
Key	account_type	varchar(8)
Key	interest_rate	numeric(3,2)
Key	date_created	date