Department of Computer Science and Engineering

Project Report
Object Oriented Programming Lab(CSE-2112)

Project Name
The Banker's App

Submitted To-

- **DR. MUHAMMAD IBRAHIM, PhD**
- **MR. MD. ASHRAFUL ISLAM**

Submitted By-

- Md. Mahadi Hasan, ROLL-03
- Shamik Shafkat Avro, ROLL-10
- Anika Tabassum, Roll-61

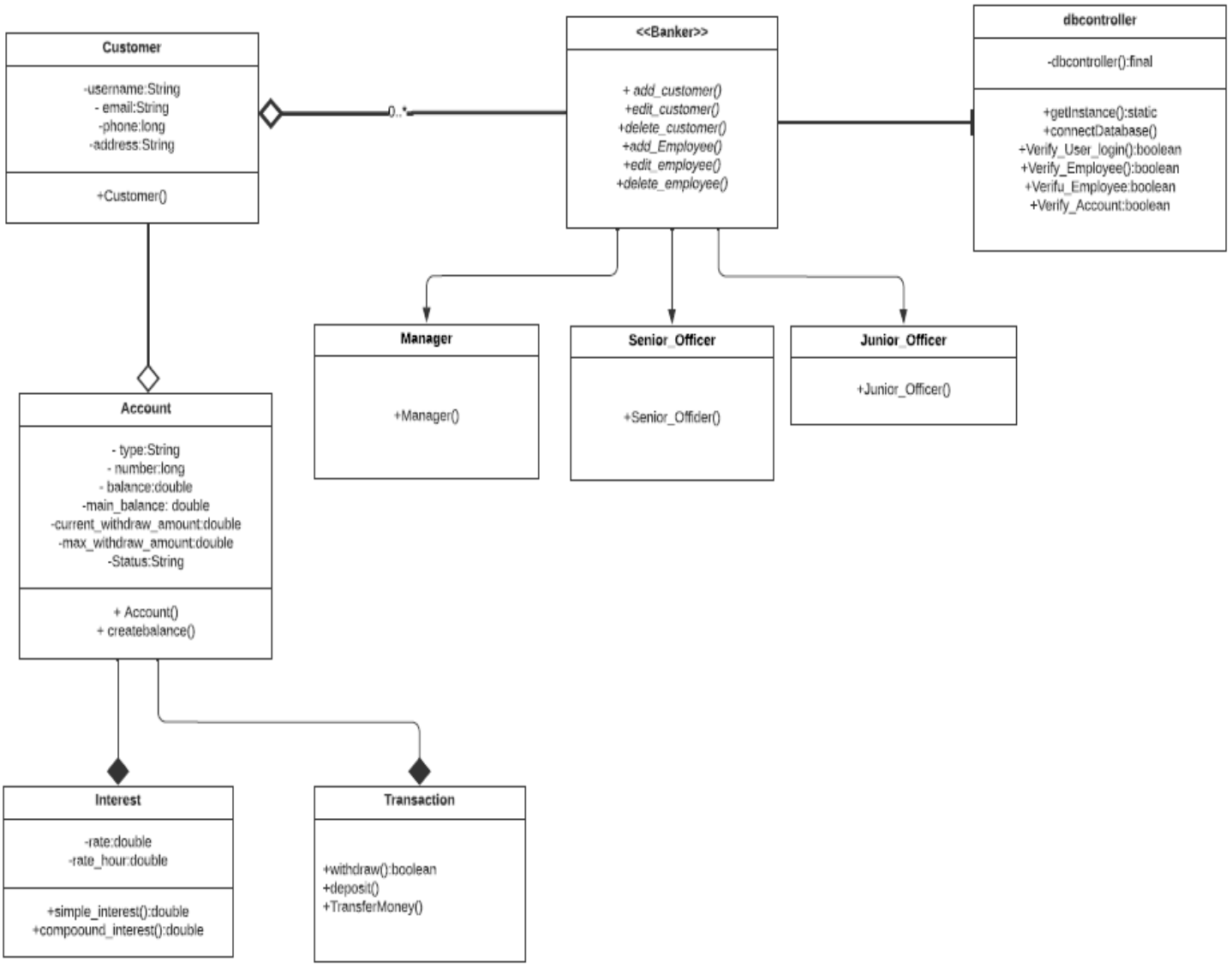
A. Requirement Analysis

Description:

The main purpose of our banking system is to develop an application, which could store bank data and provide an interface for retrieving customer related data accurately and automate all banking processes. Secondly, we look to showcase bleeding-edge industry standards of OOP design principles and implement core parts of the program with as much modularity as possible.

The intended users of our banking systems are for bank employees. Our system will provide a customer to update his privacy, deposit and withdraw money, fund transfer and ask for loans. And for the bank employees to help them to create new accounts for users, loan finalisation, seize an account through any bank employee. Our system provides a secure system which is password protected and only allows the authorised employee to access certain things. The admin may create different levels of permission for the users of the application.

B. System Design:



Class Hierarchy:

The Banker's App contains general banking features like withdraw money, deposit money, and transfer money from one account to another. Any information about customers can be easily searched,

adding new customers or deleting new customers can be done. Manager rank has the most accessibility to use any feature, and he can control other senior and junior employees.

Banker:
Banker is an abstract class where all the information about bank employees are initialised. This class is the super class with "add_customers, edit_customers, delete_customers and add_employee, edit_employee, delete_employee" abstract methods.

Manager:
Manager is a subclass extending Banker where all the features a manager can access is written. A manager can add a new customer, edit any information about a customer, delete any customer, add a new employee, edit information about an employee, delete any employee.

Senior Officer:
Senior officer is a subclass extending Banker where senior officers can add any customer , edit any information customer. If a senior officer tends to delete any customer, or add an employee, or delete an employee, or edit information about an employee , it will generate errors.

Junior Officer:
Junior officer is a subclass extending Banker where a junior officer only has access to check the amount of money, withdraw money, deposit money and transfer funds. If a junior officer tends to access other features, it will generate errors.

Customer:
Customer is a class where information about customers is initialised.

Account:
In this class we have five types of accounts. Current account is for normal users. They can deposit money, withdraw money whenever they want. This account has a fixed rate by the time.

Fixed deposit where a certain amount of money can be kept for a specific time, and one can not withdraw money from this account before the maturation of the account.
Savings account provides compound interest with a fixed rate.
Islamic account where any religious person can keep his money with low interest by abiding islamic rules.
Credit card provides compound interest with a fixed rate less than others.
This class has a major feature
createBalance() : this method creates balance for different types of account and helps to update the amount in database after withdrawing or transferring fund.

Interest:
In this class, rates and time for interest are handled by methods. This class has two types of interest. Simple interest has been used for current accounts. In Fixed Deposit, double interest has been used. And for savings accounts, compound interest has been used.

Transaction:
In this class, three major features have been introduced.

i) Withdraw: Withdrawing money has been handled by this function. A customer can only withdraw money when he has a current account. A fixed deposit account can not withdraw money. A current account can only withdraw a certain amount of money, which is limited by value.

ii)Deposit: Deposit function handles crediting money. A current account can only deposit money, if the account is matured. An immature account or fixed deposit account cannot deposit money.

iii)Transfermoney: A customer can transfer money whether or not it is handled by this function. A fixed deposit account can not transfer or receive money. A customer can transfer money to another account if he has enough money in the current account.

Controller classes:

Controller: This class controls all the action of homepage. Switching scene, logoutbutton, transactionbutton, addaccount, updateaccount, addemployee, deleteemployee, limitmanager all buttons are controlled by this class.All other controller classes extend this class.
AddCustomerController: AddCustomerController is a subclass extending Controller class, controls adding new customers. In this class, the name, address, email, phone number, and all necessary information about a customer are stored. A customer can choose what type of account he wants among, Current account, Savings account, Islamic account, Fixed deposit account or credit card.

AddEmployeeController: this class is subclass extending Controller class, controls adding new employee. In this class all necessary information about employees are stored .

DashboardController: this class controls the actions of homepage, which shows information about all customers and the options one employee can access in table view. This class has introduced generics and collection framework.

DeleteEmployeeController: this class controls the deletion of any employer.

DepositPageController: This class controls the deposit option of any account.

EditAccountController: This class controls the option to edit any information about any customer.

AdminPageController: This page controls the bank employees accounts. This page shows a designated page which an employee can access.

LoginController: This class controls all the login information. If any employee wants to login, he has to enter a username and password provided from the authority. If he uses the wrong username and password, this will generate an error.

TransactionPageController: This page controls all types of transactions. Here three types of transactions are available. By clicking the withdraw button, we can withdraw money. By clicking the deposit button we can deposit money and by clicking the transfer button, we can transfer money from one account to another.

TranferFundPageController: this class controls the options of transferring fund page where by entering the payer account number, receiver account number one can transfer fund.

WithdrawPageController: this page controls the withdrawal money option.

dbcontroller: this class controls all the interactions with the database.

- **getInstance():** To gain modularity of database methods and accessibility of the methods across other classes, we create a static instance of the dbcontroller class. We have followed industry standards so that upgrading database related specifics do NOT break the rest of the system.
- **Verify_User_login():** this method helps to store login information about users, here we have used object of the banker class.
- **Verify_Employee():** this method helps to verify the login about employees if the username belongs to any employee or not.
- **Verify_Account():** this method helps to verify if the account number exists or not and what type of account it is.
- **Update_account():** this method helps to update the amount of any account after withdrawal , deposit or transfer.
- **Edit_Employee():** this method is used for edit information about employee;
- **addCustomer():** this method is used for adding new customer.
- **deleteAccount():** this method is used for deleting any account.
- **addEmployee():** this method is used for adding new employees and updating database.
- **editCustomer():** this method is used when any information about a customer is edited and helps tha database to be updated.

- **deleteEmployee():**this method is used when any employee is deleted and updates database.
- **addTransaction():** this helps to update the database of transactions in trans_table.
- **limitManager():** this method helps to keep information like, interest rate, maximum amount, minimum amount etc. about accounts.

Discussion:

Static and Final:
An instance of the 'dbcontroller' class has been declared as 'static' and 'final' so that we can control the whole project with a single instance of that . There is a 'Banker' attribute in that class. And all the stuff is controlled via that 'Banker' variable throughout the project.

Encapsulation:
We have used encapsulations throughout the whole project. So that two different classes can use the attributes with better control and different classes can change one part of the code without affecting other parts. And this helps to increase security of data.

Abstraction:
Banker is an abstract class, so we can implement methods by inheriting subclasses and we can restrict using Banker objects. By this we have achieved abstraction of Java. This helps to hide certain details and shows only essential information.

Inheritance:
Manager class, Senior officer class and Junior officer class extending Banker class have inherited all abstract methods and common properties. Here we have achieved Inheritance of Java. All the subclasses have inherited abstract methods of their superclass. For example, adding a new customer, Manager class and Senior officer class have the authority but if junior class does not have the authority and if the class tries to access that it will generate errors. So, we have

achieved method overriding to ensure that which class can access what features. This helps to reuse attributes and methods.
Besides in the ‘controllers’ package all the controllers inherited the ‘Controller’ class .Thus we maintained reuse of code as a common method and common ‘Label’, ‘JFXTextField’ are achieved by inheritance.

Polymorphism:
Here the subclasses Manger, Senior officer, Junior Officer have extended their superclass Banker and inherited all the methods and attributes. By polymorphism each class can rewrite the methods and perform different tasks via method overriding . This helps us to reuse codes.

Generics & Collection Framework:
Here we have used generics and collection framework to have the statements in column-row view. We have used this in controller package and in DashboardController class where we can have the information in table view.

Java Exception Handling Mechanism:
Throughout the whole project , all the exceptions are handled to avoid unwanted circumstances.Like, methods of database handle ‘SQLException’ , ‘NullPointerException’. ‘Switch_to_scene’ method in ‘Controller’ class handles ‘IOException’. When any ‘JFXTextField’ requires a number but the user inputs otherwise it generates a ‘NumberFormatException’ and this exception is catched to generate an alert message.

Association,Aggregation and Composition :
This property of OOP is maintained throughout the whole project . For example, ‘Banker’ class has an ‘aggregation’ relation with the ‘Customers’ class with multiplicity 0..*. This is achieved as ‘Customer’ and an ‘Arraylist of Customers’ is an attribute of the ‘Banker’ class.The other relations can be visible in the provided UML Diagram.

Login:
An employee can login to his account by a Username and Password. If he enters any wrong username or password, it will generate an error. a manager , senior officer or junior officer will have their own username, authorised by the bank.

Home page:
Home page is where we have introduced all the features an employee can access. This page shows all the information about all customers with their ID, account number, account type, balance, contact and status of the account. We have global search implemented where we can search for any account by using any keyword.

Add Account:
Add account is used for adding new customers. Only managers and senior officers have access to add new customers. They can add new customers but fill certain information boxes. If any junior employee tries to access this section, he will get an error.

Update account:
A customer can edit/update his account or delete his account. By entering the account number, we can edit information or delete the account. Only a manager can delete any customer, a senior officer can only edit information.

Add Employee:
In this section, new employees can be added. Manager can access this section and new employees.

Update Employee:
In this section, managers can update information about employee.

Delete Employee:
In this section a manager can delete any employee.

Limits Manager:
This section contains information about all types of account, their interest rate, maximum withdrawal amount and minimum amount for opening any new account. This app has five types of accounts, Current account, Savings account, Fixed Deposit, Islamic account and credit card section.

Make Transaction:
This section has three options for customer,
Withdraw money: by entering account number and withdrawal amount, a customer can withdraw money.
Deposit Money: by entering account number and amount, a customer can deposit money.
Transfer Fund: A customer needs to enter the payer account number, receiver account number and the amount to transfer any fund.

Statement:
All transactions are recorded as statements, we have implemented fuzzy search over this list so that any particular transaction may be found with ease.

Edit Password:
An employee can edit his password for privacy issues. Only the user can access this section to update a new password.

Logout:
This section will help one to log out.

Conclusion:
This is the first time we have developed a java application, where we have built a prototype of a professional banking app using oop and JavaFx library. We have tried to introduce all principles of oop like, encapsulation, inheritance, polymorphism. We tried to develop an application with better user interference but we failed to achieve this fully and experienced many hurdles. We had very little knowledge about using oop in the database, and we failed to apply our ideas

because of it. We had very little time to establish all of our ideas, that is why we have failed to introduce loan section in our project.

However, we have learned a great deal through this project. We have learned to use principles of oop, design principles, DRY principles. We have gained experience about app development. We have learnt to handle problems and find solutions ourselves.

Future Plan:

We couldn't develop the loan section for the time being, we want to work on that. We want to upgrade our user interference to make it look more graphically attractive. Our future plan is to make it usable for network users. We would like to make this app more reliable for everyone. Besides all these, we would like to improve the basic knowledge achieved in this project and make a better and stronger project in the future.