Object Design Document

Team 18 members:

Paul Menexas	**U78320172**	pmenexas@bu.edu
Abbireddy, BhaskarDurgaVeeraVenkata Ganga Raju	**U89625488**	bhaskar9@bu.edu
Shweta Baindur	**U73181758**	shwetab@bu.edu

Classes:

1) **Account** - This is an abstract class which is the super class of the checking, savings and securities account classes.
2) **Bank** - This class contains the specific attributes which the bank manager can update
3) **BankController** - This class is used to implement the Singleton design pattern for the ease of creating one instance of specific classes that can be used during the program.
4) **Banker** - This is the banker class which extends the User class and contains the data members and methods for the banker manager.
5) **CanadianDollar** - This class implements the Fiat class and is used to display the balances in Canadian dollars.
6) **CheckingAccount** - This class is the subclass of the Account class and contains all the methods related to withdrawing and depositing money from a clients checking account.
7) **CheckingAccountFactory** - used to implement the factory design pattern for the CheckingAccount class.
8) **Client** - this class extends the User class and contains the members and instantiates the client object.
9) **Database** - This class is used for connecting to the mysql database and for the sql jdbc commands.
10) **Euro** - This class implements the Fiat class and is used to display the balances in Euro.
11) **Loan** - Contains data members related to the loans which a client requested.
12) **Main** - contains the main method from where the program starts.
13) **NullAccount** - This is the NullAccount class which extends the Account class and is used to implement the null design pattern which handles situations for when a client does not have an open account of any specific type yet.
14) **Position** - contains all methods and members related to the position of a stock.
15) **SavingsAccount** - This class is the subclass of the Account class and contains all the methods related to withdrawing and depositing money from a clients savings account.

16) **SavingsAccountFactory** - used to implement the factory design pattern for the SavingsAccount class.
17) **SecuritiesAccount** - this is the SecuritiesAccount class which contains the members and methods for trading stocks.
18) **Stock** - contains the stock data members
19) **StockMarket** - Contains the stock market data members and methods.
20) **StockPrice** - Contains data members related to the stock prices in the market.
21) **Transaction** - Contains data members related to the transaction history of a client.
22) **USDollar** - This class implements the Fiat class and is used to display the balances in US dollars.
23) **User** - User is the superclass of both Banker and Client and contains data members that are needed across both types of users, like username and password.
24) **Utils** - Helper class which contains helper functions.

Interfaces:

1) **Depositable** - This interface is implemented by the CheckingAccount and SavingsAccount classes and contains the method in which the client can deposit money to the bank.
2) **Fiat** - This interface is implemented by the USDollar, CanadianDollar and Euro classes.
3) **Withdrawable** - This interface is implemented by the CheckingAccount and SavingsAccount classes and contains the method in which the client can withdraw money from the bank.
4) **AccountFactoryCreator** - This interface is used to implement the factory design pattern.

Database connection:

We used a MySQL database for persistent storage and connected it to our Java program through the help of the mysql JDBC driver.

User Interface:

We used the Java Swing GUI Framework to create window based application interfaces to showcase our project.

Design Patterns and why we chose them in our design:

We used the creational design patterns - factory and singleton, the null design pattern and the structural design pattern - facade pattern.

1) Null pattern:

We used the null design pattern to gracefully handle the situations where a client of the bank does not have an open account yet. The abstract class Account.java was inherited by the NullAccount subclass and an object of this class was then instantiated whenever a client did not have any open account.

2) Facade Pattern:

We used a facade pattern in our design as it abstracts the client from the complex functionalities that the client can do. For example, we have an Account, Loan and Transaction class where the methods of these classes are used by the client to do specific functions. In the Client class, we used objects of these specific classes as data members for an easier way of implementing these functions as well as to provide an abstraction to the client.

3) Factory Pattern:

The factory design pattern was implemented for each specific type of Account class, specifically the Checking Account and Savings Account classes. For each type of account class we have a CheckingAccountFactory and SavingsAccountFactory class which implements the interface AccountFactoryCreator. Depending upon which account is created at that point in time, an object of that specific type of account is instantiated and used.

4) Singleton Pattern:

We used a singleton pattern to make sure that a class only had one instance of a specific class. This pattern ensured that we did not have multiple objects of the same class when not required in our code and this avoided redundant object creation. In our project, the BankController class ensures that only one instance of its object as well as one instance of the Client, Banker, Bank, Database and StockMarket is created as these objects need not be instantiated multiple times.

Object GUI Relationship:

With the help of the implementation of the Singleton pattern in the BankController class, we were able to interface the Swing UI with the java code. In the runController() function present in the class, an object of the LoginPage UI is instantantiated.
This runController() method is then called in the Main class to start the running of our program.

Benefit of our design:

The usage of the null, facade, singleton and factory design patterns in our code make this project more extendable for any future work and objects can be reused easily. We implemented object oriented practices to the best of our ability and also used interfaces when needed and inheritance to achieve abstraction and code reusability.
To sum up, we believe that the design we chose is sound. We were able to implement all features provided in the requirements such as the bank manager retrieving a daily report, there are no bugs that we are aware of, the data persists even if the system restarts at any time, our object model is strong and robust, and the user interface is easy to use and aesthetic.