Object Design Document

> # Background
>
> This is a GUI-based Bank ATM system project focusing on **Java**, **Java Swing**, **object-oriented design**, **design patterns** and **teamwork**. There are **3** components which incorporates **66** classes in total, thus rendering the whole project of high **scalability** and **extendability!**
>
> Our object model is very **strong** and **robust**. It can self-handle various legal and illegal user operations **without crashing**. When the system is reloaded, all **user** and **account** data will be automatically and seamlessly read from the **database**.
>
> In this document, we will explain the **object design choices**, **object model**, **Object and GUI relationship**, and **design drafts** of our projects in detail.
>
> For project running examples, general workflow of the project, how to compile and run it on your machine, highlights of this project, and noteworthy things, please do refer to our **ReadMe** document.
>
> Project Developers: **Dayong Wu** and **Tian Yu**.

Object Design Choices and Object Model

Since we need to build a system that is both **strong** and highly **scalable**, we really need to analyze and design our object model carefully. From our perspectives, there are **THREE** strategies to safely construct a robust and highly OO object model:

1. Extract every possible and potential "thing" (i.e. object) in the project to reduce the

dependences between modules.

2. Separate the data read-write processes from the main project logics (and the objects).
3. Separate the **backend** functions from **frontend** ones (i.e. **never mix backend with frontend!**). We make each GUI window as a class, and call relevent methods of the objects together with launch data reads-and-writes in the button listener classes of it (i.e. the specific GUI window class).

Based on that, we have formed a clear blueprint of the object design of this project. We have implemented **THREE** main components to handle different types of requests throughout the project, namely **BankAttributes**, **Data**, and **GUI**. They exist as **packages** in Java.

- **Package BankAttributes** contains all the base classes of the project, such as Manager.java, Customer.java, Currency.java, etc. Each class represents an essential object of the project. For instance, the project will definitely involve a manager and receive a lot of customers duing its runtime, thus rendering a manager class and a customer class a must.
- **Package Data** contains all the classes that are related to data reads-and-writes. For example, when the player of the project creates a new customer, relevent data (e.g. username and password) should be written into the database in order to persist all the data from last run.
- **Package GUI** is responsible for managing all the elements of the GUI windows of the project. It serves as a "**middleman**" to connect both itself and the previous two packages!

Allow us to describe the customer login scenario to better your understanding. For example, in the GUI customer login window (i.e. **GUICustomerLogin.java**), once the player has entered the correct username and password and hit the "login" button, the program will automatically launch the **read method** in **CustomerDataManager.java** of the **Data Package** and read all the account information of that particular customer. Afterwards, a customer object will be created by the **Customer.java** class of the **BankAttributes Package** and instantiated by the corresponding information just read.

In this way, we have successfully created a pre-existing customer "from scratch"!

But in fact, this is a brand new customer object that is just created after the player hits the login button. It is just that the newly created customer has all the relevent data of a particular record in our customer database.

This is the **magic** of our aforementioned **THREE** strategies to ensure a robust and highly scalable (and extendable) object model.

We will explain the purposes of each class in the **THREE** components (i.e. packages) in the next part.

Classes of the Project (Recommended Viewing Order)

==> BankStarter.java

- This is the starter of the **ENTIRE** project!

- Only this class has the main() method and it is very concise.

==> Package BankAttributes：

1. Person.java

- This class has the attributes of a person.

2. User.java

- This class stands for a user (i.e. the bank manager or a customer).

3. Manager.java

- This class holds all operations of the bank manager (i.e. our professor ^).
- It extends from User since a customer is also a user.

4. Customer.java

- This class holds all operations of a customer.
- It extends from User since a customer is also a user.

5. Account.java

- This class manages all the possible actions of a bank account such as deposit, withdraw, etc.

6. Checking.java

- This class represents a Checking account, which is also an account.

7. Savings.java

- This class represents a Savings account, which is also an account.

8. Securities.java

- This class represents a Securities account (for stock buying or selling), which is also an account.

9. AccountFees.java

- This class represents the account opening fee that the bank manager can charge when a new account opened.
- Because the manager can adjust the account opening fee at any time, we need a specific class to hold all fees in history together with their set time.
- This class is associated with [BossDataManager.java] in the [Data package].

10. Collateral.java

- This class denotes the collaterals of the customers' loans.

11. Currency.java

- This class is the top class of all types of currencies.

12. CurrencyExchangeBehavior.java

- This class manages the algorithm of exchanging one type of currency to others.
- This is the implementation of Strategy Pattern!

13. CNYExchangeBehavior.java

- This class manages the algorithm of exchanging CNY to other types of currencies.
- This is the implementation of Strategy Pattern!

14. HKDExchangeBehavior.java

- This class manages the algorithm of exchanging HKD to other types of currencies.
- This is the implementation of Strategy Pattern!

15. USDExchangeBehavior.java

- This class manages the algorithm of exchanging USD to other types of currencies.
- This is the implementation of Strategy Pattern!

16. CNY.java

- This class represents the CNY (Chinese Yuan, i.e. ¥) currency.

17. HKD.java

- This class represents the HKD (Hong Kong Dollar, i.e. HK$) currency.

18. USD.java

- This class represents the USD (US Dollar, i.e. $) currency.

19. Transaction.java

- This class holds all attributes and print methods of a single transaction.

20. Stock.java

- This class represents the attributes of a stock.

21. Loan.java

- This class represents the loan that can be applied and repaid by a customer.

22 Interest.java

- This interface contains the interest rate of all types of transactions.

23. IDCenter.java

- This class is responsible for generating a random ID for an account.

==> Package Data：

1. DataManager.java

- This is a generic class that holds the data read from the database
- @param The type of data read from the database (i.e. Account/Customer/AccountFees).

2. CustomerDataManager.java

- This class is responsible for managing all the data read & write from the customer database,
  i.e. [Customers.csv].

3. BossDataManager.java

- This class is responsible for managing all the data read & write from the boss database, i.e. [Fees.csv].

4. AccountDataManager.java

- This class is responsible for managing all the data read & write from the account database,
  i.e. [Accounts.csv].

==> Package GUI：

1. GUIHomepage.java

- This class serves as the entrance of the entire project.
- This class would initiate a homepage window for user to choose either "Customer Login" or "Manager Login."

2. GUIManagerLogin.java

- This class serves as the Login window for Manager.
- It provides "login" and "back" options for user, either login to manager window or back to homepage.

3. GUIManagerWindow.java

- This class serves as the Manager Homepage window.
- It provides "Check Bank Status" and "Adjust Bank Fees" options for manager. Manager could go for homepage of either checking specific details of current Bank or adjusting service fee of Bank.

4. GUIManagerAdjustWindow.java

- This class serves as the Adjust-Fee Homepage window for manager.
- Manager could choose "adjust Open-Account Fee" to go to the specific page of "Adjust Open-Account Fee".

5. GUIManagerCheckWindow.java

- This class serves as the Check-Status Homepage window for manager.
- Manager could choose which part she/he is interested in, such as "Daily Report", "Users Details", or "Bank Revenue".

6. GUIAdjustOFWindow.java

- This class serves as a functional page for manager to adjust open-account fee

7. GUICheckCustomersWindow.java

- This class serves as an informative page for manager to view all users details with their transaction history
- Manager could search a specific username for his/her personal transaction history

8. GUICheckReportWindow.java

- This class serves as an informative page for manager to view Bank's today report.

9. GUICheckRevenueWindow.java

- This class serves as an informative page for manager to view the revenue gained from all active accounts.

10. GUICustomerLogin.java

- This class serves as the Login window for customers.
- If user doesn't have an account yet, user could choose "Customer Signin" option then go to the Sign-in Window.
- If user already has an account, then he/she could login to the homepage for customer.

11. GUICustomerSignin.java

- This class serves as the Sign-in window for customers.

12. GUICustomerWindow.java

- This class serves as a homepage for customers.
- It provides three functional options: "Saving Account", "Checking Account" and "Open Accounts".
- Customers could choose to go for "Saving Account" or "Checking Account" as long as they have opened corresponding account(s).
- Customers could log-out to Customer Login page as long as they want.

13. GUIOpenAccountWindow.java

- This class serves as a functional page for customer to open an account that he/she doesn't have one yet.
- It provides two options: "Open Saving Account" and "Open Checking Account". The option is playable only if the customer doesn't have one such account.

14. GUICustomerAccountWindow.java

- This class serves as a parent class of "GUICheckingWindow" and "GUISavingWindow".
- Due to some same UI designs but a few different details, we put most UI designs in this class and leave button listeners in child classes.
- It provides "Deposit", "Withdraw", "Send", "Loans", and "View Transaction History" for both Checking and Saving Accounts. Customer could choose each one of this button
  to go to its corresponding functional page.
- Customer could also choose the option "Close Account" to close current account.

15. GUISavingWindow.java

- This class serves as a homepage for saving account of customers.
- Beside the functions inherited from parent class "GUICustomerAccountWindow", it also
  provides "Stock" function for customers who have balance over $5000 in saving account.
  When the customer is qualified and willing to enter the stock market, system will

automatically help she/he buy $1000 shares, and then immediately ask the customer to do a series of decisions before selling all shares he/she holds.

16. GUICheckingWindow.java

- This class serves as a homepage for checking account of customer.
- This class has all inherited functions from paren class "GUICustomerAccountWindow" but
  have different action listeners.

17. GUIDepositWindow.java

- This class serves as a parent class of "GUICheckingDepositWindow" and "GUISavingDepositWindow".
- Due to some same UI designs but a few different details of implementation, we put most
  UI designs in this class and leave action listeners to be implemented in child classes.

18. GUICheckingDepositWindow.java

- This class serves as a functional page for customer to deposit to his/her checking account.
- Most UI designs are inherited from parent class "GUIDepositWindow", and this class specifies
  the depositing details in checking account.

19. GUISavingDepositWindow.java

- This class serves as a functional page for customer to deposit to his/her saving account.
- Most UI designs are inherited from parent class "GUIDepositWindow", and this class specifies
  the depositing details in saving account.

20. GUIWithdrawWindow.java

- This class serves as a parent class of "GUICheckingWithdrawWindow" and "GUISavingWithdrawWindow".
- Due to some same UI designs but a few different details of implementation, we put most
  UI designs in this class and leave action listeners to be implemented in child classes.

21. GUICheckingWithdrawWindow.java

- This class serves as a functional page for customer to withdraw from his/her checking account.
- Most UI designs are inherited from parent class "GUIWithdrawWindow", and this class specifies the withdrawing details in checking account.

22. GUISavingWithdrawWindow.java

- This class serves as a functional page for customer to withdraw from his/her saving account.
- Most UI designs are inherited from parent class "GUIWithdrawWindow", and this class specifies the withdrawing details in saving account.

23. GUISendWindow.java

- This class serves as a parent class of "GUISavingSendWindow" and "GUISavingSendWindow".
- Due to some same UI designs but a few different details of implementation, we put most
  UI designs in this class and leave action listeners to be implemented in child classes.

24. GUISavingSendWindow.java

- This class serves as a functional page for customer to enter a target account number, and
  an amount of money to transfer from his/her saving account to target account.
- Most UI designs are inherited from parent class "GUISendWindow", and this class specifies the transfer details from saving account.

25. GUICheckingSendWindow.java

- This class serves as a functional page for customer to enter a target account number, and an amount of money to transfer from his/her checking account to target account.
- Most UI designs are inherited from parent class "GUISendWindow", and this class specifies the transfer details from checking account.

26. GUILoanWindow.java

- This class serves as a parent class of "GUISavingLoanWindow" and "GUICheckingLoanWindow".
- Due to some same UI designs but a few different details of implementation, we put most UI designs in this class and leave action listeners to be implemented in child classes.
- This class is a homepage for customer to do any actions on loans.
- It provides two options: "Apply for a Loan" and "Repay the Loan" for customers. Customer could go for the "apply-loan" page or "repay-loan" page.
- Customer could open "repay-loan" page only if he/she has a loan to be repaid.

27. GUISavingLoanWindow.java

- This class serves as a homepage for customer to do any action on a loan on his/her saving account.
- Most UI designs are inherited from parent class "GUILoanWindow", and this class determines the feasibility of "apply-loan" and "repay-loan" on his/her saving account.

28. GUICheckingLoanWindow.java

- This class serves as a homepage for customer to do any action on a loan on his/her checking account.
- Most UI designs are inherited from parent class "GUILoanWindow", and this class determines the feasibility of functions "apply-loan" and "repay-loan" on his/her saving

account.

29. GUIApplyLoanWindow.java

- This class serves as a parent class of "GUISavingApplyLoanWindow" and "GUICheckingApplyLoanWindow".
- Due to some same UI designs but a few different details of implementation, we put most
  UI designs in this class and leave action listeners to be implemented in child classes.
- This class is a functional page for customer to apply a loan.
- It asks customer to input a collateral and loan amount in order to finish the application.

30. GUISavingApplyLoanWindow.java

- This class serves as a functional page for customer to apply a loan on his/her saving account.
- Most UI designs are inherited from parent class "GUICustomerApplyLoanWindow", and this class
  determines the function "apply-loan" on his/her saving account. The customer is able to apply
  for a loan only if he/she doesn't have a loan on saving account yet.

31. GUICheckingApplyLoanWindow.java

- This class serves as a functional page for customer to apply a loan on his/her checking account.
- Most UI designs are inherited from parent class "GUICustomerApplyLoanWindow", and this class
  determines the function "apply-loan" on his/her checking account. The customer is able to apply
  for a loan only if he/she doesn't have a loan on checking account yet.

32. GUIRepayLoanWindow.java

- This class serves as a parent class of "GUISavingRepayWindow" and "GUICheckingRepayWindow".
- Due to some same UI designs but a few different details of implementation, we put

most
UI designs in this class and leave action listeners to be implemented in child classes.

- This class is a functional page for customer to repay the loan that customer has.
- It asks customer to input a repay-amount to repay his/her loan. Customer could repay his/her
loan with any amount but not exceeding total repay amount.

33. GUISavingRepayWindow.java

- This class serves as a functional page for customer to repay the loan of his/her saving account.
- Most UI designs are inherited from parent class "GUIRepayLoanWindow", and this class
determines the function "repay-loan" on his/her saving account.

34. GUICheckingRepayWindow.java

- This class serves as a functional page for customer to repay the loan of his/her saving account.
- Most UI designs are inherited from parent class "GUIRepayLoanWindow", and this class
determines the function "repay-loan" on his/her checking account.

35. GUIHistoryWindow.java

- This class serves as a parent class of "GUISavingHistoryWindow" and "GUICheckingHistoryWindow".
- Due to some same UI designs but a few different details of implementation, we put most
UI designs in this class and leave action listeners to be implemented in child classes.

36. GUISavingHistoryWindow.java

- This class serves as an informative page for customer to view his/her transaction history
including stock buy/sell details(if has one) in saving account.
- Most UI designs are inherited from parent class "GUIHistoryWindow", and this class

specifies transaction history details in saving account.

37. GUICheckingHistoryWindow.java

- This class serves as an informative page for customer to view his/her transaction history in
checking account.
- Most UI designs are inherited from parent class "GUIHistoryWindow", and this class specifies transaction history details in checking account.

38. IconUtil.java

- This class stores all icons there in order to be used in GUI design classes.

Object and GUI Relationship

As a matter of fact, the relationship between GUI and all the objects has already been introduced (in detail) in the first part of this document.

The **GUI Package** serves as a **"middleman"** to connect itself to the other two packages (i.e. **objects** and **data**). We think this is a very reasonable way of system design since we will only launch the login function of a customer and load all her/his account information after the "login" GUI button is clicked.

Benefits of Our Object Design

The **BIGGEST** benefit of our object design strategy is **FAST**!

For a bank manager, time is gold, time is money! As the development group hired by the bank manager, time is the ultimate pursue of both our program running speed and our development speed!

By implementing the aforementioned **THREE** components, no matter we are in which GUI class, we can easily build connections to our database to retrieve and inject data to our newly created object from the base class component.

Another huge benefit of our object design strategy is **FOOLPROOF**!

Since we have modularized every possible and potential object from the project, it is very straightforward for us to trace bugs. For instance, if the **deposit** feature of the customers is anomalous, we can quickly pinpoint the **deposit()** method of the **Customer.java** class (**BankAttributes Package**) and relevent **write()** method of **CustomerDataManager.java** class (**Data Package**).

Work Division of Group Members

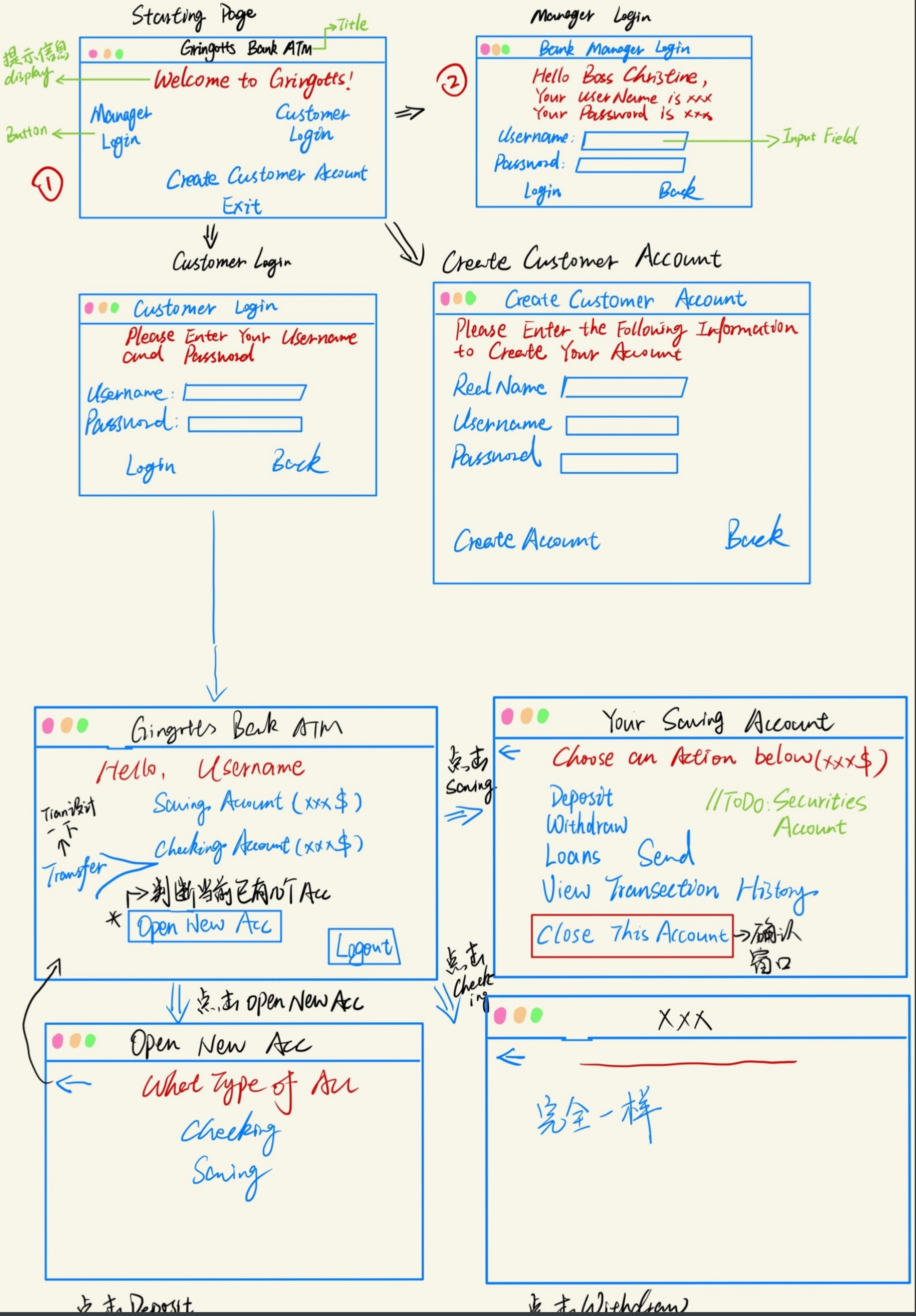
We feel very lucky that the two of us are both very **responsible** and **prompt**!

- **Tian Yu** is responsible for developing all **frontend** features (i.e. all classes in the **GUI Package**) and implementing all the colorful and graphic elements to our project.
- **Dayong Wu** is responsible for developing all **backend** features (i.e. all classes in the **BankAttributes Package** and **Data Package**) and relating them to their relavent GUI classes (i.e. **Object and GUI Relationship!**)
- Both of us have participated in the development of the **stock feature** of our project.
- Both of us have participated in the writing of both this document and the **ReadMe** file.

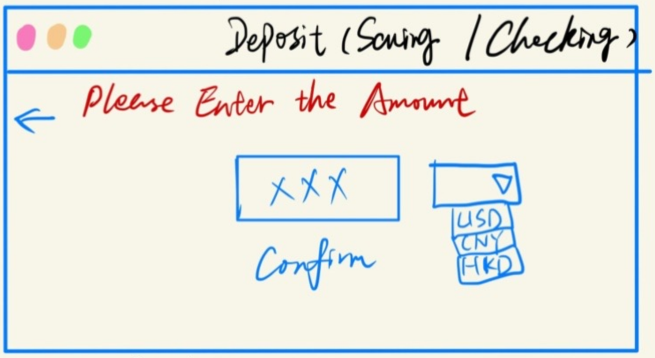
We also play as program testers for each other, which is very helpful for eliminating bugs! We all know "**those closely involved cannot see clearly**", therefore this is a very good practice for us to grow more experiences on detecting bugs.

Appendix

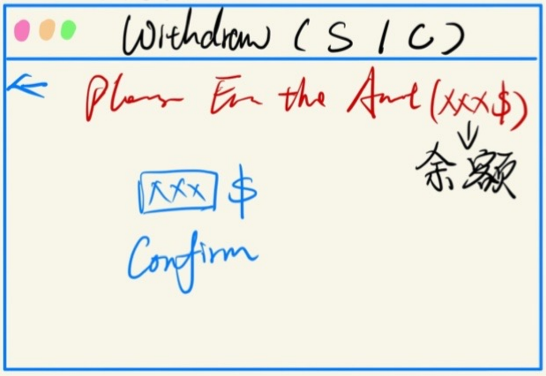
We are very proud of some of our **GUI design drafts** for this project and we think they deserve a "booth" in this document :)


Starting Page
Title
Gringotts Bank ATM
提示信息
display
Welcome to Gringotts!
Button
Manager
Login
Customer
Login
Create Customer Account
Exit
①
⇒
②
Manager Login
Bank Manager Login
Hello Boss Christine,
Your UserName is xxx
Your Password is xxx
Username:
Password:
Input Field
Login
Back
⇓
Customer Login
Customer Login
Please Enter Your Username
and Password
Username:
Password:
Login
Back
Create Customer Account
Create Customer Account
Please Enter the Following Information
to Create Your Account
Real Name
Username
Password
Create Account
Back
Gringotts Bank ATM
Hello, Username
Saving Account (xxx$)
Checking Account (xxx$)
Tian设计一下
Transfer
判断当前已有哪个Acc
* Open New Acc
Logout
点击 open New Acc
点击
Saving
Your Saving Account
←
Choose an Action below(xxx$)
Deposit
Withdraw
Loans
Send
View Transection History
//ToDo: Securities
Account
Close This Account
确认
窗口
点击
Checking
Open New Acc
←
What Type of Acc
Checking
Saving
XXX
←
完全一样
点击Deposit
点击Withdraw

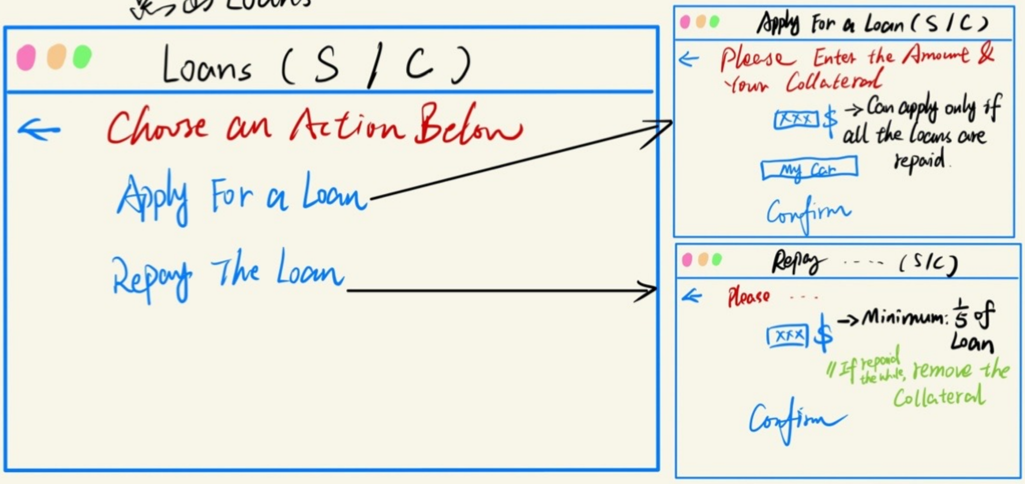
点击Deposit



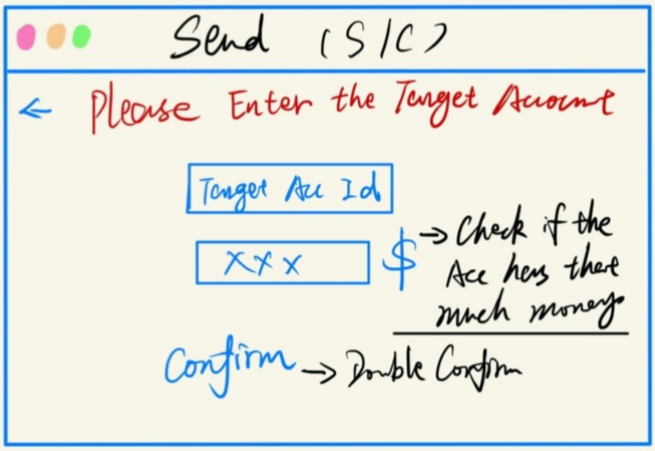
点击Withdraw



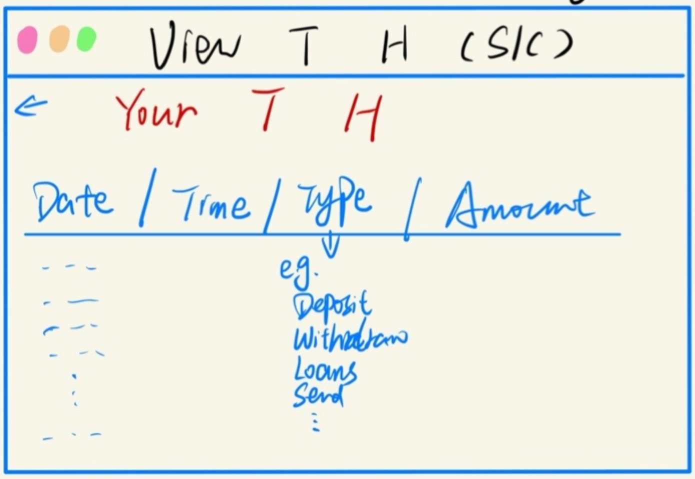
点击 Loans



点击 Send


Send (S/C)
←
Please Enter the Target Account
Target Acc Id
XXX $
→ Check if the Acc has that much money
Confirm → Double Confirm

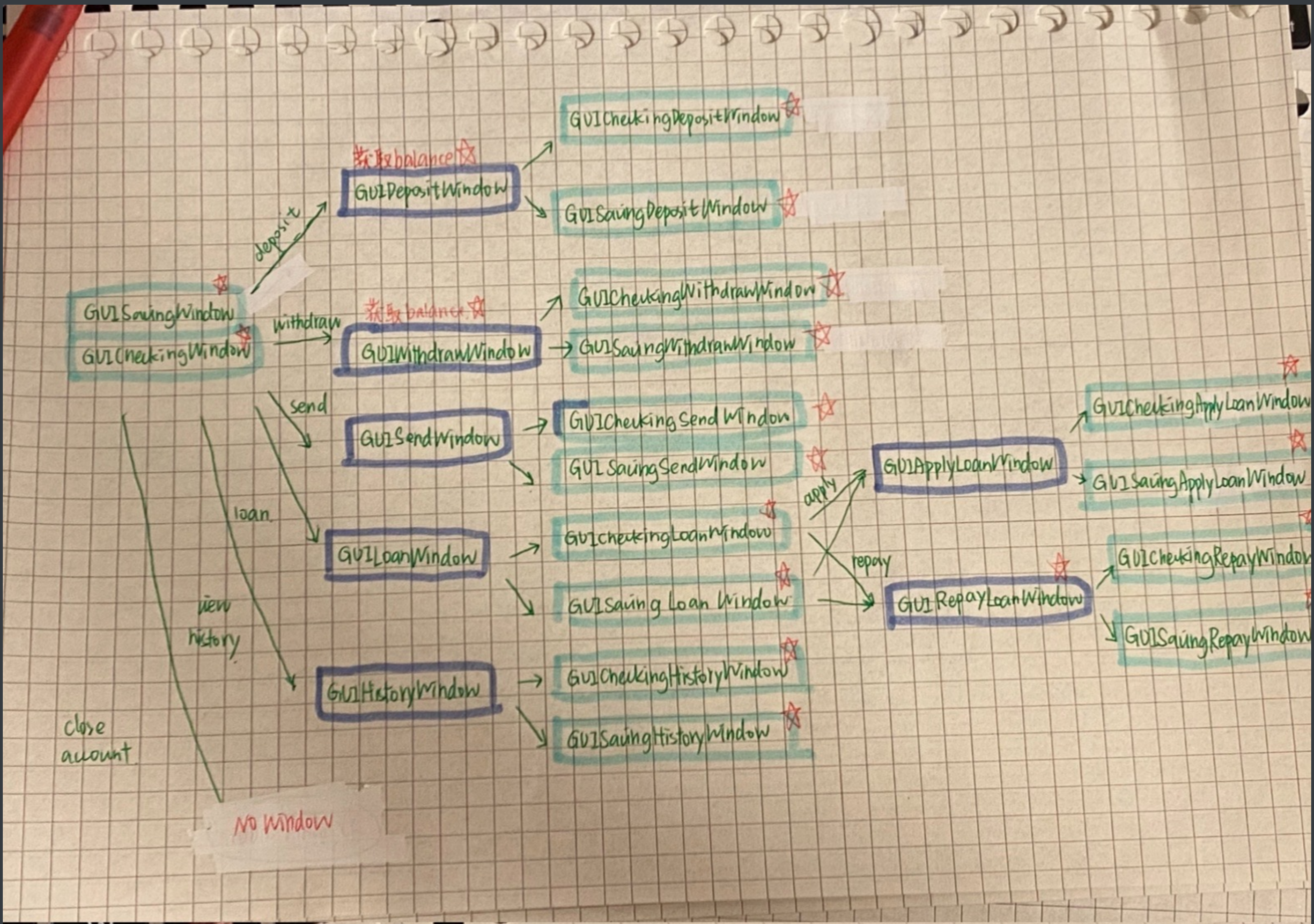
点击 View Transaction History

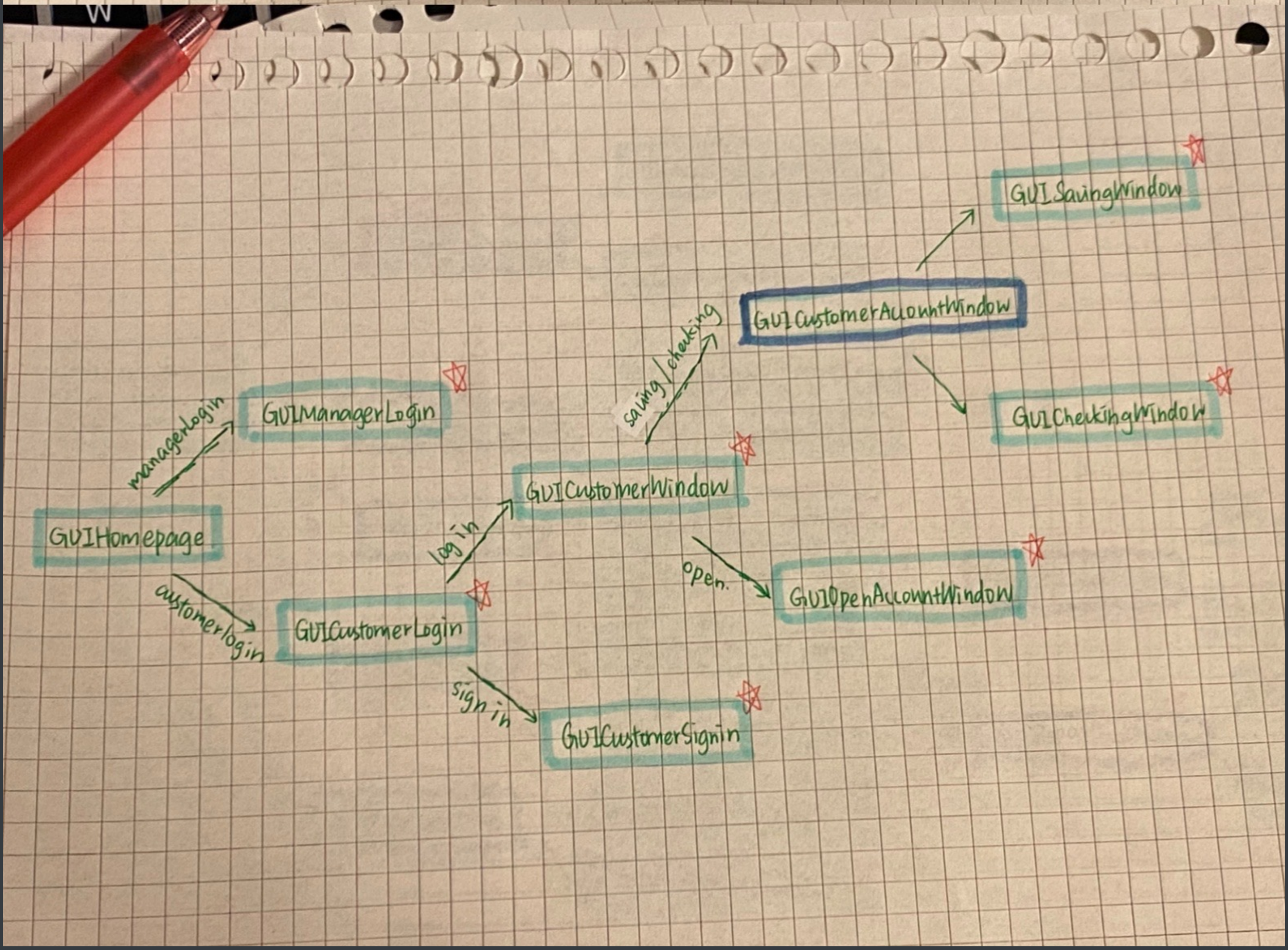


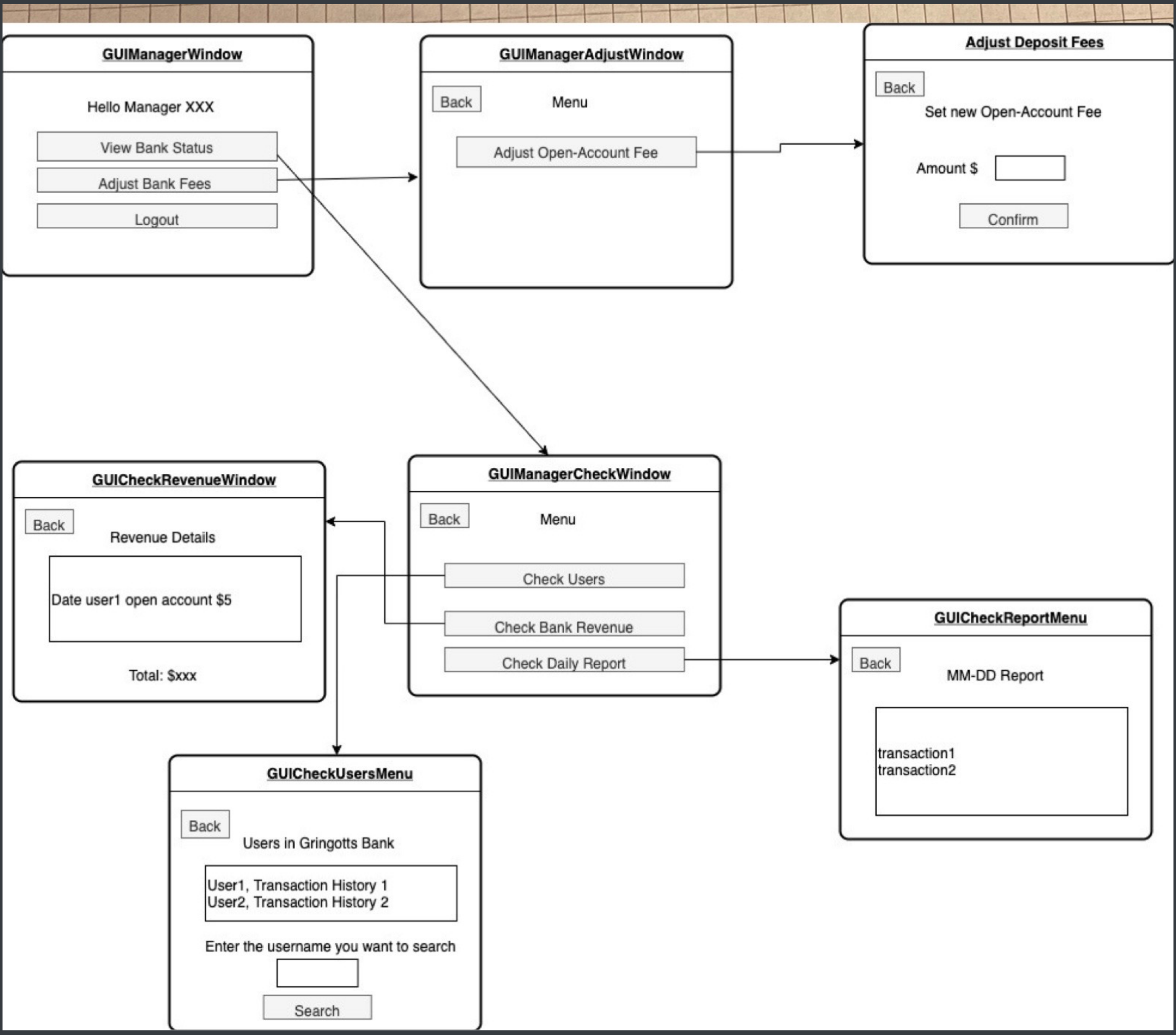
Milestone	
28 Apr – 29 Apr	
30 Apr – 1 May	Manager
2 May – 3 May	Stock & Refine
4 May	Documentation

→ Above Concludes All Actions of a Customer EXCEPT For STOCK Feature.

(Challenge: Customer Applies For a Loan
↓
Manager Approves / Declines
↓
Return the Result To Customer)


GUISavingWindow
GUICheckingWindow
deposit
withdraw
send
loan
view history
close account
No Window
获取balance
GUIDepositWindow
GUIWithdrawWindow
GUISendWindow
GUILoanWindow
GUIHistoryWindow
GUICheckingDepositWindow
GUISavingDepositWindow
GUICheckingWithdrawWindow
GUISavingWithdrawWindow
GUICheckingSendWindow
GUISavingSendWindow
GUICheckingLoanWindow
GUISavingLoanWindow
GUICheckingHistoryWindow
GUISavingHistoryWindow
apply
repay
GUIApplyLoanWindow
GUIRepayLoanWindow
GUICheckingApplyLoanWindow
GUISavingApplyLoanWindow
GUICheckingRepayWindow
GUISavingRepayWindow


GUIHomepage
managerlogin
GUIManagerLogin
customerlogin
GUICustomerLogin
login
sign in
GUICustomerSignin
GUICustomerWindow
saving/checking
open
GUICustomerAccountWindow
GUIOpenAccountWindow
GUISavingWindow
GUICheckingWindow



Group 5 [Tian Yu, Dayong Wu] – 4 May 2021