My Fancy Bank – Gringotts Bank ATM

Background

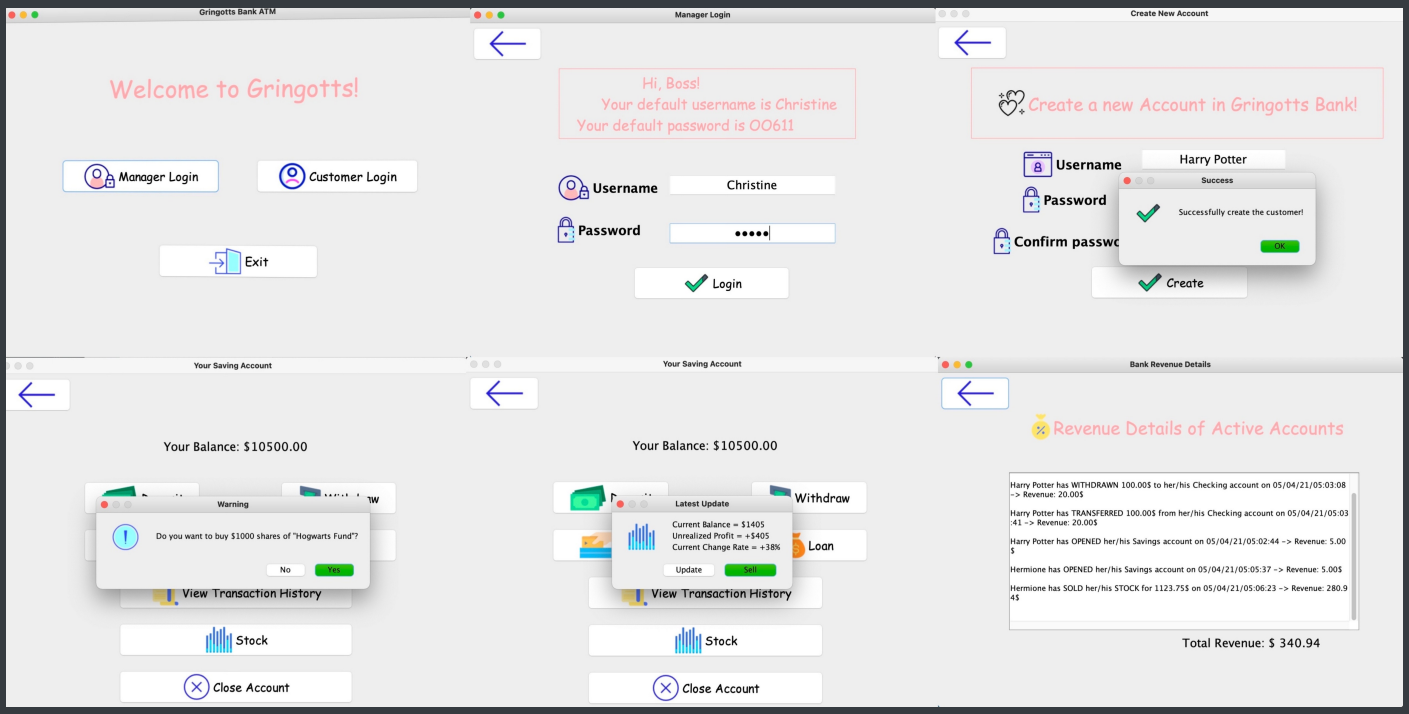
This is a GUI-based Bank ATM system project focusing on **Java**, **Java Swing**, **object-oriented design**, **design patterns** and **teamwork**. There are **3** components which incorporates **66** classes in total, thus rendering the whole project of high **scalability** and **extendability!**

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**.

For the illustrations of our **object design choices**, **object model**, **Object and GUI relationship**, and our **design drafts**, please refer to our **Object Design Document**.

Project Developers: **Dayong Wu** and **Tian Yu**.

Project Running Example



General Workflow of the Project

- There are two types of user interfaces, namely **manager login** and **customer login**.
- **Customer login** interface is designed for bank customers to **open checking and savings accounts**. If a customer has more than **$5000** in her/his savings account, (s)he can choose to enter the **stock market** (i.e. open a **securities account**).
- The common functions between **checking** and **savings** accounts are: **deposit, withdraw, send (i.e. transfer), apply a loan, repay a loan, view transaction history, and close the account**.
- In the **stock market**, the customer can keep updating the stock to view the up-to-date **current change rate**, which will affect her/his **unrealized profit** and current stock balance. After the customer chooses to sell the stock, the system will add the customer's **stock balance** to her/his **savings account balance** (after being charged a certain amount of service fee of course).
- **Manager login** interface is designed for the boss of the bank (i.e. **our professor!**) to check all her customers. The system can generate a report of all customers' operations to the boss. The boss can also check a specific customer by just entering her/his name.
- The bank manager can also get a daily report of all customers' operations. The program will automatically detect the date of today.
- Additionally, there is also a "**revenue history panel**" built for our boss to check her

revenue history. There also demonstrates a number showing the total revenue that our boss has made so far!

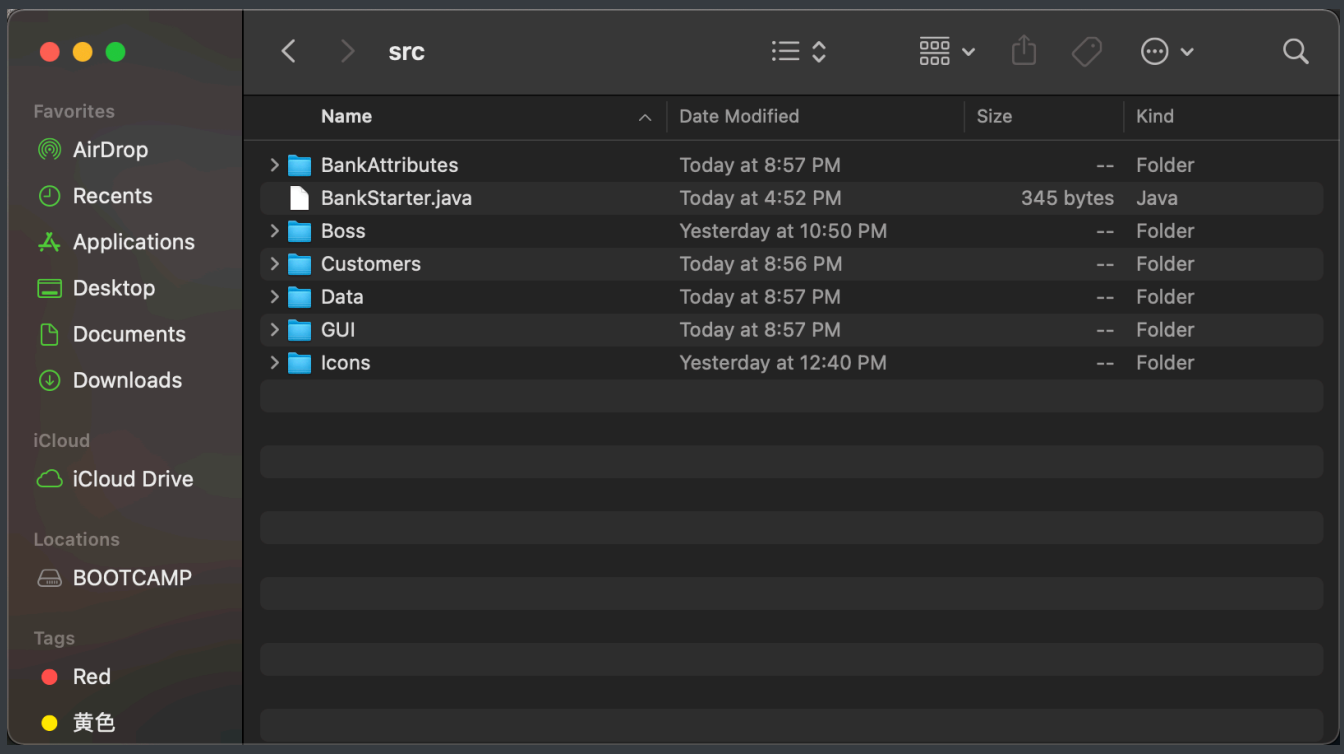
- Another power of the boss is adjusting the bank openning fee. Our professor can change the fee that the customer will be charged when (s)he opens a new checking or savings account.

Classes of the Project

Pease read our **Object Design Document** for all the details of the **classes** and the **object model** of this project.

Instructions on how to compile and run the program via [Mac Terminal]

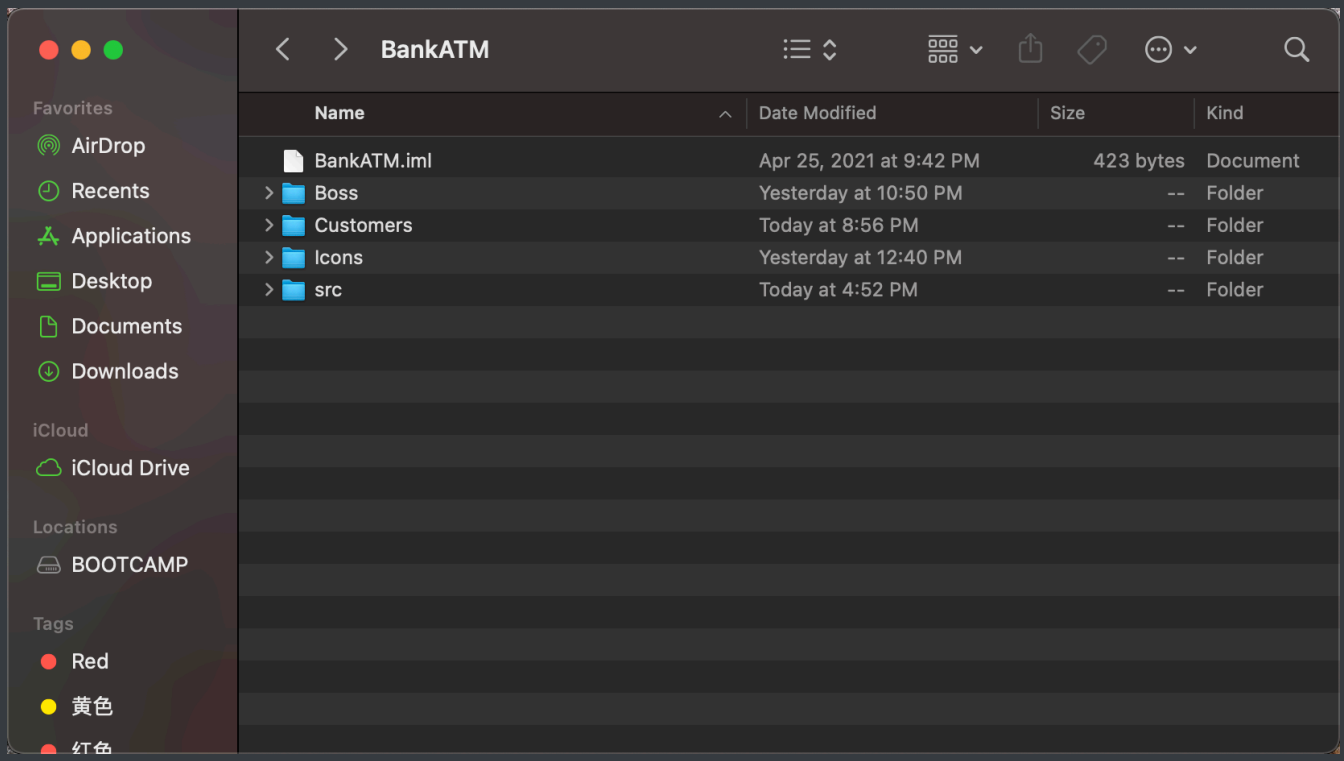
1. **ATTENTION PLEASE!** The way that **IntelliJ** locates file directories is **different** from the way that **Mac Terminal** finds files!
2. The name of our project folder is **BankATM**.
3. If you are going to run the program via Mac Terminal, the first thing to do is to move folders named **Boss, Customers, and Icons** to the **/src** folder! So the final **/src** folder will look like this:



4. Open Terminal and type "cd " (notice that there is a **whitespace** after "cd" !).
5. Then **drag the project folder** to the terminal so that Mac can automatically complete the directory of that folder for you.
6. Press "Enter". Now you are inside the newly created folder.
7. Type "javac BankStarter.java" in the terminal, and press "Enter".
8. Type "java BankStarter" in the terminal, and press "Enter".
9. Now you should be able to run the project through a popped up GUI interface.
10. Please note that you should have a **JDK** installed in you MacBook with version **1.8** (we have tried to run on **JDK 15** but **FAILED!!!**).

Instructions on how to compile and run the program in [IntelliJ IDEA]

1. Use IntelliJ IDEA CE to open the project folder (i.e. **BankATM**).
2. If you are using the version that we uploaded to **GradeScope**, the project hierarchy should have **no problem**. But please make sure to double-check that **Boss, Customers, and Icons** should be **one layer** outside of the **/src** folder while **BankAttributes**, **Data**, **GUI**, and **BankStarter.java** should be inside the **/src** folder! So the final project hiereachy will look like this:



3. Click the "Run" button or press Control+R to run the project.
4. Please note that you should have a **JDK** installed in you MacBook with version **1.8** (we

have tried to run on **JDK 15** but **FAILED!!!**).

---

* Highlights of the Project

1. Colorful and Lively GUI Experience

- We have implemented a lot of colorful icons and graphics on the buttons, backgrounds, and messages of the project.

2. Usage of Design Pattern

- Since We found that the algorithms for exchanging one type of currency (e.g. Chinese Yuan ¥) to others (e.g. US dollar $) vary, We have implemented the **Strategy Pattern** to facilitate the currency exchanging operations for customers.

3. Strong and Robust Object Model

- It is very hard for our program to crash.
- It can self-handle various legal and illegal user operations. When the system is reloaded, all **user** and **account** data will be automatically and seamlessly read from the **database**.

4. Realistic Stock Market Simulation

- In the real world stock market, a customer can view the change rate of the stock market forever without selling her/his stock.
- In our program, it is also the same. The customer can keep updating the stock market and refreshing the up-to-date **change rate** without selling her/his stock.
- For instance, you put $1000 to the stock market, and our program will simulate the real world stock market by letting you keep hitting the "update" button. After 3 times of hitting, your stock balance drops to $500 and you may feel extremely frustrated. However, chances are that the next hit will give you a $1200 stock balance!
- This little "update" button has **magic**. You can feel the **power of luck** if you play around with our stock market feature ^
- Considering our group only has **TWO** members, we think by implementing this

"update"button, we are really realizing a difficult stock feature task by costing the minimum energies.

5. Proper Code Format

- Proper indentations, **"MORE THAN ENOUGH"** comments, etc.

Things Worth Noting

1. About Our Database

- We use **.csv** files to store our data.
- There are **THREE** data files in our project, namely **Accounts.csv**, **Customers.csv**, and **Fees.csv**. The first two exist in the folder called **Customers** and the last one exist in the folder called **Boss**.
- **Accounts.csv** stores all the accounts' information, **Customers.csv** stores all the customers' information, **Fees.csv** stores the history of all account opening fees.

2. Data in Fees.csv

- Please notice that after you download our project from GradeScope, both **Accounts.csv** and **Customers.csv** are empty. This is because we want our players to start playing with a brand new copy.
- However, you will find there is a pre-written record in **Fees.csv**, namely "**5.0,05/02/21/05:09:46**". This is because we have preset the account opening fee as **$5** on **05/02/21/05:09:46**. During project runtime, if the manager wants to adjust this fee, the program will add records after the preset line.
- Therefore, please **don't delete** the first (i.e. preset) record in **Fees.csv** otherwise the program will fail to retrieve the starting account opening fee.

3. Account Operations History Display

- The manager can check the account operations history of **all customers**, or a **single customer**, and **her/his (i.e. the manager's) revenue history**.
- Please keep in mind that we will display the account operations history only if the

account is currently **ACTIVE**, which means if a customer has closed her/his Checking account, we will not display the operations history of this Checking account.

---

* Finale

At the end of this course, we want to say thank you to **our professor**, **Jenny**, and **Adam** for their great work in this semester! We have really learned a lot on Java, OO, design patterns, Swing, and teamwork!

It is truly inspiring to witness our improvements on programming skills and problem solving abilities by finishing all previous assignments and this final project!

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