Project 1: OOP Design and Implementation

Barron Botts, Michael Chapman, Ryuichi Ikegami

Capitol Technology University

Author Note

This lab was prepared for CS 200, taught by Professor Andrew Mehri, and its report was due by 11/16/2019.

Table of Contents

[Abstract 3](#_Toc24831814)

[Introduction 4](#_Toc24831815)

[Procedure 4](#_Toc24831816)

[Figure 1.2 4](#_Toc24831817)

[Figure 1.6 5](#_Toc24831818)

[Figure 1.3 5](#_Toc24831819)

[Figure 1.9 6](#_Toc24831820)

[Figure 1.10 6](#_Toc24831821)

[Analysis 7](#_Toc24831822)

[Figure 1.1 7](#_Toc24831823)

[Figure 1.4 7](#_Toc24831824)

[Figure 1.5 8](#_Toc24831825)

[Figure 1.7 8](#_Toc24831826)

[Conclusion 9](#_Toc24831827)

[Figure 1.8 9](#_Toc24831828)

[References 10](#_Toc24831829)

# Abstract

This group project allowed us to create a deep understanding of the benefits of the Agile methodology of project designing and implementation. UML diagrams helped to share a single visualization of the project at hand. Inheritance was used tastefully to achieve our client’s design requirements. Polymorphism was misused due to poor planning, yet encapsulation was used as it should. Use of recursion was an added cherry on top of properly using Scrum and UML organization methods.

# Introduction

This lab asks students to work as part of groups to create an application that would be useful for at least three unique actors. The three actors groups are charged to develop for are customers, bank tellers, and system administrators. While designing the project for those actors, groups must make use of encapsulation, inheritance, and polymorphism when coding. Before groups begin their coding phase, they are to design their project using UML diagrams and any tools related to Agile development.

# Procedure

Before we could begin coding, we absolutely had to organize our thoughts because we were more confused than not when we started. The first diagram we drew out was out class diagram (Figure 1.9), which shows the relationships we wanted. To clear up any further confusion we drew up a use-case diagram (1.10). Our Agile method was decidedly Scrum. Botts acted as our scrum master. Our sprints ended on Tuesdays, Thursdays, and Saturdays which led to four total sprints (Figure 1.8).

After organizing, we tackled our class diagram, which managed to be nearly half of our user story points. User holds the security flag, which defines each user’s use of the ATM and Audit classes, rather than having their ID numbers define such. Special cases in which Customers are employees would require new login credentials anyway. After that we only needed a steady workflow to adhere to our much different schedules. When we created those classes we thought we knew for sure how and where to use polymorphism, but only long since we had already finished the audit class did we realize that it would have been better to use polymorphism in that class (Figure 1.7). Encryption took surprisingly long to complete re ran into some unpredictable errors, the chosen encryption key manages to not save the letters ‘f’ or ‘v’ correctly. We had to use subpar naming conventions to fix this quickly (cplusplus.com, 2019).

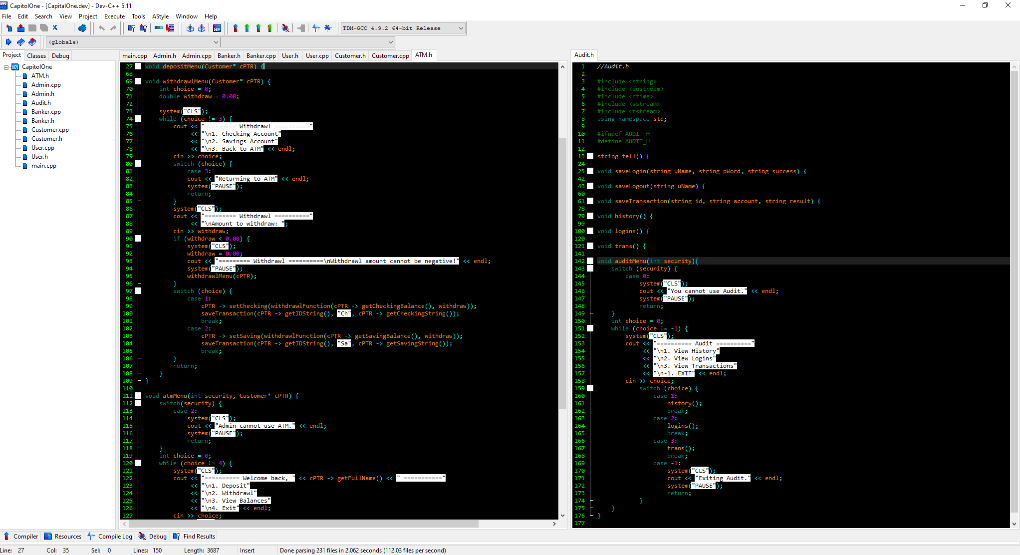


Figure 1.2 – Source code depicting some of the menu-driven interfaces.

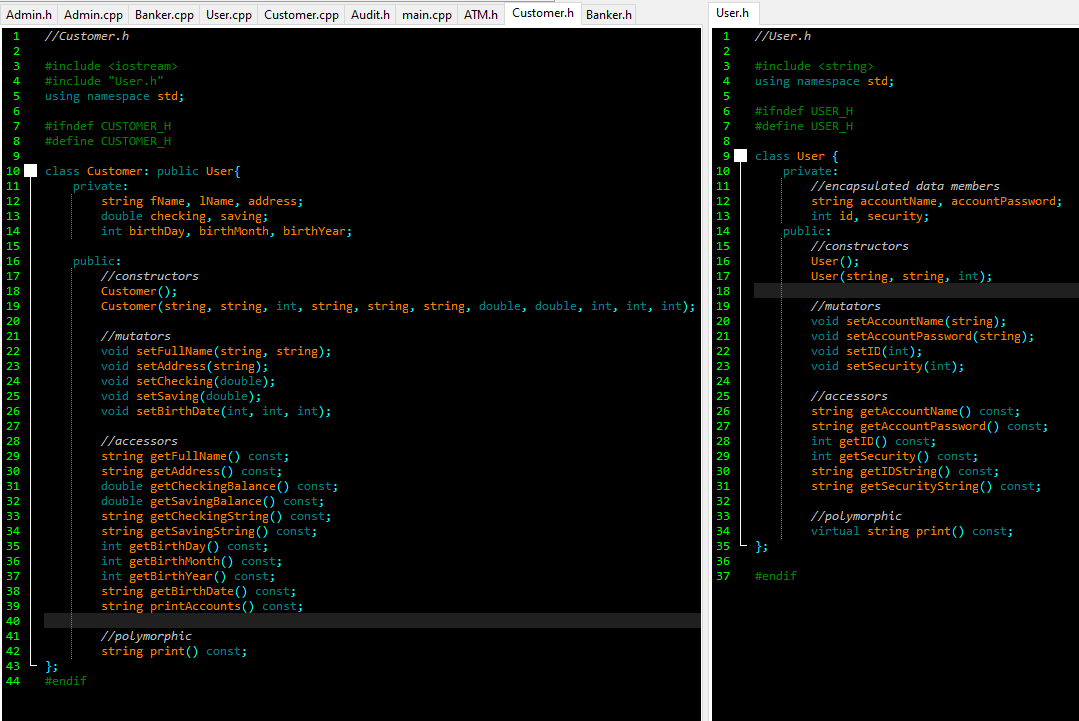


Figure 1.6 – Inheritance from User to Customer. Banker and Administrator also inherit User.

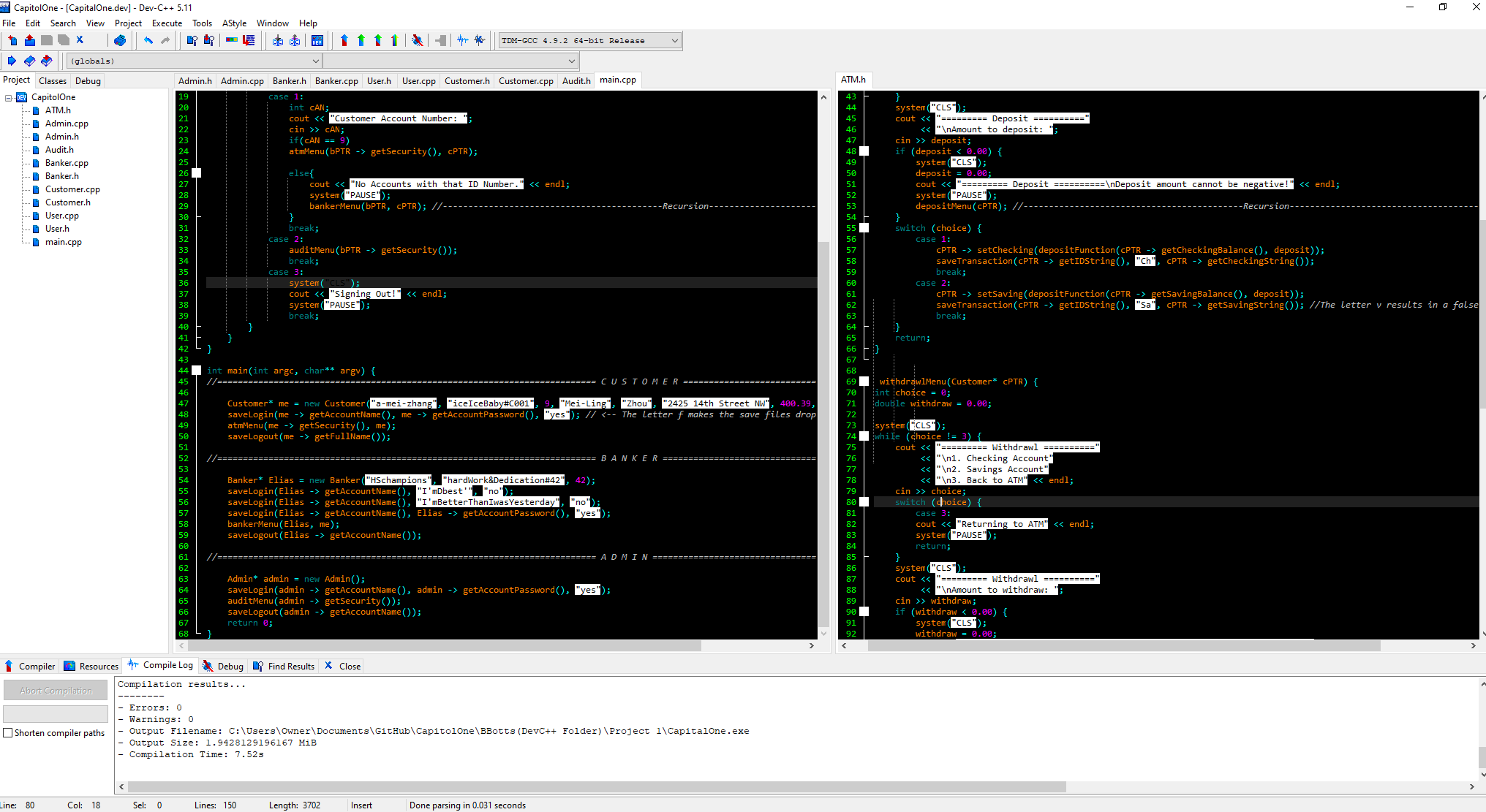


Figure 1.3 – Source code of where recursion is used. Recursion allows the customer using the atm to try to overdraft either of their accounts yet puts them in the menu with the option to affect their account again. Recursion also allows the user to attempt to enter a negative number for deposits or withdrawals – which would give a result opposite of their intended functions - but sends them back to the menu to attempt to enter a valid transaction. The final use of recursion in this program allows the banker to enter a false account number without forcing them to sign out.

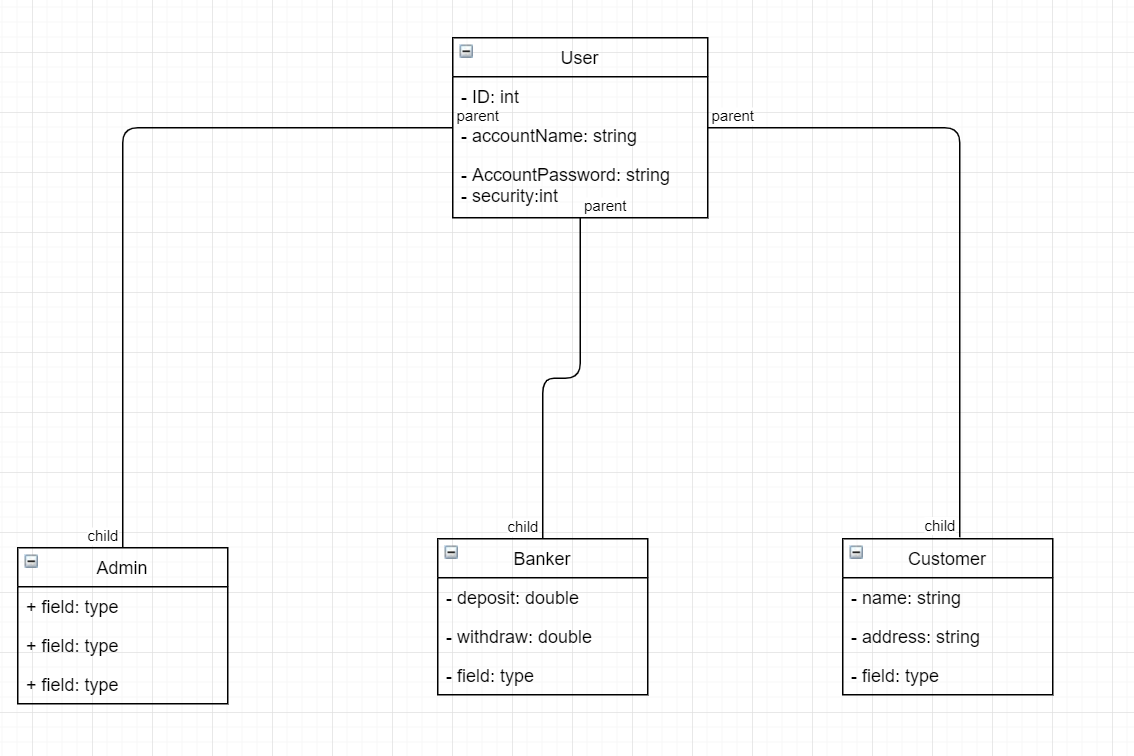


Figure 1.9 – UML Class Diagram. User is the parent of Customer, Banker, and Administrator.

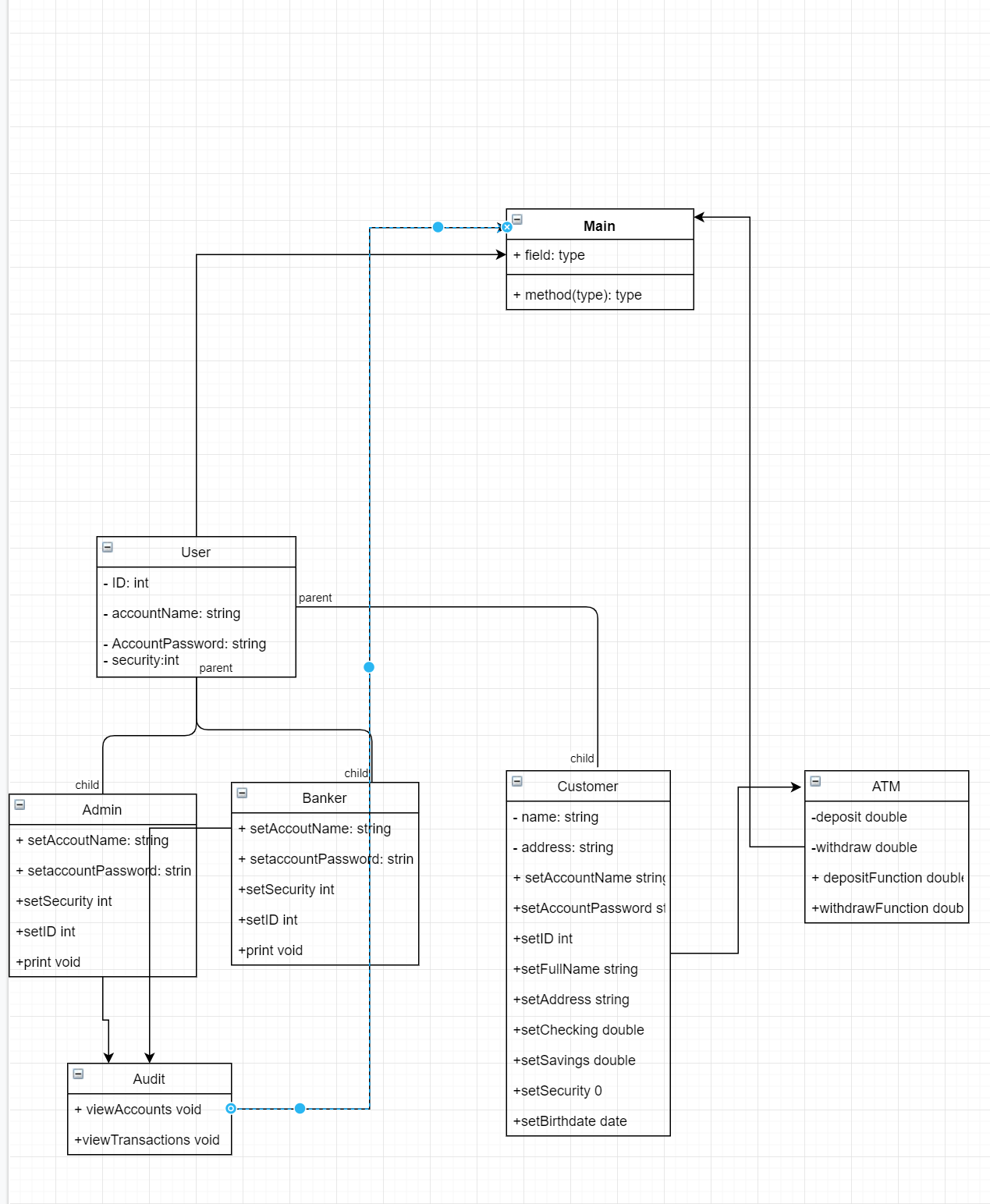


Figure 1.10 – Use Case. Displays the interaction between classes.

# Analysis

Recursion (Figure 1.3) turned out to clear one small point of contention I had been holding onto all semester. When I got around to exception handling, I disliked breaking the user’s sense of flow by kinking them down to an earlier menu. When I saw the opportunities to include them, I did not hesitate. Creating the Audit class proved to be difficult since we decided early on to use an unfamiliar format of saving information. Not only was it using encryption and decryption (cpluscplus.com, 2019), but we also had to use the timestamps to keep accurate logs of what happened while the program was in use (Parewa Labs Pvt. Ltd., 2019).

Encapsulation feels like a non-issue after using it so much ever since we learned it. User and Customer were the only courses to include it, because they were the only classes that really needed to use encapsulation. We were very much on track to misuse inheritance during our earliest design phase, thankfully our client set us straight before we derailed ourselves. Our earliest Class diagram had each user inherit in order of permissions level, which is how we created the encapsulated integer ‘security’ as a part of each user. Mentioned earlier about figure 1.7, our use of polymorphism is not used, therefore nonexistent in this program. I believe this is a good learning experience and we will be able to better identify more efficient uses of polymorphism in the future.

We thankfully used UML diagrams way before we even thought about coding to organize ourselves when we had shaky footing on the concepts that we were to utilize (Figures 1.10 and 1.11). There was a period between the class and use-case diagrams where we sat down to code, and thankfully our code fit perfectly into the latter diagram, but that was not worth the risk, we should have planned better. Agile was useful in keeping track of what we had yet to complete for the program. It was nice to see what the most important and quickest tasks should have been so we could try to organize based on complexity of our personal schedules.

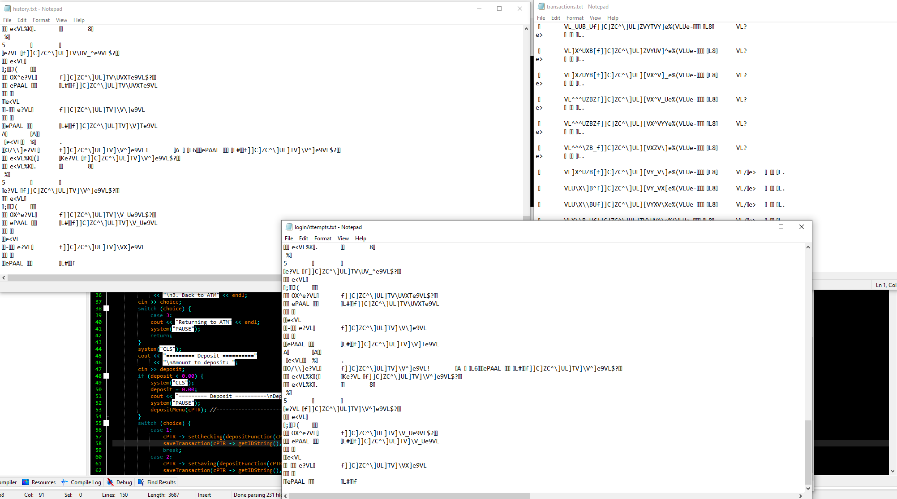


Figure 1.1 – Encrypted persistent storage of transactions, login attempts, and complete history.

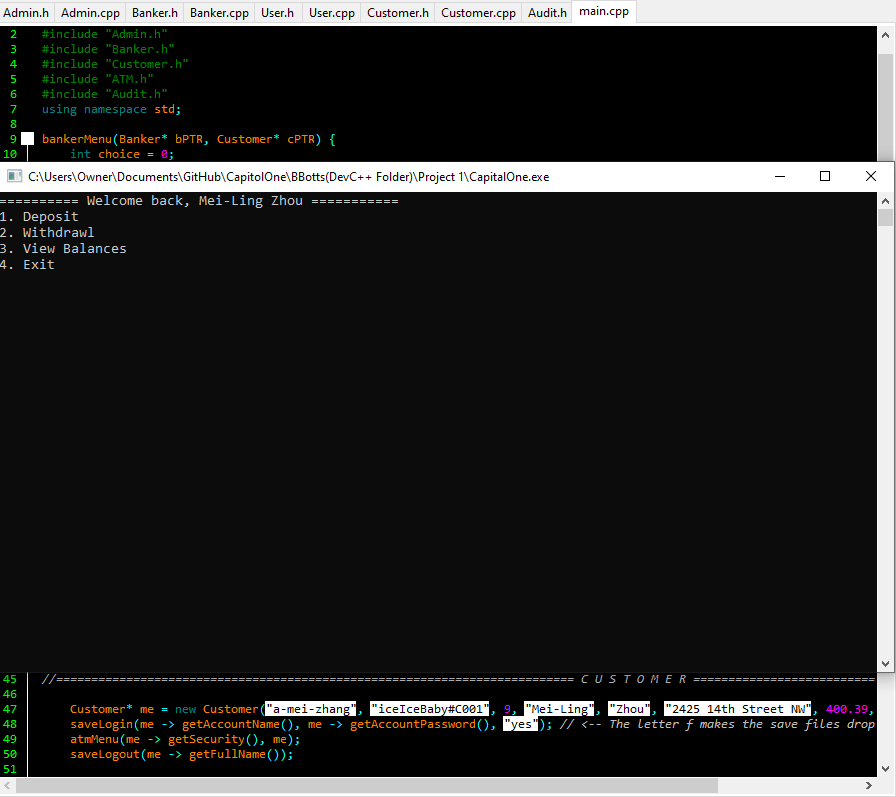


Figure 1.4 – Customer Interface, when using the ATM. The banker uses the same interface.

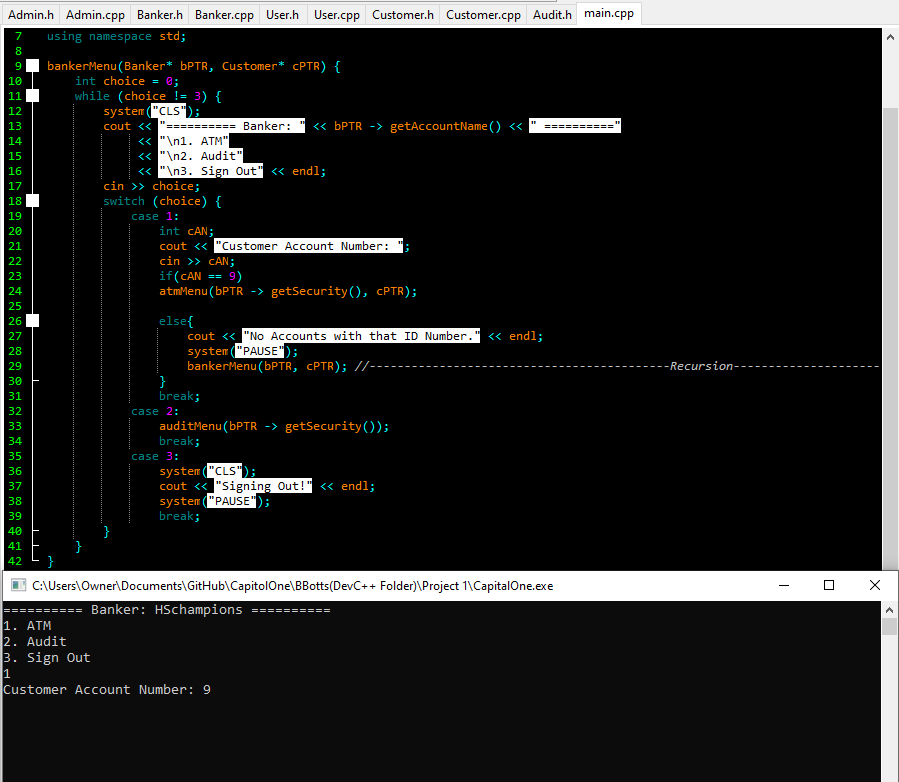


Figure 1.5 – Banker exclusive menu. The banker must use a customer’s ID number.

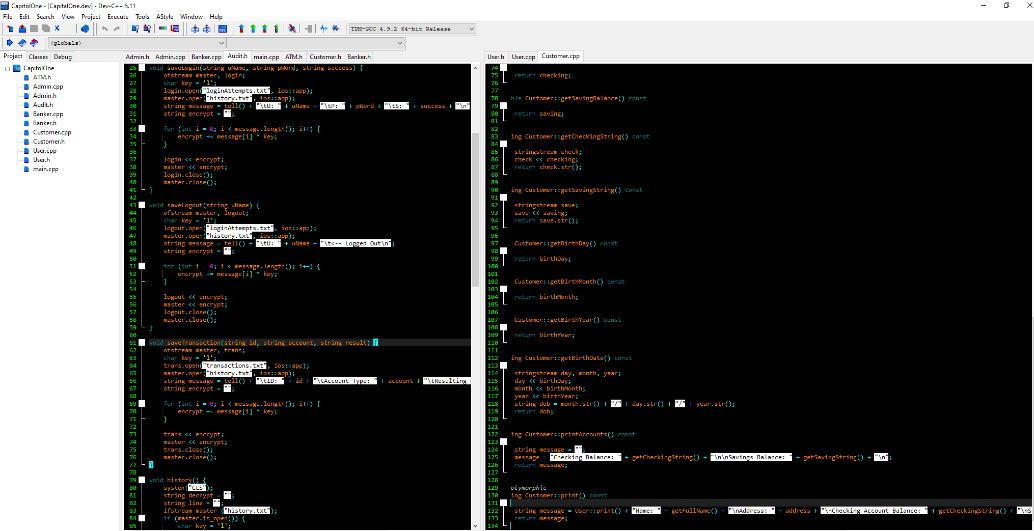


Figure 1.7 – Polymorphism was misused in the project. It should have been fully utilized on the left, where we must print information about each user that logs in. We almost use it in our logout process, where the loginAttempts.txt file saves the Customer’s name, but only saves the username of Banker and Administrator. We planned to use polymorphism on the right of this figure, where Customer’s print function goes as far as to use User’s print function to redefine what print means for Customer objects.

# Conclusion

This program is a strong introduction to working in a group, the task was bigger than what I could have handled on my own. We used the tools of Agile and Scrum introduced in the context of working in an office, to make largely disjointed efforts fit together in a necessary way. We almost managed to pull together a proper utilization to polymorphism. We ended up with a reserved use of inheritance and a perfect use of encapsulation.

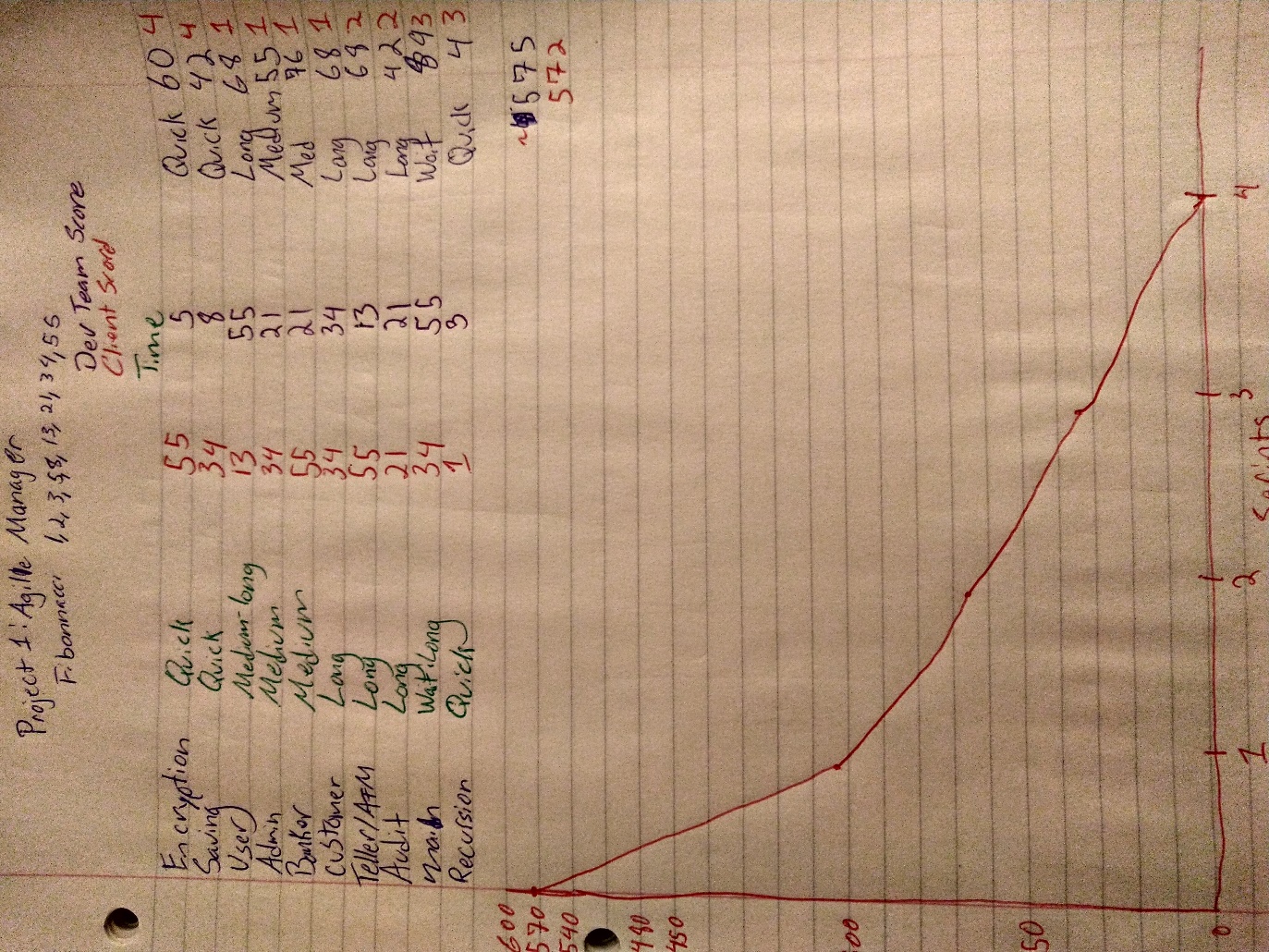


Figure 1.8 – Agile userStory and burndown charts.

# References

cpluscplus.com. (2019, November 16). *Input/output whith files*. Retrieved from C++ Tutorials: http://www.cplusplus.com/doc/tutorial/files/

cplusplus.com. (2019, November 16). *getline (string)*. Retrieved from C++ Reference: http://www.cplusplus.com/reference/string/string/getline/

Parewa Labs Pvt. Ltd. (2019, November 16). *C++ strftime()*. Retrieved from C++ Standard Library: https://www.programiz.com/cpp-programming/library-function/ctime/strftime