CPSC501 Assignment1 Report

In the original code, I identified several issues and poor coding practices:

1. Duplicated Code: There's an excessive amount of repeated code in the program, which hasn't been adequately encapsulated or abstracted, making maintenance more challenging.
2. Long Function: Some functions exceed 40 lines, which can make comprehension and debugging more difficult.
3. Global Data: Numerous public classes are used to transfer user data, posing a potential security risk.
4. Shotgun Surgery: There are many scattered functionalities that need to be consolidated to enhance coherence and maintainability.
5. Bad Loops: Some loops in the code are complex and hard to understand, which can lead to errors and hinder readability.
6. Large Class: There's a class with up to 350 lines of code, indicating that it might be trying to do too much and could benefit from refactoring.
7. No Comments: The lack of comments makes it challenging to recall the initial thought process and logic, which can be problematic for future modifications or collaborations.

commit 8cfc482a590b1df8b8fecc9dc314f2ac3b587b39

In the first revised version of the code, I made the following improvements:

Adding Comments: I spent two days delving into the code and added comprehensive comments throughout. These comments not only help me understand the code better but will also save a significant amount of time when revisiting it in the future.

Writing JUnit Tests: I wrote tests for almost all the crucial methods and ensured that they all passed. For instance, when testing the "register" function, I first input my name, password, and daily withdrawal limit. For example: String input = "王哲正\n123\n123\n100.0\n";

Then, I used assertEquals to validate the correctness of the results.

```
assertEquals(1, accounts.size());
Account account = accounts.get(0);
assertEquals("王哲正", account.getUserName());
assertEquals("123", account.getPassword());
assertEquals(100.0, account.getQuotaMoney(), 0.001);
assertNotNull(account.getCardID());
```

commit bf775af11ea9c307a50a768b46e8fd3fb73891c3

This is the first major refactoring version I've completed. The primary changes include: addressing the issue of long functions, consolidating scattered functionalities, and modifying large classes and duplicated code.

To enhance code readability and maintainability, I decomposed the original monolithic class based on distinct functionalities, resulting in nine more refined classes. I also made sure that similar functionalities and data were grouped together. Despite such restructuring, the test code still runs as expected. This refactoring has streamlined the entire ATM system. The modularization of each functionality not only improves code readability but also makes it easier to maintain.

commit 6bbc4d7ee5c3f6b5c546d27d79549caa82a1d61a

Second Refactoring:

The main changes this time around are: fixing the issue with global data and adding a JUnit test for the login class.

To enhance data protection, I changed the access level of all functionalities to protected. In addition, I introduced a test specifically for the login class. In the test, I started by entering my username, password, and daily withdrawal limit: String input = "王哲正\n123\n123\n100.0\n";

```
assertEquals(王哲正cardID, account0.getCardID());
assertEquals("王哲正", account0.getUserName());
assertEquals("123", account0.getPassword());
assertEquals(100.0, account0.getQuotaMoney(), 0.001);
assertNotNull(account0.getCardID());
```

Next, I retrieved a randomly generated eight-digit number using the cardID associated with 王哲正. Finally, I used assertEquals to validate the result's accuracy.

commit 0a1ae30f7a8d2a94575669cfccb24c092488f5e8

Second major refactoring version

Register class refactoring：

Write a private method createAccount to generate the account

Write a private method getPassword to generate a password

Put the successful registration message in printRegistrationSuccess to reduce code duplication.

Login class refactoring：

Put the verification password in the private method validatePassword

Put successful login into the private method successLogin

Simplified the structure of the conditional statement to make the code more readable.

Draw class refactoring：

Reorganize blocks of code into a more readable structure and use more descriptive variable names.

Encapsulate the logic of executing withdrawals by extracting the performWithdrawal method to reduce code duplication.

Combine else and if

Use more descriptive prompt messages to improve user experience.

DeleteAccount class refactoring：

Decompose the logic of deleteAccount into 6 private methods confirmAccountDeletion,

hasBalance, cannotDelete, removeAccount, printAccountDeleted, printAccountReserved

Improve code readability.

Use more descriptive method and variable names to make your code easier to understand.

Reduced nested conditional statements to make code clearer.

ATMSystem class refactoring：

Change main to public.

Encapsulate invalidCommandHandler and mainMenu,

Junit tests all passed

commit 68e359f79120b63e107f966a928e0fa3fa0bcbca

Forth commit changed Readme file:

The main functionalities within the ATMSystem class have been encapsulated into 8 separate classes, which are: Register, Login, Draw, Deposit, DeleteAccount, ChangePass, ATMOperationHandler, and TransferMoney.

This enhances the code's readability.

Nine Junit Tests have been added, which include: testRegister, testGetRandomCardId, testGetAccountByCardId,

testDeleteAccount, testUpdatePassword, testTransferMoney, testDrawMoney, testShowAccount, and testLogin.

The classes have been made protected, increasing the security of the code.

commit f39f344df4e586fd5fc605cbd491bd92a5b80e8c

Fifth commit fixed forth commit‘s problem，Fixed showUserCommand loop problem

In summary, refactoring can improve code readability, maintainability, and scalability.
Structural optimization: After the code has been refactored, it is no longer one lengthy class or multiple lengthy functions, and the structure is clearer and more orderly.
Utilize Java encapsulation: Through encapsulation, I can hide the state of an object and restrict access to that state, thus ensuring that the object is not accidentally modified externally. At the same time, Java's encapsulation can make the code more concise and easier to read.
Improve the maintainability of code: When the code structure is clear and the functions are modularized, it will be much easier to maintain. When I encounter a problem, I can quickly locate the module where the problem lies instead of searching in a mess of code.
Lay the foundation for further refactoring: A good code structure can make future refactoring work smoother and make it easier to adjust and optimize each class in accordance with coding standards.
Supports increased testing: The refactored code makes it easier to write test cases, especially using testing tools such as JUnit. We can add some edge tests of the code to ensure the security of the system (because this is a banking system after all) and reduce potential loopholes and errors.
In general, code refactoring is a continuous process, its purpose is to make the code clearer and maintainable, and also improve the efficiency of testing.