# Case Stydt

# Group names Name:

**Shadarien Williams**

**Computer Programming Capstone**

**24/Fall -**

# Instructor: Dr. Mubarak Banisakher

# November 5, 2024

Table of Contents

[Design: 2](#_Toc181683016)

[Comments and Documentation: 3](#_Toc181683017)

[Code Readability: 3](#_Toc181683018)

[Safety: 3](#_Toc181683019)

[Security: 4](#_Toc181683020)

[Performance: 4](#_Toc181683021)

[Test Code: 4](#_Toc181683022)

[Logging: 4](#_Toc181683023)

[Metric Generation: 4](#_Toc181683024)

## Design:

The script organizes the banking system through an Account class and related methods. However, the design could benefit from encapsulating file operations within methods, like loadAccounts and saveAccounts, rather than handling them throughout the code. This approach improves reusability and readability.

Adding error handling within functions, such as when loading or modifying files, would make the design more robust.

## Comments and Documentation:

While functional, the script lacks docstrings for classes and methods. Adding docstrings would clarify the purpose and parameters for each function, improving maintainability. Inline comments could enhance understanding, especially in areas that manipulate account data within files.

## Code Readability:

Variable names are generally meaningful, such as depositAmount and withdrawAmount, but readability would improve with clearer spacing, consistent naming, and avoiding single-letter variables like ch. Repeated code blocks, especially around file handling (e.g., opening, loading, and saving data), could be modularized into helper functions to make the code more concise and less error-prone. Magic strings, like "accounts.data" and "newaccounts.data", could be defined as constants at the top of the file.

## Safety:

The script assumes valid user input, so adding validation (e.g., verifying account types, checking integer inputs) would make it safer. The depositAmount and withdrawAmount methods should check for negative values to prevent incorrect deposits or withdrawals.

## Security:

Input validation for account numbers and deposit amounts is minimal. For improved security, validate inputs to prevent unexpected behavior or errors. For a real-world application, it’s best to avoid storing sensitive data like account numbers in plain text. An encryption method or secured database would be safer.

## Performance:

The script reloads all account data from the file each time an operation is performed. For larger datasets, consider an alternative, like a database system or more efficient file-handling practices. Methods such as depositAndWithdraw and modifyAccount open and close files each time they’re called, which could be optimized by keeping files open for longer when practical.

## Test Code:

The script lacks test code. Unit tests could validate functions like createAccount, depositAmount, and withdrawAmount to ensure they handle various edge cases correctly. Mock tests for file operations would help simulate real-world usage without needing actual files.

## Logging:

Logging is absent, which limits debugging and audit trails. Adding logging statements to key actions (like creating, modifying, and deleting accounts) would improve traceability. A logging framework (like Python’s built-in logging module) would offer better control over log levels and outputs compared to print statements.

## Metric Generation:

The script could benefit from simple metrics, such as the number of accounts created, total deposit amounts, or transaction counts. Such metrics provide insight into system usage. Implementing a basic metrics collector within the Account class would add value for tracking these details.

## Metric Generation:

The script could benefit from simple metrics, such as the number of accounts created, total deposit amounts, or transaction counts. Such metrics provide insight into system usage. Implementing a basic metrics collector within the Account class would add value for tracking these details.