## Cairo University Faculty of Computers and Artificial Intelligence



Software design specification document

**2023**

**Project Team**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email** |
| 20206054 | Farah Mohamed | farahmo166@gmail.com |
| 20206026 | Rahma Ayman | rahmaaymn000@gmail.com |
| 20216028 | Bassant Ali | Bassant.ali23@icloud.com |
| 20216107 | Nada Ahmed | Nadatawfik2003@gmail.com |

2

[Class diagram design 2](#_TOC_250003)

[Class diagram Explanation 3](#_TOC_250002)

[Sequence diagram design 3](#_TOC_250001)

[Github repository link 4](#_TOC_250000)

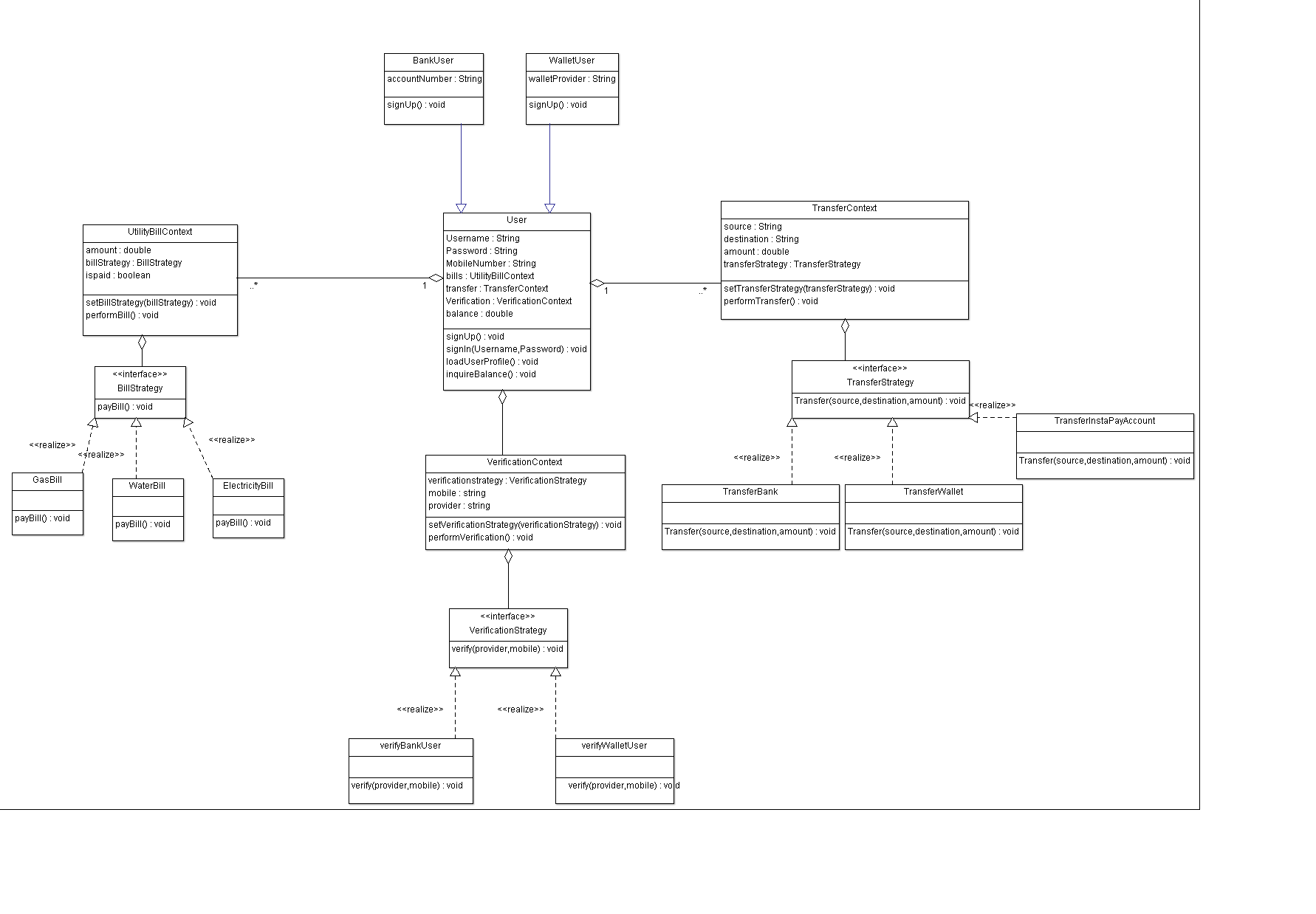
# Instructions[To be removed]

* **IMPORTANT. Rename this document to**

### Phase1\_StudentID1\_StudentID2\_StudentID3\_StudentID4\_SDS Document.docx

* **Remove the following notes and any red notes**

# Class diagram design



# Class diagram Explanation

# Class diagram Explaination

# Abstract pattern used in :

# 1-"User " class <interface> : It represents a general user of the Instapay system. This class can have different types of users, such as bank users and wallet users. By using an abstract pattern, the User class provides a common interface for all types of users, allowing for code reusability and simplifying the implementation of features that are common to all user types, such as transferring funds and inquiring about balance.

# 2-Bank User and Wallet User:

# They represent users who have registered using their bank account or with a wallet provider, These classes inherit from the User class, which allows them to inherit common functionality while also providing the flexibility to add specific behaviors or properties unique to each type of user. For example, the Bank User class can implement the functionality for transferring funds to a bank account, which is not used to Wallet User.

# 3-Utility Bill Context: The Utility Bill Context class represents the context or environment in which utility bills are managed and processed within the Instapay system. Using an abstract pattern allows for encapsulating the logic related to utility bills, such as creating bills, deducting amounts, and managing bill types (Gas, Electricity, Water). It provides a standardized interface for interacting with utility bills, regardless of their specific contents, making the code more modular and easier to maintain.

# 4-Transfer Control Context: The Transfer Control Context class represents the context or environment in which fund transfers are controlled and managed within the Instapay system. This class handles the transfer of funds between different accounts, such as wallet-to-wallet transfers, bank-to-wallet transfers, and bank-to-bank transfers. By using an abstract pattern, the Transfer Control Context class provides a unified interface for managing transfers, regardless of the specific account types involved. It simplifies the implementation of transfer functionality and allows for easy extensibility in the future if new transfer scenarios need to be supported.

5-Verification Context:  
The Verification Context class represents the context or environment in which user verifications are handled within the Instapay system. This class is responsible for managing the verification process during user registration, specifically for verifying mobile numbers and bank/wallet accounts. By using an abstract pattern, the Verification Context class provides a unified interface for performing verifications, regardless of the specific verification methods involved. This approach allows for flexibility in adding or modifying verification strategies in the future without impacting the overall system design.

The Verification Context class encapsulates the logic for verifying user information, such as mobile numbers and bank/wallet accounts, ensuring their validity and accuracy.

It interacts with external APIs or services, such as the bank's API or the wallet provider's API, to perform the necessary verification steps.

By using the Verification Context class, the system abstracts away the implementation details of the verification process and provides a standardized approach to validating user information.

The class follows an abstract pattern, allowing for different verification strategies to be implemented interchangeably.

It defines a common interface, such as VerificationStrategy, outlining the methods required for verifying different account types.

Each verification strategy, such as VerifyBankUserStrategy, VerifyOTPStrategy, and VerifyWalletNumberStrategy, implements the VerificationStrategy interface and provides its own logic for verifying the account type.

The Verification Context class enables easy switching between different verification strategies based on the user's chosen registration method (e.g., bank account or wallet).

# Strategy pattern used in :

1-Verification Strategy :  
This class is implemented using the Strategy pattern to manage different verification methods for bank and wallet accounts. It allows for flexible addition or modification of verification strategies in the future without impacting the overall system design. This approach simplifies the codebase and avoids conditional statements.

The Verification Strategy class defines a common interface, such as VerificationStrategy, which outlines the methods required for verifying bank and wallet accounts.

Each verification strategy, such as VerifyBankUserStrategy, VerifyOTPStrategy, and VerifyWalletNumberStrategy, implements the VerificationStrategy interface and provides its own logic for verifying the account type.

By using the Verification Strategy pattern, the system can easily switch between different verification strategies based on the user's chosen registration method (bank account or wallet). It provides a modular and maintainable approach to handle account verification.

2-Transfer Strategy :  
This class utilizes the Strategy pattern to manage different transfer methods, such as transferring to a wallet, another Instapay account, or a bank account. This pattern allows for flexibility in adding or modifying transfer strategies in the future without impacting the overall system design.

The Transfer Strategy class defines a common interface, such as TransferStrategy, which outlines the methods required for executing transfer operations.

Each transfer strategy, such as TransferToWalletStrategy, TransferToInstapayAccountStrategy, and TransferToBankAccountStrategy, implements the TransferStrategy interface and encapsulates the logic for its specific transfer method.

By using the Transfer Strategy pattern, the system can select the appropriate strategy based on the user's chosen transfer method, providing a modular and extensible approach to handle different transfer scenarios.

3-Bill Strategy :  
The Bill Strategy class employs the Strategy pattern to support different bill payment options, including Gas, Electricity, and Water bills. This pattern allows for flexibility in adding new bill types and their specific payment processes in the future.

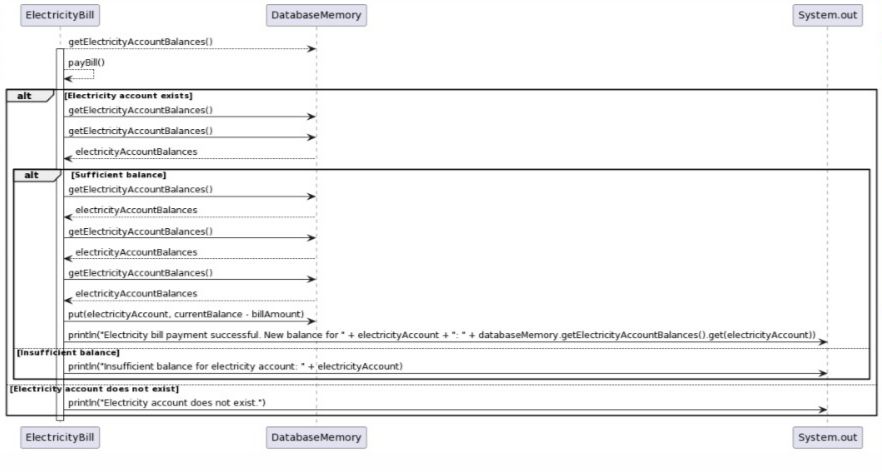
The Bill Strategy class defines a common interface, such as BillStrategy, which outlines the methods required for creating and deducting bills.

Each bill type, such as GasBillStrategy, ElectricityBillStrategy, and WaterBillStrategy, implements the BillStrategy interface and provides its own logic for creating and deducting bills based on the specific requirements of the bill type.

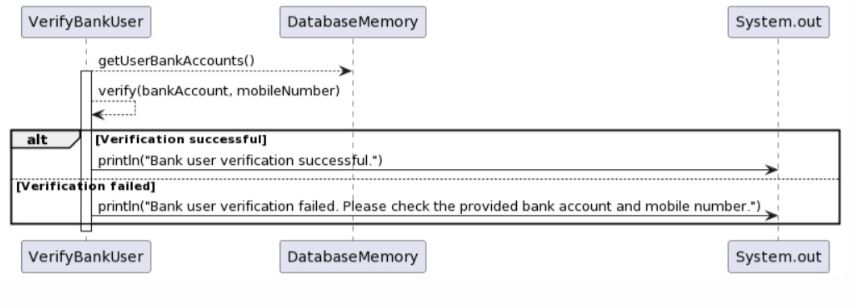
By using the Bill Strategy pattern, the system can dynamically select the appropriate strategy based on the user's selected bill type. This approach promotes code reuse, reduces duplication, and improves maintainability by encapsulating the specific bill payment processes within separate strategy classes.

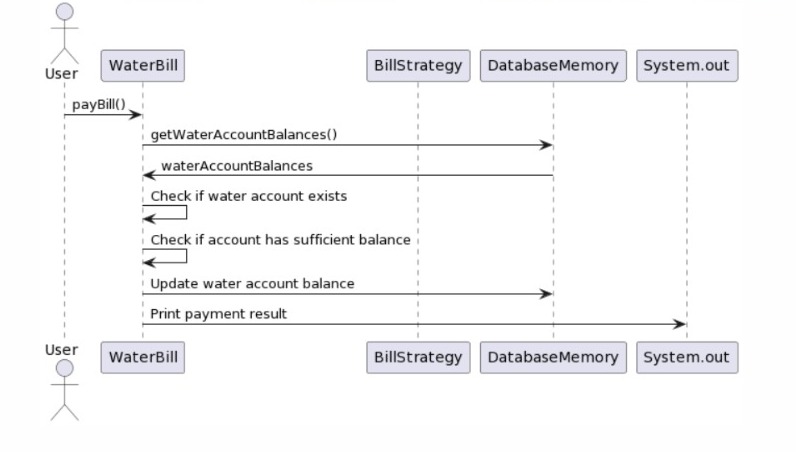
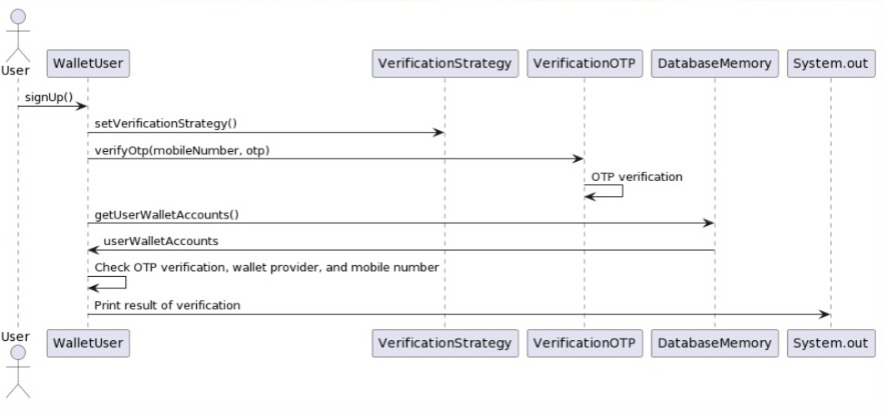
# Sequence diagram design

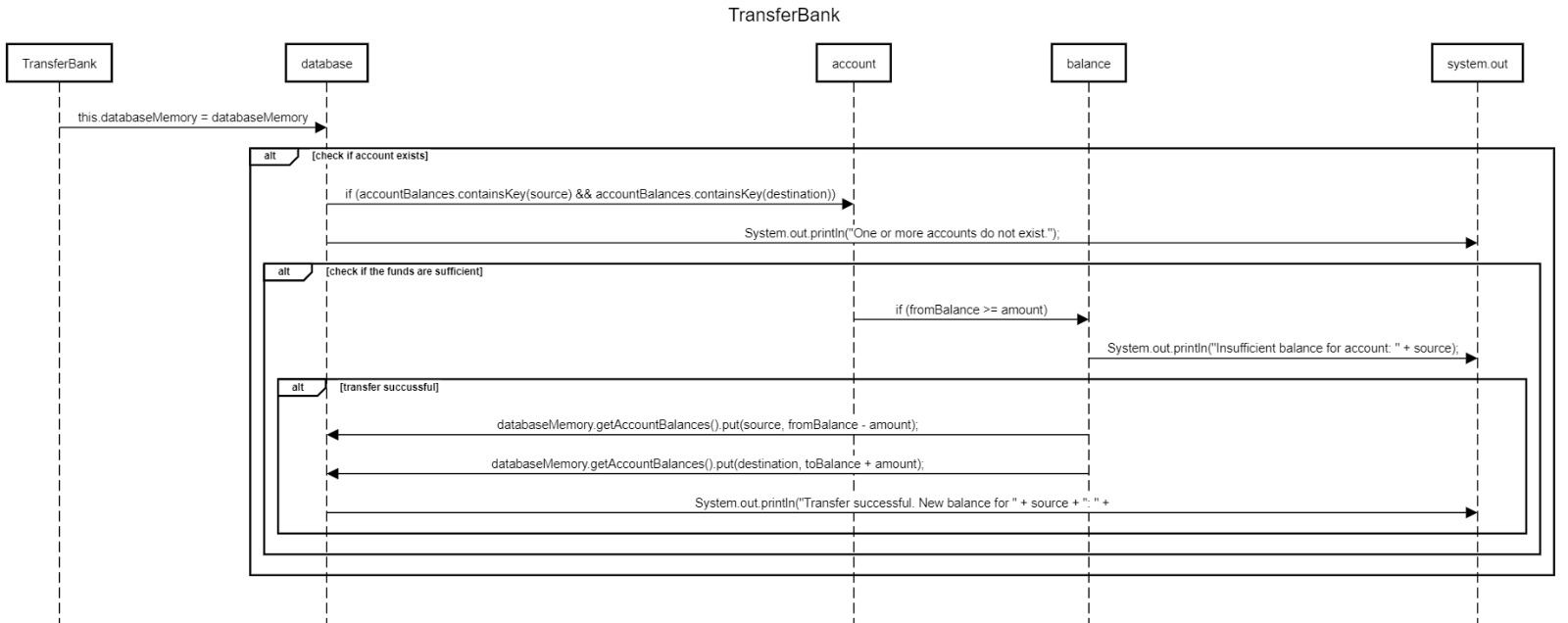
ElectricityBill



BankVerify



waterbillWallet User

TransferBank

# Github repository link

* **URL for your repository, should contains implemented**

**https://github.com/FarahMo6/InstaPaySystem.git**