# CS3500 Object-Orientation Software Engineering Team Project: Automated Teller Machine (ATM)

(Team 11):

Divine Dickson-Uwakwe

Seven Gunthrope

John Sam-Bruce

Andrew Vervaet

# Table of Contents:

## Table of Contents (Cont.):

# Introduction

The goal of this project is to develop an automated teller machine (ATM) following the waterfall model and applying software engineering principles. The project had our team follow four sections of the assignment: Use Case Modeling, Class Diagram development, Communication Diagrams, and Implementation.

In section one of the project, our team was provided with a partial set of requirements for the ATM system, so our initial task was to analyze, interpret, and expand these requirements into a complete and reliable specification. Treating the assignment as a real-world software development scenario, we anticipated that additional needs or refinements might emerge, and we designed the system with adaptability and maintainability in mind to handle these changes effectively. Once the requirements were fully established, we developed a comprehensive Use Case Model to define system behavior. This included creating a Use Case Diagram to illustrate user interactions at a high level and preparing detailed Use Case Descriptions to outline each scenario's flow, goals, and conditions.

For the second part of the project, we identified key operations and supporting attributes for each Use Case, highlighting them in red for clarity, and organized them into classes to reflect the system's components. Each class was given a name, description, relevant attributes, and the core operations that define its behavior. We also modeled any static relationships between classes, specifying relationship names and multiplicities, and used separate diagrams for the complex connections when needed.

For the third section, we created Communication Diagrams for each Use Case, illustrating the dynamic relationships and message flows between objects. Each diagram is based on the Class Diagrams developed in the previous assignment, with adjustments made where necessary to add meaningful operations or refine the class structure. The diagrams show the flow of operations with sequence numbers, include the operation sequences, and represent the scenario sequences to trace each Use Case clearly. These diagrams help visualize how objects interact during system execution and ensure that the dynamic behavior of the ATM system is accurately captured and easy to follow.

The final phase was the implementation phase, where we transformed all classes from our ATM system design into C++ code, following the one-to-one mapping between UML classes and programming language class definitions. Each class from our design was implemented, including its attributes and operations. With this, we also outlined inheritance relationships where applicable, showing what classes inherit what. This approach ensures that the system design is directly translated into a functional, object-oriented codebase.

This project provided a comprehensive experience in applying software engineering principles to develop an ATM system following the waterfall model. By progressing systematically through the four phases—Use Case Modeling, Class Diagram development, Communication Diagrams, and Implementation—we were able to transform an incomplete set of requirements into a fully specified, maintainable, and functional system. Each phase built upon the previous one, ensuring that system behavior, object relationships, and dynamic interactions were clearly defined before moving to implementation. The project reinforced the importance of careful analysis, clear documentation, and structured design in producing reliable software. Ultimately, the exercise demonstrated how UML modeling, combined with disciplined object-oriented implementation, can produce a coherent and operational system while accommodating future refinements and maintaining clarity for both developers and stakeholders.

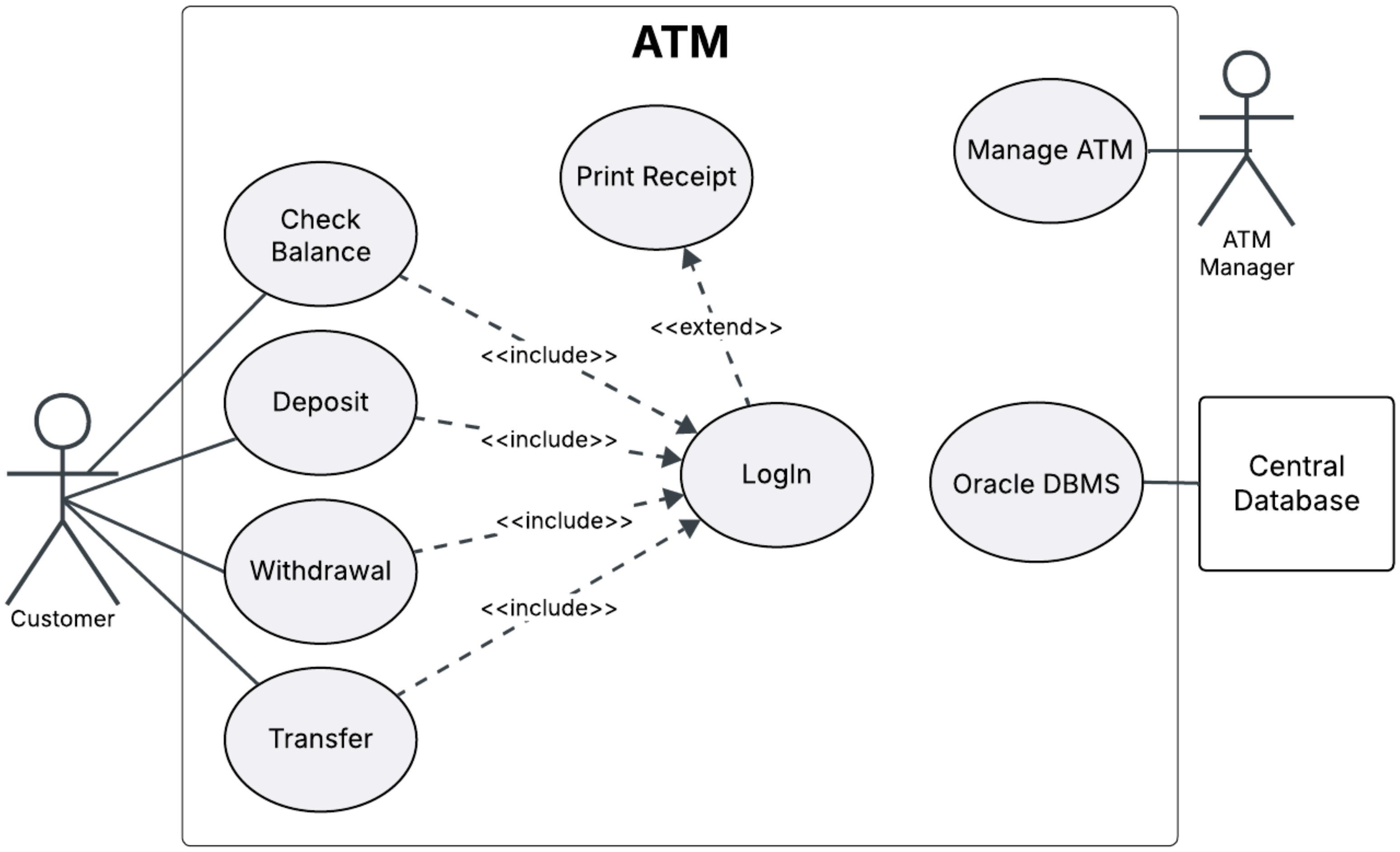
## Team Information

Our team worked collaboratively throughout the ATM project, combining our individual strengths in analysis, design, and programming to achieve the project goals. Each member contributed to different phases of the project, including developing Use Case Models, Class Diagrams, Communication Diagrams, and implementing the system in C++. We coordinated tasks through regular meetings and discussions, both in person and virtually, which helped ensure everyone stayed on the same page and deadlines were met.

Like any team project, we faced a few challenges along the way. At times, differences in working styles and communication preferences caused minor delays or misunderstandings. However, these challenges also gave us opportunities to improve our collaboration, clarify expectations, and build a stronger team dynamic. By sharing feedback openly and supporting one another, we were able to overcome obstacles and maintain a positive working environment.

Throughout the project, we learned several key lessons. Effective communication, early planning, and clearly defined roles were critical for staying organized and productive. We also recognized the importance of flexibility and adaptability, as unexpected changes in requirements required us to adjust our designs and code efficiently. Overall, the experience strengthened our teamwork skills, improved our ability to tackle complex software engineering problems, and highlighted the value of combining diverse skills and perspectives to achieve a common goal.

<u>Use Case Diagram</u>



# Use Case Description (LogIn)

Name: LogIn

Author: Andrew Vervaet

Last Update: 12/7/25

Pre-Conditions:

- Customer has an active account made with a bank
- Customers access the ATM with an ATM card issued by a bank.
- The customer has an active working card and a pin number
- The customer is actively wanting to make a transaction on their account.
- The ATM can access other ATMs from other branches and can access other banks for information. – [ManageATM]:connect_to_ATMs()

Dialog:

- The customer inserts their bank card – [LogIn]:insert_card()
- Card is verified by ATM – [LogIn]:card_verification()
    - If the card is invalid, an error is presented to the customer and asked to try again .(If [LogIn]:card_verification() == “Invalid”, [LogIn]:log_in_retry() )
- They input the 4-digit pin number – [LogIn]:input_pin()
- The OracleDBMS logs the login attempt, then verifies if the pin is registered to that card. – [LogIn]:log_attempts(), [LogIn]:verify_pin()
    - If the pin is wrong, they are asked to try again. (If [LogIn]:verify_pin() == “invalid”, [LogIn]:log_in_retry() )
    - If they fail the pin 5 or more times, the ATM retains the card and alerts the Montana Savings Bank (MSB) bank. (If [LogIn]:log_attempts() >= 5, [Alert]:fraud_detection() )
        - The ATM retains the card. – [Alert]:confiscate_card()
        - The Central Database flags the account and notifies the MSB bank’s fraud/security system. – [Alert]:flag_account()

        - The customer can get assistance from a manager to re-obtain their card (refer to **Manage ATM** Use Case). – [ManageATM]:call_manager()

- When successful login is achieved, the ATM connects to the ATM's Central Database.(If [LogIn]:input_pin() == "valid", [CentralDatabase]:create_connection())
    - If the system cannot connect, a maintenance alert is triggered (refer to **Manage ATM** Use Case). – (If [CentralDatabase]:create_connection() == "invalid", [Alert]:maintenance_alert() )
- The Central Database determines if the customer is a member of the MSBbank as follows: – [LogIn]:determine_customer()
    - If the card is verified by the MSB bank, they are recognized and treated as a Regular Customer (If [LogIn]:determine_customer() = "Regular")
        - If the user has been a member of the MSB bank for three years or more, they are recognized and treated as a preferred customer. (If [LogIn]:determine_customer() == "Preferred")
    - If they are not verified to be a member of the MSB bank, the Central Database connects to other networks to obtain their information and registers them as a Non-MSB customer.(If [LogIn]:determine_customer() == "Not_MSB")
- The customer's identity, account type(s), and customer type are verified and recorded. – [LogIn]:record_customer()
- After their information for their account is obtained by the Central Database, they are connected to the ATM's Oracle DBMS (via **Oracle DBMS** Use Case) – [OracleDMBS]:connect_to_oracle()
- The login is logged into the Central Database. – [Central Database]:log_logins()
- The customer is then given the following options:
    - Check Balance, Deposit, Withdrawal, or Transfer (via **Check Balance**, **Deposit**, **Withdrawal**, **Transfer** Use Cases). – [Screen]:select_options()

- After the customer finishes using the ATM, the customer logs out and takes their card back. – [LogIn]:log_out()

Post-Conditions:

- The user session terminates normally, by timeout, or due to an error. – [LogIn]:no_actions(), [LogIn]:terminate_session()
- If login was successful: (If [LogIn]:log_in() == “true”)
  - The system displays the main transaction menu (Check Balance, Deposit, Withdrawal, Transfer). – [Screen]:select_options()
  - A secure session is established with the Oracle DBMS for account operations. – [OracleDMBS]:connect_to_oracle()
- After session termination (for any reason): – [LogIn]:log_out()
  - Any open connections to databases are securely closed. – [OracleDMBS]:disconnect_from_oracle()
  - The ATM resets to an idle state and displays the welcome screen. – [Screen]:reset_display()

# Use Case Description (Oracle DBMS)

Name: Oracle DBMS

Author: Andrew Vervaet

Last Update: 12/7/25

Pre-Conditions:

- The customer is logged in and connected to the central database – [CentralDatabase]:create_connection()
- The ATM is connected to the Central Database and operational. – [CentralDatabase]:create_connection()
- The ATM can access other ATMs from other branches and can access other banks for information. – [ManageATM]:connect_to_ATMs()

Dialog:

- The customer selects one of the following options: – [Screen]:select_options()
    - Check Balance, Deposit, Withdrawal, or Transfer (refer to **Check Balance**, **Deposit**, **Withdrawal**, or **Transfer** Use Cases) – [Transaction]: start_transaction(transaction_status)
    - If a transaction is selected (refer to **Deposit**, **Withdrawal**, **Transfer** Use Cases): (If [Transaction]:start_transaction(transaction_status) == "true")
        - The transaction is processed. – [Transaction]:process_transaction()
        - The transaction is entered into the ATM's Central Database using Oracle DBMS.– [Transaction]:transaction_into_database()
    - If Check Balance was selected (refer to **Check Balance** Use Case): – [CheckBalance]:show_balance()
        - The balance of that account is displayed. – [Screen]:display_balance()
- The Oracle DBMS sends the new information to the Central Database for the ATM. – [OracleDMBS]:log_info()

- A confirmation message is given and a printed receipt is made if selected (refer to **Print Receipt** Use Case). – [Screen]:confirmation_message(), [PrintReceipt]:ask_for_print()
    - If the user wants a receipt, they select print and receive one. – [PrintReceipt]:select_print(), [PrintReceipt]:print()
- The Oracle DBMS then waits for the next user action (Check Balance, Deposit, Withdrawal, or Transfer). – [OracleDBMS]:wait_mode()

Post-Conditions:

- After the customer finishes using the ATM, the customer logs out and takes their card back. – [LogIn]:log_out()
- All account transactions are processed by the Oracle DBMS and synchronized with the Central Database. – [OracleDBMS]:sync_CD()
- If a transaction was successful: (if [Transaction]:process_transaction() == "true")
    - The updated account balance is stored and displayed to the customer. – [OracleDBMS]:log_info(), [Screen]:display_balance()
- If a transaction failed: (if [Transaction]:process_transaction() == "false")
    - The failure is logged, and the system displays an appropriate error message. – [OracleDBMS]:log_info(), [Screen]:display_error()
    - The manager is alerted(refer to **Manage ATM** Use Case). – [ManageATM]:call_manager()
- After session termination (for any reason):
    - All open database connections are closed. – [OracleDBMS]:disconnect_from_oracle()
    - The ATM resets to an idle state, ready for the next customer. – [Screen]:reset_display()

# Use Case Description (Deposit)

Name: Deposit

Author: Seven Gunthrope

Last Update: 12/7/25

Pre-Conditions:

- The ATM is functioning properly.
- The user has an active bank account. [LogIn]:record_customer()
- The Oracle DBMS database is connected. [Central Database]:create_connection()
- Customers can only deposit money to their checking, savings, or money market accounts.
- The customer has at least one of the following: a checking account , savings account, or money market account. [Transaction]:check_account_type()
- The type of customer has been chosen for their type of deposit (preferred customer or regular customer). [LogIn]:record_customer()
- The ATM only accepts checks and cash (no coins).
- The user inserts their card and inputs their pin and it is sent to the **LOGIN** Use Case. [LogIn]:card_verification(), [LogIn]:input_pin()
- The database is then accessed and the credentials are then sent to the **ORACLE DBMS** Use Case.

Dialog:

- The user chooses a deposit. [Screen]:select_options()
- They select either their checking, savings, or money market account. [Deposit]:select_account()
- A menu is pulled up asking for a deposit of a check or cash: [Screen]:deposit_screen()
    - If Cash is selected, ATM asks for a cash to be inserted: (If [Screen]:deposit_screen()==”cash”, [Transaction]:start_transaction(transaction_status) )

        - The customer inserts their cash amount for their account. [Deposit]:check_cash_amount()
        - The ATM scans the cash: [Deposit]:scan_cash()
            - If the cash is valid, it accepts the cash and adds it to the total (If [Deposit]:check_cash_amount() == "successful")
            - If the cash is invalid, it sends the cash back to the customer to try again and displays an error (If [Deposit]:check_cash_amount() == ”failed”, [Alert]:invalid_cash() ) [Deposit]:display_deposit_alert()
        - When done, the customer confirms the total cash to deposit [Screen]:select_options()
        - The customer selects a deposit to finish inserting money into the account and deposit their cash into their account.[Deposit]:create_transaction_ID()
    - If a Check is selected, the ATM asks for a check to be inserted. (If [Screen]:deposit_screen()==”check”, [Transaction]:start_transaction(transaction_status) )
        - The customer inserts the check into the ATM.
        - The check is scanned by the ATM [Deposit]:check_scan_status()
        - The customer confirms the check scanned by the ATM [Screen]:select_options()
        - The check is accepted and the information is sent to the Central Database using the Oracle DBMS (refer to **ORACLE DBMS** Use Case). (If [Deposit]:check_scan_status() == ”True”), [Deposit]:create_transaction_ID()
            - The check is declined and the customer is told it was unable to scan. (If [Deposit]:check_scan_status == ”False”)

        [Alert]:invalid_scan_attempt()
        [Deposit]:display_deposit_alert()

    - If the customer is not a preferred customer, a 3-day hold will be placed on the deposit (If [LogIn]:determine_customer() != ”preferred”)

- If the customer is a preferred customer, there will be no hold placed on the deposit.
- If the customer is a non-MSB customer, they are charged $3.50 from their account. [Transaction]:charge_non_msb_fee()
- If the correct amount of cash is submitted, the transaction is then sent to the **Oracle DBMS** use case along with the transaction ID.

Post-Conditions:

- The previous balance and the current balance with the amount deposited is sent to the **ORACLE DBMS** use case. [OracleDBMS]:log_info()
- The user is given an option to terminate the session, print receipt, or make another transaction:
    - If the user chooses to print their receipt, the request is sent to the **ORACLE DBMS/PRINT RECEIPT** use case, then the session is automatically terminated. [Print Receipt]:select_print()
    - If the user chooses to terminate the session, they are logged out. [LogIn]:terminate_session()
    - If the user chooses to make another transaction, they are given the option to Check Balance, Deposit, Withdrawal, or Transfer (refer to **CHECK BALANCE**, **DEPOSIT**, **WITHDRAWAL**, or **TRANSFER** Use Cases) [Screen]:select_options()

# Use Case Description (Withdrawal)

Name: Withdrawal

Author: Seven Gunthrope

Last Update: 12/7/25

Pre-Conditions:

- The user has an active bank account.
- The user has a balance.
- The Oracle DBMS has an active connection. [OracleDBMS]:connect_to_oracle()
- The type of customer has been chosen for their type of deposit (preferred customer or regular customer). [LogIn]:determine_customer()
- For the consumer's loan account, the limit of the loan and the interest rate will be negotiated on an individual basis, and this information is stored in the central database.
- The withdrawal from all accounts is limited to $500 per day and once this notification displays if they try again their account will be flagged. [Withdraw]:withdrawal_limit()
- The withdrawal must be in multiples of $10.
- Customers can only withdraw money from their checking, savings, or money market accounts.
- The customer has at least one of the following: a checking account , savings account, or money market account. [Transaction]:check_account_type()
- The user inserts their card and inputs their pin and it is sent to the **LOGIN** Use Case. [LogIn]:card_verification(), [LogIn]:input_pin()
- The database is then accessed and the credentials are then sent to the **ORACLE**

Dialog:

- **DBMS** use case. [CentralDatabase]:create_connection()
- The user chooses withdrawal. [Screen]:select_options()

- If the customer already made a total of $500 in withdrawal that day, they are told they cannot make another withdrawal and are sent back to the main screen. (If [Withdrawal]:withdrawal_limit() > 500), [Alert]:flag_account() )
- The customer selects the account to withdraw from (checking, savings, or money market account). [Transaction]:check_account_type()
- The user's current balance is displayed: [Screen]:display_balance()
    - If the user does not have a balance they are told that they must deposit money into their account and if they believe this is an error to refer to the bank manager (via **MANAGE ATM** Use Case). [Error]:call_for_help()
- The user then chooses how much money they need to take out: [Withdrawal]:cash_withdrawl()
    - If the user enters a value that is greater than their balance they are told they do not have enough money in their account to complete this transaction unless they are a preferred customer then they are allowed to.
      (If ([Withdrawal]:cash_withdrawl() > [CheckBalance]:show_balance() )
      && [LogIn]determine_customer == "Not-MSB")
      [Alert] invalid_customer_type()
      [Withdrawal]display_withdrawal_alert()
    - If the number isn't divisible by ten, they are told, "Input a valid amount divisible by $10. Please try again." (if [Withdrawal]:cash_withdrawl()%10 != 0, [Error]:not_div_ten() )
- The ATM decides if the customer can make the Withdrawal. If at least one option is not satisfied, the transaction is canceled and they are asked to try again:
    - Non-preferred customers cannot withdraw money more than the account balance.
    - Customers can withdraw money from their consumer loan account, up to the limit established by the loan. [Withdrawal]:loan_amount()
    - Preferred customers can withdraw more money than the balance from their checking, savings, or money market accounts.

        - The money will automatically come from their consumer loan account, up to the limit established by the loan (If [Withdrawal]:cash_withdrawl() > [CheckBalance]:show_balance() && [LogIn]:determine_customer == ”Not-MSB”, [Withdrawal]:loan_amount()

- If the customer is a non-MSB customer, they are charged $3.50 from their account. [Transaction]:charge_non_msb_fee()
- Withdrawal Transaction ID is created [Withdrawal]:create_wtransaction_ID()
- Transaction is sent to Oracle DBMS (via **ORACLE DBMS** Use Case) [OracleDBMS]:log_info()
- The user is then given the money [Transaction]:return_money()

<u>Post-Conditions</u>:

- The new balance is sent to the **ORACLE DBMS/PRINT RECEIPT** use case with the amount taken out. [PrintReceipt]:select_print()
- The user is given an option to terminate the session, print receipt, or make another transaction:
    - If the user chooses to print their receipt, the request is sent to the **ORACLE DBMS/PRINT RECEIPT** use case, then the session is automatically terminated. [OracleDBMS]:disconnect_from_oracle()
    - If the user chooses to terminate the session, they are logged out. [LogIn]:terminate_session()
    - If the user chooses to make another transaction, they are given the option to Check Balance, Deposit, Withdrawal, or Transfer (refer to **CHECK BALANCE**, **DEPOSIT**, **WITHDRAWAL**, or **TRANSFER** Use Cases) [Screen]:select_options()

# Use Case Description (Transfer)

Name: Transfer

Author: Divine Dickson-Uwakwe

Last Update: 12/7/25

Pre-Conditions:

- The customer is authenticated through card and pin, and the ATM is connected to Central Database. [LogIn]:insert_card(), [LogIn]:verify_pin(), [CentralDatabase]:create_connection(),
- Determine if the account exists and belongs to the customer. – [LogIn]:determine_customer()
- Transfers are limited to allowed pairs: (a) between Checkings/Savings/Money Market Account; (b) from Checkings/Savings/Money Market Account to consumer loan; (c) from Checkings/Savings/Money Market Account to mortgage. (If [LogIn]:determine_customer() = "MSB")
- Non-MSB customers are not permitted to transfer (only deposit/withdraw allowed). (If [LogIn]:determine_customer() = "Non-MSB")
- Available balance must consider any check deposit holds for non-preferred customers (3-day hold). (If [LogIn]:determine_customer() = "Regular")

Dialog:

- The customer selects **Transfer** from the main menu. The ATM requests the **Source Account** and the customer chooses one (Checking, Savings, or Money Market). [Transfer]:select_source_account()
- The ATM requests the **Destination Account**; the customer chooses one:
  - Another deposit account (Checking/Savings/Money Market), or
  - A Consumer Loan or Mortgage (payment). [Transfer]:select_destination_account()
- The ATM validates that the source/destination pair is allowed. [Transfer]:validate_account_pair()

    - If not allowed, the ATM shows: “Selected accounts cannot be used for this transfer.” and returns to Step 2.
- The ATM prompts for the **Transfer Amount**. [Transfer]:input_amount()
- The ATM retrieves **available balances** from the Central Database, applying deposit-hold rules (no holds if Preferred customer; 3-day check holds if Non-Preferred).
- The ATM validates the amount: [Transfer]:validate_amount()
    - If available balance < amount: show “Insufficient available balance.” and return to Step 5. [Error]:display_error()
- The ATM displays a confirmation summarizing: source, destination, amount, and any notes. The customer confirms the transfer. [Transfer]:confirm_transfer(), [Transfer]:record_transfer()

<u>Post-Conditions</u>:

- On success, the ATM displays **“Transfer Successful”** and shows updated balances. Then the transfer transaction is successfully recorded in the **Central Database (Oracle DBMS)**. [Transfer]:show_confirmation()
- The customer decides to choose “Yes” to print receipt for successful transfer or “No” to not print receipt [Screen]:select_options()
- The session remains active, allowing the customer to return to the main menu, log out or card is returned.

# Use Case Description (Check Balance)

Name: Check Balance

Author: Divine Dickson-Uwakwe

Last Update: 12/7/25

Pre-Conditions:

- The customer has successfully logged in to the ATM using a valid card and PIN. –
- The ATM is connected to the **Central Database (Oracle DBMS)** and is functioning properly.
- The customer's account information is validated. [LogIn]:determine_customer()
- For **Preferred Customers**, all funds are immediately available. (If [LogIn]:determine_customer() = "Preferred")
- For **Non-Preferred Customers**, any check deposits within the last three days are subject to a temporary hold and will not appear in the available balance. (If [LogIn]:determine_customer() = "Regular")
- The **Oracle DBMS** must be online and capable of retrieving balance data. [OracleDBMS]:connect_to_oracle()
- Customers can check the balance of their checking, savings, money market, Certificate of Deposit, consumer loan, or mortgage account.

Dialog:

- The customer selects **Check Balance** from the main menu.
- The ATM connects to the **Central Database** to retrieve the list of the customer's accounts. [CentralDatabase]:create_connection()
- The ATM displays the available account types and prompts the customer to select one: Checking, Savings, Money Market, Consumer Loan, Certification of Deposit, Mortgage. [Screen]:select_options()
- The ATM validates that the selected account type is supported for balance inquiry.

    - If the account type is not supported, the ATM displays: “Selected account cannot be accessed for balance inquiry.” and returns to Step 2. [Transaction]:check_account_type()
- If the selected account is a **Checking** or **Savings**, the ATM retrieves and displays the balance. [CheckBalance]:show_balance()
- If the selected account is a **Loan** or **Mortgage**, the ATM retrieves and displays:
    - Outstanding balance [CheckBalance]:show_balance()
    - Next payment due date and amount (if available).
- If the selected account is a **Certificate of Deposit (CD)**, the ATM displays only the balance. [CheckBalance]:show_balance()

Post-Conditions:

- The customer decides to choose “Yes” to print receipt for the check balance or “No” to not print receipt [PrintReceipt]:select_print()
- After displaying or printing the balance, the session continues, allowing the customer to perform another transaction or log out. [Screen]:reset_display(), [CheckBalance]:log_out()

# Use Case Description (Print Receipt)

Name: Print Receipt

Author: Andrew Vervaet

Last Update: 12/7/25

Pre-Conditions:

- The customer has made transactions to print a receipt. – [Transaction]:process_transaction()
- The customer has logged into their account. – [LogIn]:log_in()
- There is printer paper in the machine for the receipts to be printed.
- The ATM can print the receipt.
- An ATM manager is present for the ATM if an issue arises for the printer.

Dialog:

- The customer logs in (refer to **LogIn** Use Case). – [LogIn]:log_in()
- The customer makes a transaction: – [Transaction]:process_transaction()
    - The choice is given to the customer to print a receipt or not for that transaction. – [PrintReceipt]:ask_for_print()
    - The customer selects whether they want the receipt to be printed or not – [PrintReceipt]:select_print()
    - The customer receives a receipt if they select "Print." (if [PrintReceipt]:select_print() == "true", [PrintReceipt]:print() )
- The customer logs out and the session is terminated. The print is logged – [LogIn]:log_out()

Post-Conditions:

- The customer's session terminates and connections to servers are removed. – [LogIn]:terminate_session()
- If the receipt was printed successfully: – [PrintReceipt]:print()
    - The customer receives a printed copy of their selected transaction(s).

- If the printer was out of paper or encountered an error: – (If [PrintReceipt]:print() == “error”, [Error]:no_pp()
    - The ATM displays a notification and logs the error. – [Screen]:no_pp_error()
    - The manager is alerted (refer to **Manage ATM** Use Case). – [Alert]:maintenance_alert()
    - The manager adds paper to the machine or fixes the printer paper roll. – [PrintReceipt]:fix_printer()
- After session termination: – [LogIn]:terminate_session()
    - The ATM resets to its idle state and is ready for the next user. – [Screen]:reset_display()

# Use Case Description (Manage ATM)

Name: Manage ATM

Author: John Sam-Bruce

Last Update: 12/7/25

Pre-Conditions:

- The ATM is operational and connected to the Central Database
- A maintenance staff or authorized bank administrator has valid credentials to access the maintenance panel.- [ManageATM]:authenticate_staff()
- Customer transactions have been logged properly prior to maintenance actions.
- There is an issue or scheduled task that requires ATM management. - [Alert]:maintenance_alert()

Dialog:

- The authorized maintenance staff inserts the Manager Access Card or enters Admin Credentials to access the maintenance mode. - [ManageATM]:authenticate_staff()
- The ATM system validates the credentials through the Central Database. - [CentralDatabase]:validate_admin()
    - If the credentials are invalid, the access is denied and the attempt is logged. -(If [ManageATM]:authenticate_staff() = “false”)
- Once verified, the ATM displays the Maintenance Menu with the following options: - [ManageATM]:display_maintenance_menu()
    - Cash Refill: Add or verify the total cash available for withdrawals. - [ManageATM]:update_database()
    - Printer Maintenance: Check paper level or replace receipt paper. - [PrintReceipt]:fix_printer()
    - Error Log Review: Display or print hardware/software error logs - [ManageATM]:view_error_logs()

    - System Restart / Diagnostics**:** Perform a restart or run basic hardware diagnostics. - [ManageATM]:run_diagnostics()
- The staff selects an option:
    - If Cash Refill: The system prompts for the number of bills inserted per denomination, updates inventory, and logs the refill.
    - If Printer Maintenance: The system checks paper status and resets the "out of paper" flag once new paper is inserted.
    - If Error Log Review: The system displays the last 20 error events with timestamps for troubleshooting.
    - If System Restart / Diagnostics: The ATM temporarily goes offline, runs internal tests, and reconnects after successful checks.
- The system updates the Central Database with all maintenance actions. - [ManageATM]:update_database()
- The staff exits maintenance mode. - [Screen]:reset_display()

Post-Conditions:

- ATM status is updated to Active or Under Maintenance depending on the operation.
- Any maintenance or troubleshooting actions are recorded in the Central Database with date, time, and staff ID. - [ManageATM]:sync_logs()
- Hardware issues resolved. - [ManageATM]:terminate_maintenance_mode()
- The ATM returns to the customer interface screen once maintenance mode is exited. - [ManageATM]:connect_to_ATMs()

<u>Class Diagrams</u>

**LogIn**

**Description:** The LogIn class holds operations and attributes that help the customer access their account information.

**Attributes:**
- pin_num: int
- customer_type: string {Regular, Preferred, Not-MSB}
- log_attempt_num: int
- inactivity: bool
- log_in_status: bool

**Operations:**
+ insert_card()
+ card_verification()
+ input_pin()
+ verify_pin()
+ no_actions()
+ log_in()
+ determine_customer()
+ log_out()
+ terminate_session()
+ record_customer()
+ log_attempts()
+ log_in_retry()

**ManageATM**

**Description:** The manageATM class hold operations that allow maintenance staff or admins to manage and service the ATM systems

**Attributes:**
- ATM_num: int
- manager_num: int
- staff_id:int

**Operations:**
+ connect_to_ATMs()
+ authenticate_staff()
+ run_diagnostics()
+ display_maintenance_menu()
+ view_error_logs()
+ update_database()
+ sync_logs()
+ terminate_maintenance_mode()
+ call_manager()
+ handle_cash_refill()

**Error**

**Description:** The Error class holds attributes and operations that track and handle different error situations.

**Attributes:**
- error_number: int
- connection: bool

**Operations:**
+ call_for_help()
+ no_pp()
+ not_div_ten()

| **Screen** | **OracleDBMS** | **CentralDatabase** |
|---|---|---|
| **Description:** The Screen class holds the operations that allow information to be displayed on the ATM screen. | **Description:** The OracleDBMS class holds the operations and attributes of the functions needed to connect customer information and accounts from the Central Database to the ATM. | **Description:** The CentralDatabase class holds the operations for holding account information and connecting to the OracleDMBS so the ATM can access the held account information. |
| **Attributes:**<br>- option_num: int | **Attributes:**<br>- mode_number: int | **Attributes:** |
| **Operations:**<br>+ select_options()<br>+ reset_display()<br>+ display_balance()<br>+ confirmation_message()<br>+ display_error()<br>+ no_pp_error()<br>+ deposit_screen() | **Operations:**<br>+ connect_to_oracle()<br>+ disconnect_from_oracle()<br>+ log_info()<br>+ wait_mode()<br>+ sync_CD() | **Operations:**<br>+ create_connection()<br>+ log_logins()<br>+ validate_admin() |

| <u>**CheckBalance**</u> | <u>**PrintReceipt**</u> | <u>**Alert**</u> |
|---|---|---|
| **Description:** The CheckBalance class represents a transaction where the customer requests to view the current balance in their account. | **Description:** The PrintReceipt class holds the operations and attributes for allowing customers to print a receipt after making a transaction between accounts. | **Description:** The Alert class holds the operations and attributes that send important alerts to management if something serious happens to an account. |
| **Attributes:**<br>- balance_num: double | **Attributes:**<br>- printer_status: string {Full, Good, Empty}<br>- print_detect: bool | **Attributes:**<br>- alert_number: int<br>- account_flaged: bool<br>- fraud_detect: bool<br>- alert_detect: bool<br>- maintenance_detect: bool |
| **Operations:**<br>+ show_balance() | **Operations:**<br>+ ask_for_print()<br>+ select_print()<br>+ print()<br>+ fix_printer()<br>+ handle_printer_maint() | **Operations:**<br>+ fraud_detection()<br>+ confiscate_card()<br>+ flag_account()<br>+ invalid_cash()<br>+ maintenance_alert()<br>+ invalid_scan_attempt()<br>+ invalid_customer_type() |

**Withdrawal**

**Description:** The withdrawal class holds operations that allow the customer to withdraw cash.

**Attributes:**
- withdrawal_limit: double

**Operations:**
+ cash_withdrawl()
+ withdrawal_limit()
+ create_wtransaction_ID()
+ loan_amount()
+display_withdrawal_alert()

**Deposit**

**Description:** The deposit class holds operations that allow the customer to deposit cash or checks into their account.

**Attributes:**
- select_account: string {checking,savings, money_market}
- scan_status: bool
- dep_num: double

**Operations:**
+ select_account()
+ check_cash_amount()
+ check_scan_status()
+ create_transaction_ID()
+ display_deposit_alert()
+ scan_cash()

**Transfer**

**Description:** The Transfer class represents a transaction where money is moved from the customer's account to another specified account.

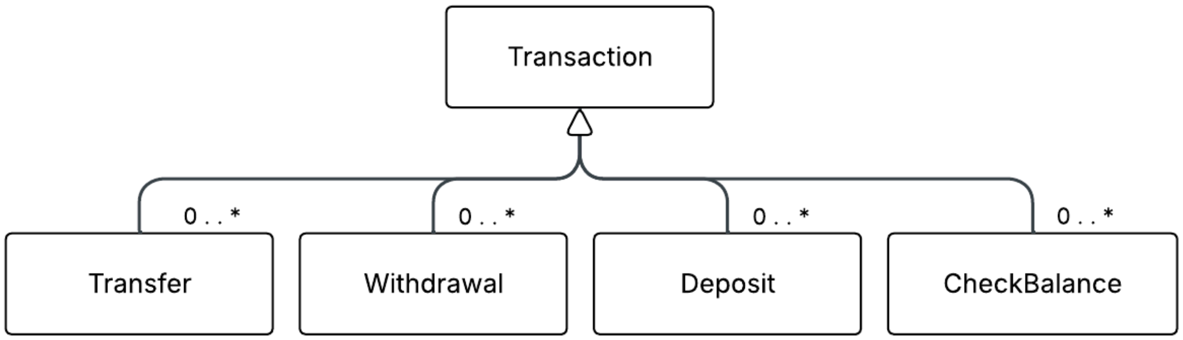
**Attributes:**
- input_transfer_num: double

**Operations:**
+ determine_the_customer()
+ select_source_account()
+ select_destination_account()
+ validate_account_pair()
+ input_amount()
+ validate_amount()
+ confirm_transfer()
+ record_transfer()
+ show_confirmation()

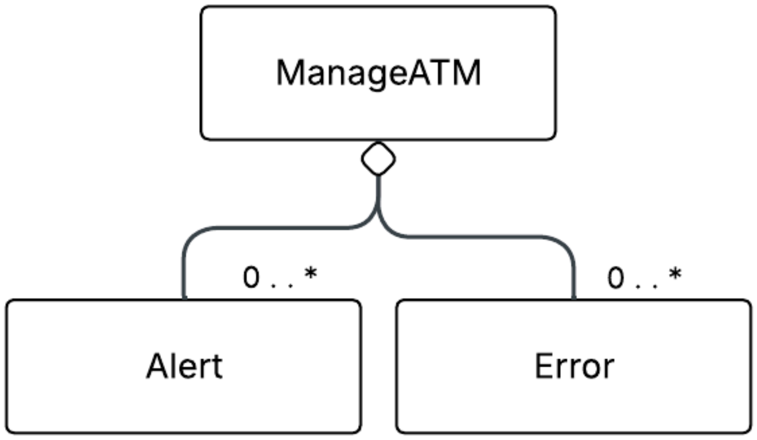
| **Transaction** |
|---|
| **Description:** This class holds operations and attributes that allow transactions to function when the customer uses the ATM. |
| **Attributes:**<br># transaction_status: bool<br># msb_fee: double<br># transactionID: int<br># account_type: string {checking, savings, money_market, consumer_loan, mortgage} |
| **Operations:**<br>+ start_transaction(transaction_status)<br>+ process_transaction()<br>+ transaction_into_database()<br>+ charge_non_msb_fee()<br>+ return_money()<br>+ check_account_type() |

## Class Relationships

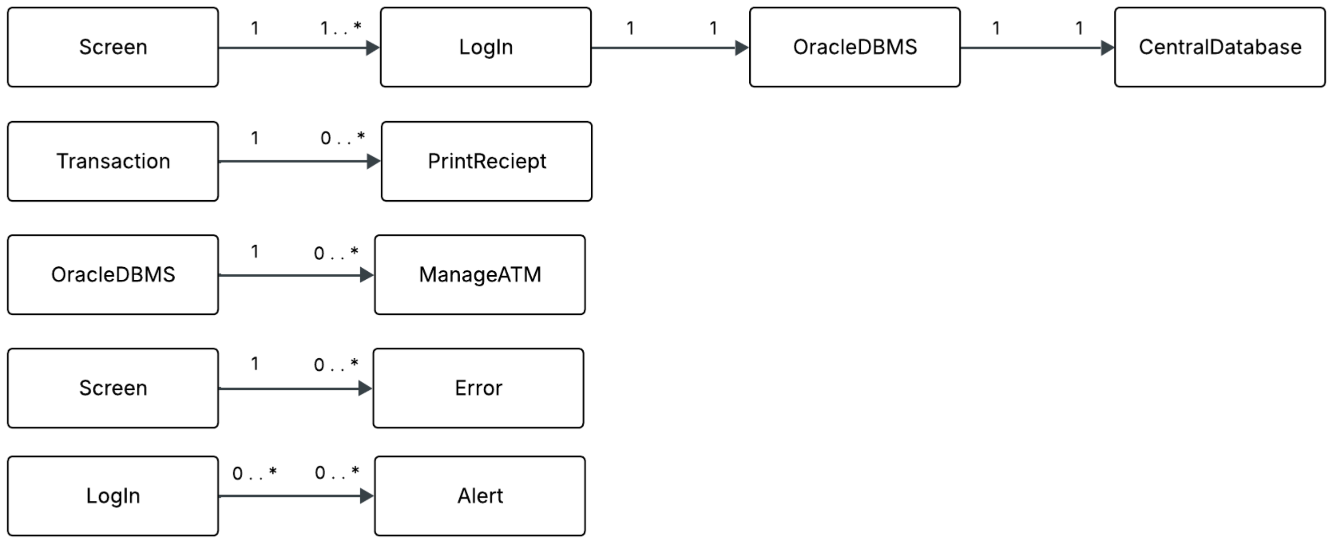
Inheritance Relationships:



Aggregation Relationships:



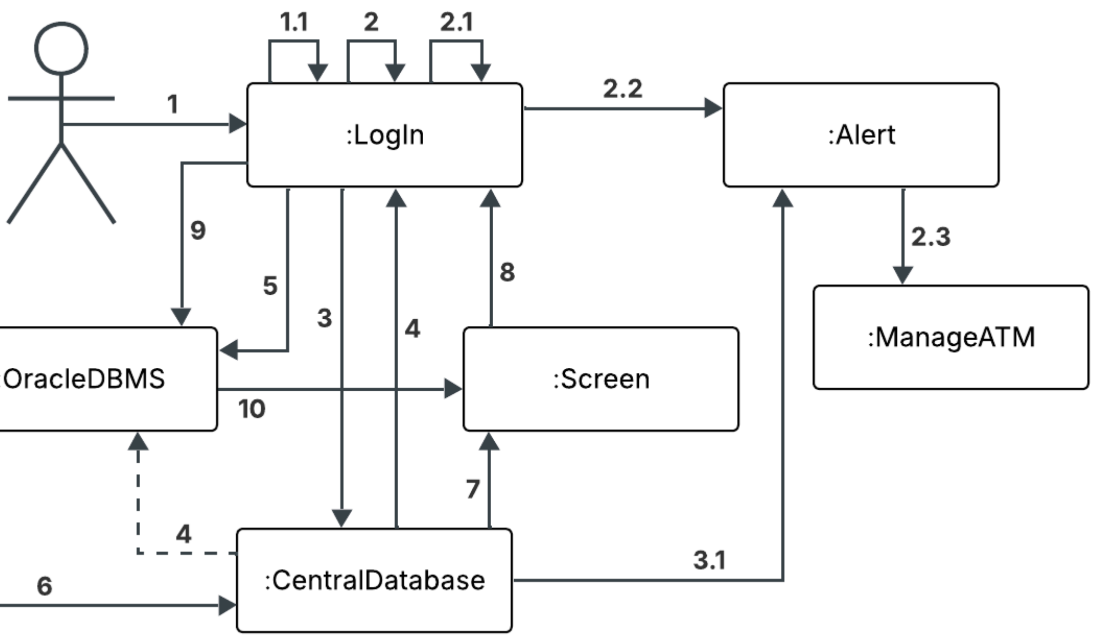
## Association Relationships:



# Object and Communication Diagrams

## **“LogIn” Communication Diagram:**

Diagram:



Operation Sequence:

1: insert_card()

1: card_verification()

 1.1: log_in_retry()    {card_verification() = "Invalid"}

2: input_pin()

2: log_attempts()

2: verify_pin()

 2.1: log_in_retry()    {verify_pin() = "invalid" && log_attempt_num < 5}

 2.2: fraud_detection()    {fraud_detect = “True” && log_attempt_num >= 5}

 2.2: confiscate_card()

 2.2: flag_account()

 2.3: call_manager()

3: create_connection()                    {input_pin() = “valid”}

    3.1: maintenance_alert()    {maintenance_detect = "True"}

4: determine_customer()

4: return customer_type        {With customer_type, return result to OracleDBMS to determine allowed transactions}

4: record_customer()

5: connect_to_oracle()

6: log_logins()

7: select_options()

8: log_out()

8: terminate_session()

9: disconnect_from_oracle()

10: reset_display()

<u>Scenario Sequences:</u>

1, 1.1 – Card is registered invalid and asked to try the card again

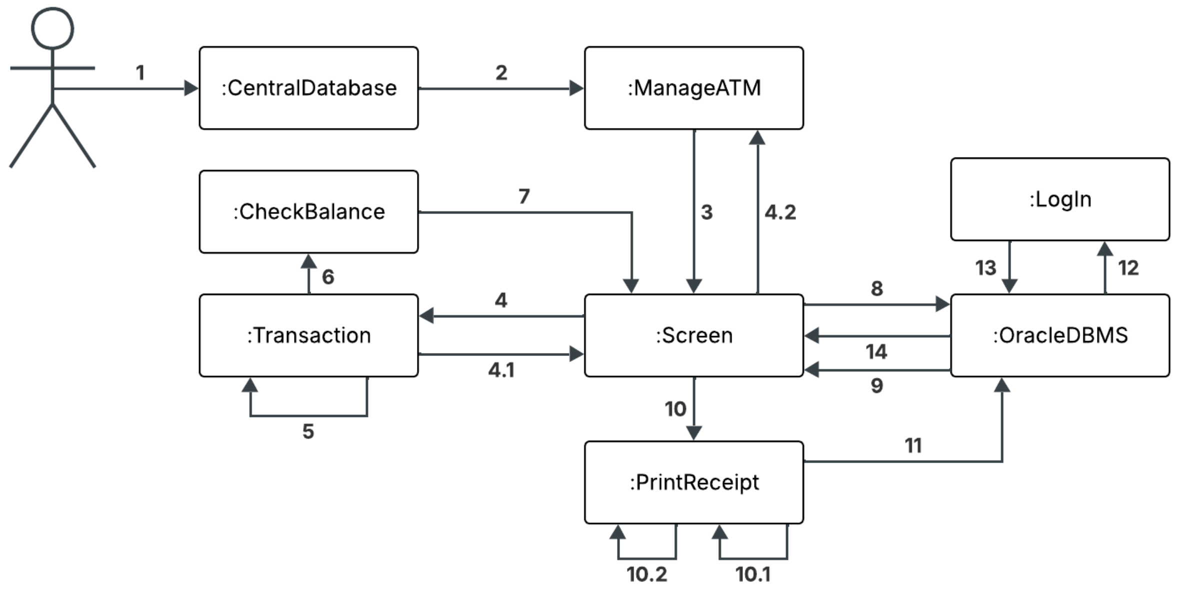
1, 2, 3, 4, 5, 6, 7, 8, 9, 10 – User logs in successfully, then logs out after use with no issues.

1, 2, 2.1, 2.2, 2.3 – User’s actions are detected to be fraudulent, and the card is confiscated.

1, 2, 3, 3.1 – The user logs in successfully, but the connection to the Central Database cannot be made.

## “Oracle DBMS” Communication Diagram:

Diagram:



Operation Sequence:

1: create_connection()

2: connect_to_ATMs()

3: select_options()

4: start_transaction(transaction_status)

    4.1: display_error()   {start_transaction(transaction_status) = “False”}

    4.2: call_manager()

5: process_transaction()   {start_transaction(transaction_status) = “True”}

5: transaction_into_database()

6: show_balance()

7: display_balance()

8: log_info()

8: sync_CD()

9: confirmation_message()

10: ask_for_print()

```
        10.1: select_print()            {print_detect = "True"}
        10.2: print()
11: wait_mode()
12: log_out()
13: disconnect_from_oraqcle()
14: reset_display()
```

<u>Scenario Sequences:</u>

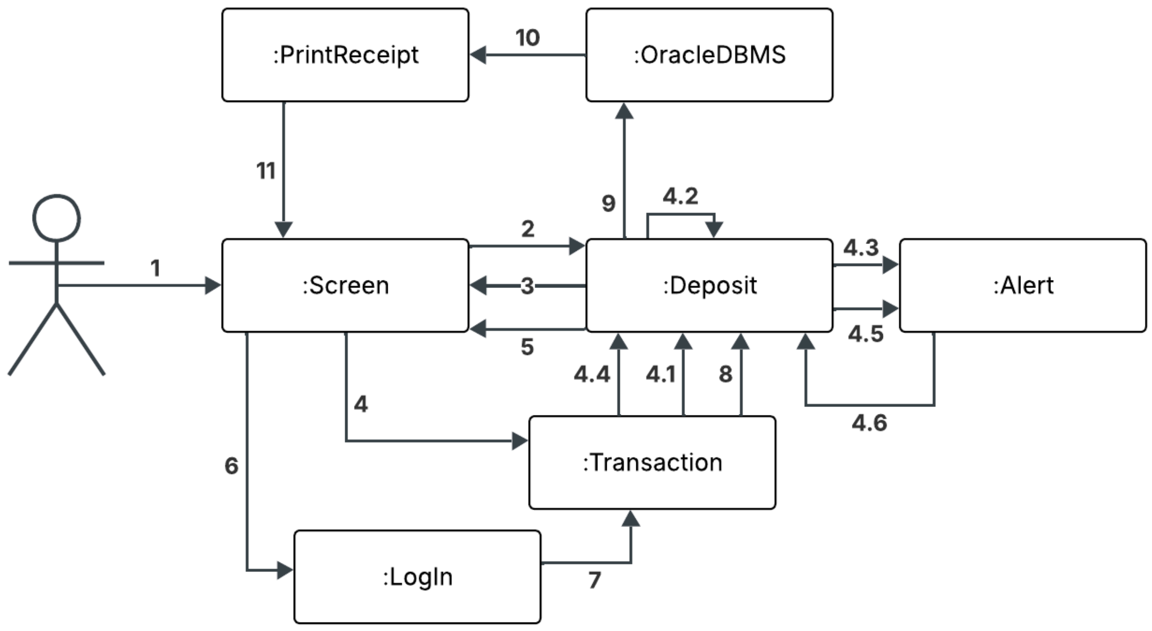
1, 2, 3, 4, 4.1, 4.2 – There is an issue with the transaction in the OracleDBMS, and a manager is called to assist.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 – The OracleDBMS successfully processes the transaction, no receipt is printed, and the user logs out.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10.1, 10.2, 11, 12, 13, 14 – The OracleDBMS successfully processes the transaction, a receipt is printed, and the user logs out.

## **<u>“Deposit” Communication Diagram:</u>**

<u>Diagram:</u>



<u>Operation Sequence:</u>

1: select_options()

2: select_account()

3: deposit_screen()

4: start_transaction(transaction_status)

| | |
|---|---|
| 4.1: check_cash_amount() | {deposit_screen()=”cash”} |
| 4.2: scan_cash() | {check_cash_amount() = ”successful”} |
| 4.3: invalid_cash() | {check_cash_amount() = ”failed”} |
| 4.4: check_scan_status() | {deposit_screen()=”check”} |
| 4.5: invalid_scan_attempt() | {check_scan_status()= ”False”} |
| 4.6: display_deposit_alert() | |

5: select_options()

6: determine_customer()

7: charge_non_msb_fee() {determine_customer() = ”non-MSB”}

8: create_transaction_ID() {check_scan_status()=”True”}

9: log_info()

10: select_print()

11: select_options()

Scenario Sequences:

1, 2, 3, 4, 4.1, 4.2, 5, 6, 8, 9, 10, 11 – A customer completes a cash deposit.

1, 2, 3, 4, 4.3, 4.4, 5, 6, 8, 9, 10 – A customer completes a check deposit.

1, 2, 3, 4, 4.1, 4.2, 4.3, 4.6 – A customer tries to complete a cash deposit and enters a invalid amount.

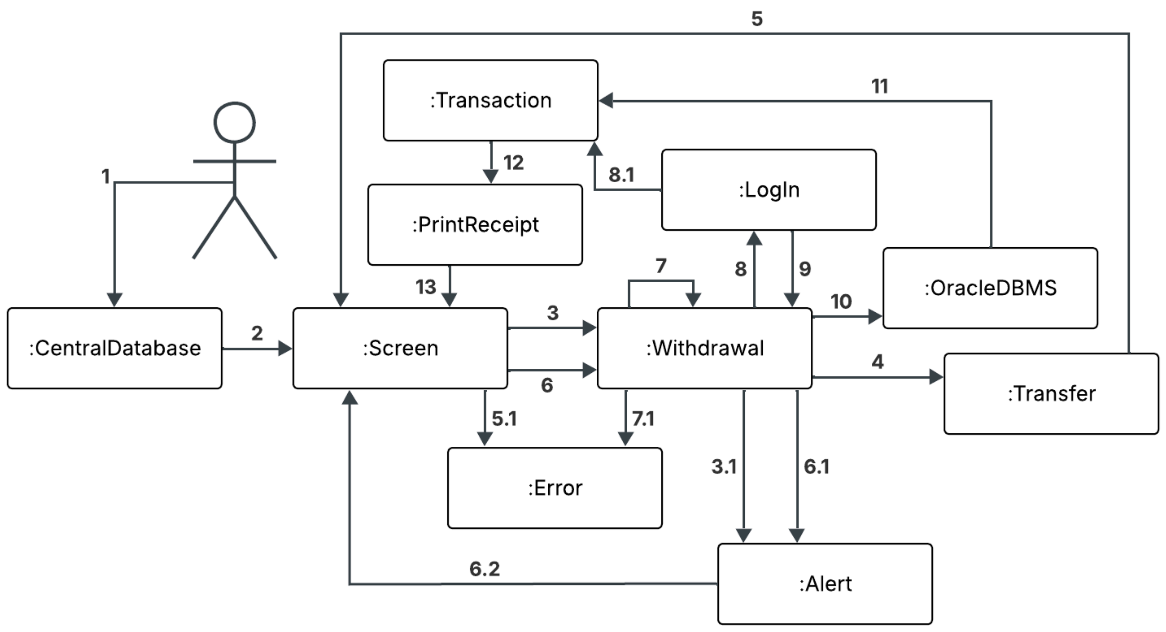
1, 2, 3, 4, 4.4, 4.5, 4.6 – A customer tries to complete a check deposit and the check does not register with the ATM.

1, 2, 3, 4, 4.1, 4.2, 5, 6, 7, 8, 9, 10, 11 – A non-msb customer completes a cash transaction and is given a fee.

1, 2, 3, 4, 4.3, 5, 6, 7, 8, 9, 10, 11 – A non-msb customer completes a check transaction and is given a fee.

## **“Withdrawal” Communication Diagram:**

Diagram:



Operation Sequence:

1: create_connection()

2: select_options()

3: withdrawal_limit()

3.1: flag_account() {withdrawal_limit() > 500}

4: check_account_type()

5: display_balance()

5.1: call_for_help() {display_balance = ”none”}

6: cash_withdrawl()

6.1: invalid_customer_type() {cash_withdrawl() > show_balance() && determine_customer = ”Not-MSB”}

6.2: display_withdrawal_alert()

7: loan_amount() {cash_withdrawl() > show_balance() && determine_customer != ”Not-MSB” }

7.1: not_div_ten()     {cash_withdrawl()%10 != 0}

8: determine_customer()

8.1: charge_non_msb_fee() {determine_customer() = ”non-MSB”}

9: create_wtransaction_ID()

10: log_info()

11: return_money()

12: select_print()

13: select_options()

<u>Scenario Sequences:</u>

1, 2, 3, 3.1 – Customer starts transaction but has already taken out more than $500.

1, 2, 3, 4, 5, 5.1 – Customer starts transaction but does not have a balance.

1, 2, 3, 4, 5, 6, 6.1, 6.2 – Non-MSB customers try to withdraw.

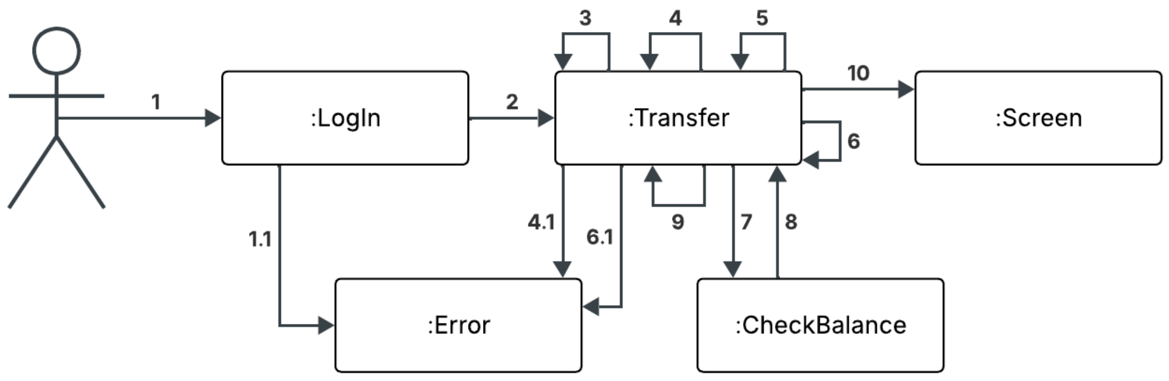
1, 2, 3, 4, 5, 6, 7, 7.1 – Amount withdrawn is not divisible by 10.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 – The customer completes the withdrawal with no MSB fee.

1, 2, 3, 4, 5, 6, 7, 8, 8.1, 9, 10, 11, 12, 13 – The customer completes the withdrawal with an MSB fee.

## **“Transfer” Communication Diagram:**

Diagram:



Operation Sequence:

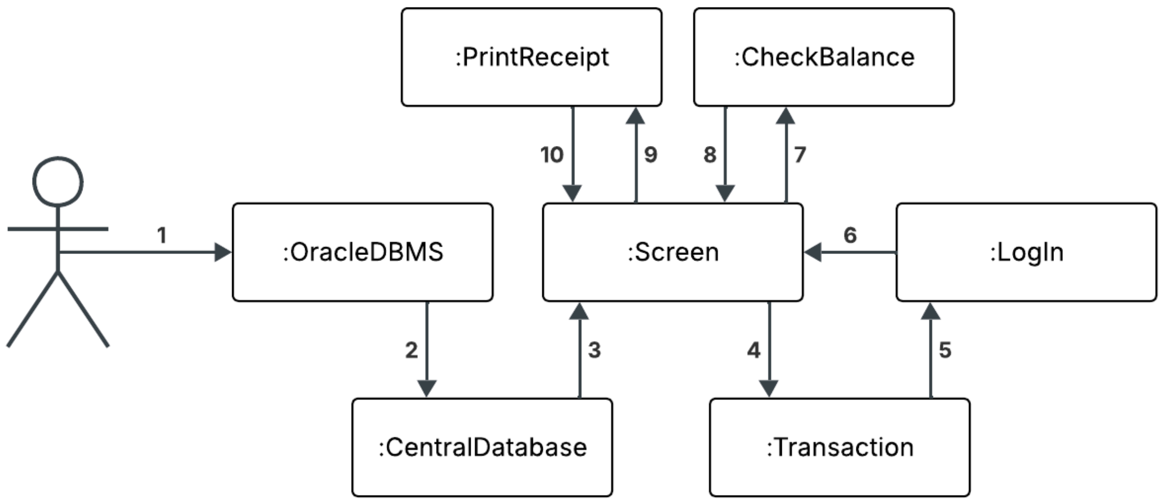
| | |
|---|---|
| 1:determine_customer() | {check from login} |
| 1.1: display_error() | {customer_type == "Not_MSB"} |
| 2: select_source_account() | |
| 3: select_destination_account() | |
| 4: validate_account_pair() | |
| 4.1: display_error() | {validate_account_pair == "invalid"} |
| 5: input_amount() | |
| 6: validate_amount() | |
| 6.1: display_error() | {validate_amount == "invalid"} |
| 7: show_balance() | |
| 8: confirm_transfer() | |
| 9: show_confirmation() | |
| 10: select_options() | |

Scenario Sequence:

1, 1.1 – Non-MSB customer tries to transfer
1, 2, 3, 4, 4.1 – Invalid source/destination account pair
1, 2, 3, 4, 5, 6, 6.1 – Insufficient / invalid amount
1, 2, 3, 4, 5, 6, 7, 8, 9, 10 – Successful transfer between allowed accounts

## “Check Balance” Communication Diagram:

Diagram:



Operation Sequence:

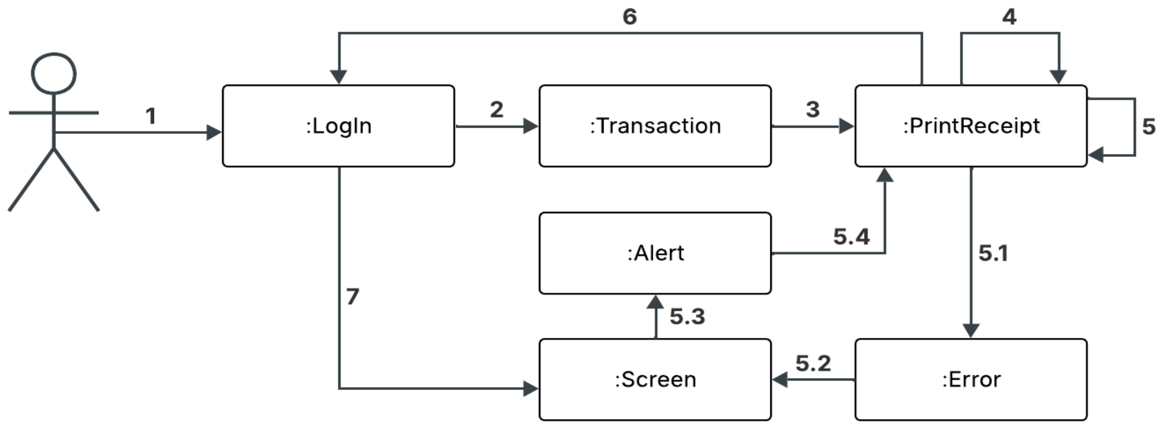
| | |
|---|---|
| 1: connect_to_oracle() | |
| 2: create_connection() | |
| 3: select_options() | |
| 4: check_account_type() | |
| 5: determine_customer() | {from login} |
| 6: select_options() | {customer chooses Check Balance} |
| 7: show_balance() | |
| 8: display_balance() | |
| 9: select_print() | |
| 10: reset_display() | |

Scenario Sequences:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10 – Successful balance inquiry with printed receipt.
1, 2, 3, 4, 5, 6, 7, 8, 10 – Successful balance inquiry without printed receipt.

## **“Print Receipt” Communication Diagram:**

Diagram:



Operation Sequence:

1: log_in()

2: process_transaction()

3: ask_for_print()

4: select_print()

5: print()                    {print_detect = "True"}

    5.1: no_pp()                    {printer_status = "Empty"}

    5.2: no_pp_error()

    5.3: maintenance_alert()

    5.4: fix_printer()

6: log_out()

6: terminate_session()

7: reset_display()

Scenario Sequences:

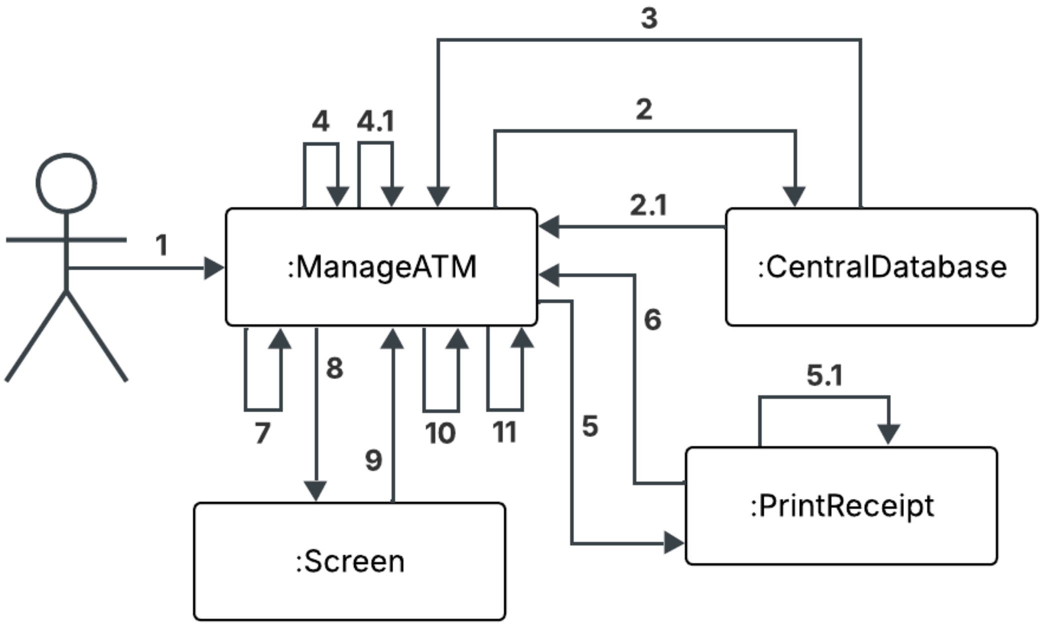
1, 2, 3, 4, 5, 6, 7 – Receipt printed successfully

1, 2, 3, 4, 6, 7 – Customer declines receipt

1, 2, 3, 4, 5, 5.1, 5.2, 5.3, 5.4 – Printer has an error and needs fixing/refill.

## **“Manage ATM” Communication Diagram:**

Diagram:



Operation Sequence:

1: authenticate_staff()

2: validate_admin()

    2.1: sync_logs()    {status = “invalidated}

3: display_maintenance_menu()

4: handle_cash_refill()

    4.1: update_database()

5: handle_printer_maint()

    5.1: fix_print()

6: view_error_logs()

7: run_diagnostics()

8: reset_display()

9: sync_logs()

10: terminate_maintenance_mode()

11: connect_to_ATM()

Scenario Sequences:

1, 2, 2.1 – Authentication fails.

1, 2, 3, 4, 4.1, 8, 9, 10, 11 – Successful cash refill.

1, 2, 3, 5, 5.1, 8, 9, 10, 11 – Successful printer maintenance.

1, 2, 3, 6, 8, 9, 10, 11 – Successful error log review

1, 2, 3, 7, 8, 9, 10, 11 – Successful system diagnostic

# Implementation/Code Generation

```
// Handles all operations related to customer login and authentication
class LogIn {
public:
        void insert_card();             // Registers customer's cars
        void card_verification();       // Verifies the card is valid
        void input_pin();               // Reads inputted pin
        void verify_pin();              // Verifies inputted pin
        void no_actions();              // Activates if the user doesn't do anything for a set
                                        // time, logging them out for security
        void log_in();                  // Logs the user into their account
        void determine_customer();              // Determines the customers type (Regular,
                                                // Preferred, or Not-MSB)
        void log_out();                 // Logs the user out of their account
        void terminate_session();       // Disconnects the ATM from the central database
        void record_customer();         // Records the customer used the ATM
        void log_attempts();            // Logs the users login attempts for security
        void log_in_retry();            // Let's the user retry their login
private:
        int pin_num;                    // Holds user's pin number
        string customer_type;           // Regular, Preferred, or Not-MSB
        int log_attempt_num;            // Holds number of linen attempts
        bool inactivity;                // Tells whether the user is active or inactive
        bool log_in_status;             // Determines the log-in status of the customer
};
```

```
// Operations for maintenance staff or admins to manage and service the ATM
class ManageATM {
public:
        void connect_to_ATMs();   // Establish connection to all ATM machines
        void authenticate_staff();    // Verifies credentials of maintenance staff/admin
        void run_diagnostics();       // Runs system diagnostic checks on the ATM
        void display_maintenance_menu();  // Displays maintenance options for staff
        void view_error_logs();       // Views error logs stored in the ATM
        void update_database();       // Updates the central database with ATM information
        void sync_logs();             // Synchronizes local ATM logs with central server
        void terminate_maintenance_mode();       // Ends maintenance mode and restores
                                                 // normal ATM operation
        void call_manager();          // Contacts the manager for critical issues
        void handle_cash_refill();   // Manages refilling cash in the ATM
private:
        int ATM_num;         // ATM identification number
        int manager_num;    // Manager identification number
        int staff_id;             // Maintenance staff ID
};
```

```
// Tracks and handles different error situations in the ATM
class Error {
public:
        void call_for_help();         // Initiates help call in case of serious error
        void no_pp();                 // Handles “no paper” printer errors
        void not_div_ten();           // Handles errors when cash is not divisible by ten
Private:
        int error_number;             // Error code/number
        bool connection;              // True if ATM is connected to network, false if
                                      // disconnected
};
```

```
// Displays information to the customer on the ATM screen
class Screen {
public:
        void select_options();          // Allows customer to select options from menu
        void reset_display();           // Resets the screen to default state
        void display_balance();         // Shows the account balance
        void confirmation_message();    // Displays confirmation messages for transactions
        void display_error();           // Shows error messages to the customer
        void no_pp_error();             // Tells the user there is no printer paper left
        void deposit_screen();          // Shows deposit transaction screen
private:
        int option_num;                 // Stores the currently selected option
};
```

```
// Connects ATM to the central database for account information
class OracleDBMS {
public:
        void connect_to_oracle();            // Establishes connection to the Oracle DMBS
        void disconnect_from_oracle();       // Disconnects from the Oracle DMBS
        void log_info();                 // Logs database access and operations for auditing
        void wait_mode();                // Puts the database connection in wait/idle mode
        void sync_CD();                  // Synchronizes data with the central database
private:
        int mode_number;                 // Stores the current mode of the database connection
};
```

```
// Holds account information and connects to OracleDBMS for ATM access
class CentralDatabase {
Public:
        void create_connection();   // Creates a connection between the Oracle DBMS and
                                    // central database
        void log_logins();          // Logs login attempts to the database
        void validate_admin();      // Validates admin access credentials
};
```

```
// General transaction class handling core transaction operations
class Transaction{
public:
        void start_transaction(bool transaction_status);  // Starts transaction process
        void process_transaction();                 // Processes the transaction
        void transaction_into_database();           // Stores transaction in database
        void charge_non_msb_fee();                  // Charges fee if non-MSB customer
        void return_money();                        // Returns cash if transaction fails
        void check_account_type();                  // Checks the type of account
private:
        bool transaction_status;      // True if transaction active, false otherwise
        double msb_fee;               // Fee applied for non-MSB customers
        int transactionID;            // Unique transaction ID
        string account_type;          // Account type: checking, savings, money_market,
                                      // consumer_loan, mortgage
};
```

```
// Allows customers to view their account balance
class CheckBalance : public Transaction{
public:
        void show_balance();         // Displays current balance to the customer
private:
        double balance_num;          // Stores the current account balance
};
```

```
// Handles printing receipts for transactions
class PrintReceipt {
public:
        void ask_for_print();          // Prompts the customer if they want to print a receipt
        void select_print();        // Allows the customer to select receipt printing options
        void print();                  // Prints the transaction receipt
        void fix_printer();            // Performs basic printer troubleshooting
        void handle_printer_maint();  // Manages maintenance or servicing of the printer
private:
        string printer_status;         // Status of the printer: Full, Good, or Empty
        bool print_detect;             // Detects whether printing is possible
};
```

```
// Sends important alerts to management for critical issues
class Alert {
public:
        void fraud_detection();       // Detects fraudulent activity
        void confiscate_card();       // Confiscates the card in emergency situations
        void flag_account();          // Flags an account for review
        void invalid_cash();          // Alerts invalid cash insertion
        void maintenance_alert);      // Sends maintenance alerts
        void invalid_scan_attempt();          // Alerts when card scanning fails
        void invalid_customer_type();         // Alerts if customer type is invalid
private:
        int alert_number;             // Unique alert ID
        bool account_flaged;          // Indicates if account is flagged
        bool fraud_detect;            // True if fraud is detected
        bool alert_detect;            // True if an alert has been triggered
        bool maintenance_detect;      // True if maintenance alert exists
};
```

```
// Allows customers to withdraw cash from their account
class Withdrawal : public Transaction{
public:
	void cash_withdrawl();		// Performs cash withdrawal
	void withdrawal_limit();	// Checks withdrawal limit for the account
	void create_wtransaction_ID();	// Creates a unique withdrawal transaction ID
	void loan_amount();		// Checks or deducts loan amounts if needed
	void display_withdrawal_alert();	// Displays alerts related to withdrawal
private:
	double withdrawal_limit;		// Maximum amount allowed for withdrawal
};
```

```
// Allows customers to deposit cash or checks
class Deposit : public Transaction{
public:
        void select_account();              // Selects account for deposit
        void check_cash_amount();           // Validates the cash amount deposited
        void check_scan_status();         // Checks if deposited check/cash scanned correctly
        void create_transaction_ID();       // Creates transaction ID for deposit
        void display_deposit_alert();       // Displays deposit-related alerts
        void scan_cash();                   // Scans cash for validation
private:
        string select_account;        // Account type: checking, savings, money_market
        bool scan_status;             // True if cash/check scanned successfully
        double dep_num;               // Deposit amount
};
```

```
// Handles transfer of funds between accounts
class Transfer : public Transaction{
public:
        void determine_the_customer();      // Determines customer initiating transfer
        void select_source_account();       // Selects account to transfer money from
        void select_destination_account();          // Selects account to transfer money to
        void validate_account_pair();       // Validates the source and destination accounts
        void input_amount();                // Lets customer input the transfer amount
        void validate_amount();             // Validates that the amount is permissible
        void confirm_transfer();            // Confirms the transfer before execution
        void record_transfer();             // Records the transaction in the system
        void show_confirmation();           // Shows confirmation message to the customer
private:
        double input_transfer_num;          // Amount to be transferred
};
```