**Requirements Elicitation  
  
Banking System Interview Summary**

**Interviewer:** Bame Kaosi Mosimakoko  
**Interviewee**: Kentsenao Victor Baseki  
**Date:** 18 September 2025  
**Time:** 1030hrs

**1. Question:** Please describe the authentication process for customers accessing the banking system. What functionalities are available to them after a successful login?  
**Response:** Customers gain access by entering a username and password, which the system verifies. Once authenticated, they can perform several operations, including depositing and withdrawing funds, checking their account balance, and reviewing their transaction history.

**2. Question:** Could you explain the step-by-step procedure for a customer to deposit funds into their account?  
**Response:** The process is customer-initiated: the customer specifies the amount to be deposited. The system then processes this request by updating the respective account's balance and provides an on-screen confirmation to the customer upon completion.

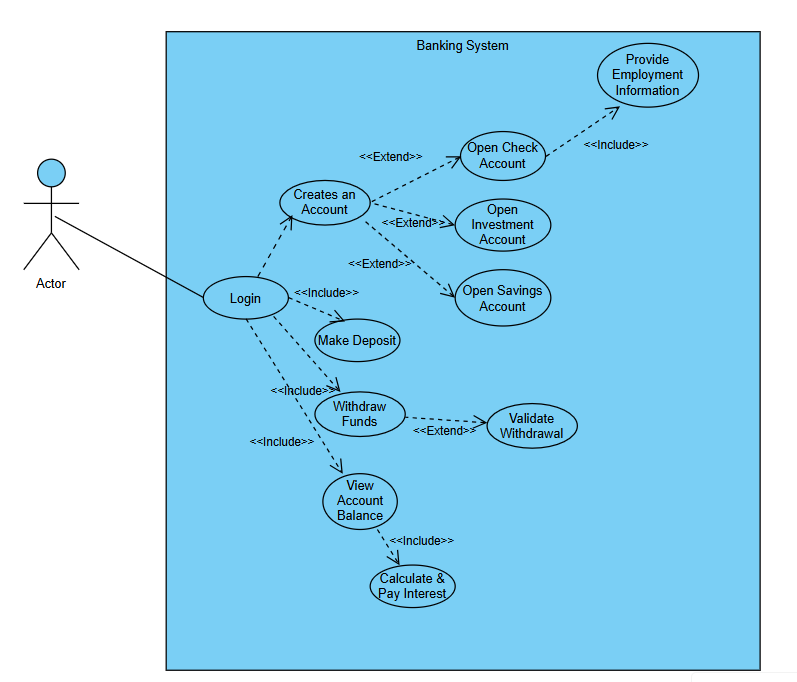
**3. Question:** What is the defined process for a customer to withdraw money from their account?  
**Response:** A withdrawal begins with a customer requesting a specific amount. The system first validates the request by checking if the account has sufficient funds. If the check passes, the system deducts the amount and displays a confirmation message for the customer.

**4. Question:** Is there a functionality for customers to open new accounts? What are the different account types offered by the bank?  
**Response:** Yes, customers can open new accounts. This process is facilitated through the system, often with assistance from bank staff for necessary verification. The bank offers three primary account types: Savings, Investment, and Cheque accounts.

**5. Question:** What is the bank's policy regarding a single customer holding multiple accounts?  
**Response:** Yes, a single customer is permitted to hold multiple accounts. The policy allows for a customer to have different types of accounts (e.g., one Savings, one Investment, and one Cheque). However, a customer cannot have multiple accounts of the same type.

**Key Observations & Notes from the Interview:**

* The primary point of interaction for customers is a graphical user interface (GUI).
* The system architecture utilizes controller components to manage application logic and mediate between the user interface and the core data models.
* The interview was instrumental in clarifying specific operational edge cases and defining user roles and permissions within the system.

1. **Structural UML Modelling  
     
   System Use Case Diagram  
     
   Actor:** The **Customer** interacts with all the use cases.

**Core Use Cases:** The customer can perform four main actions:

Open Account

Make Deposit

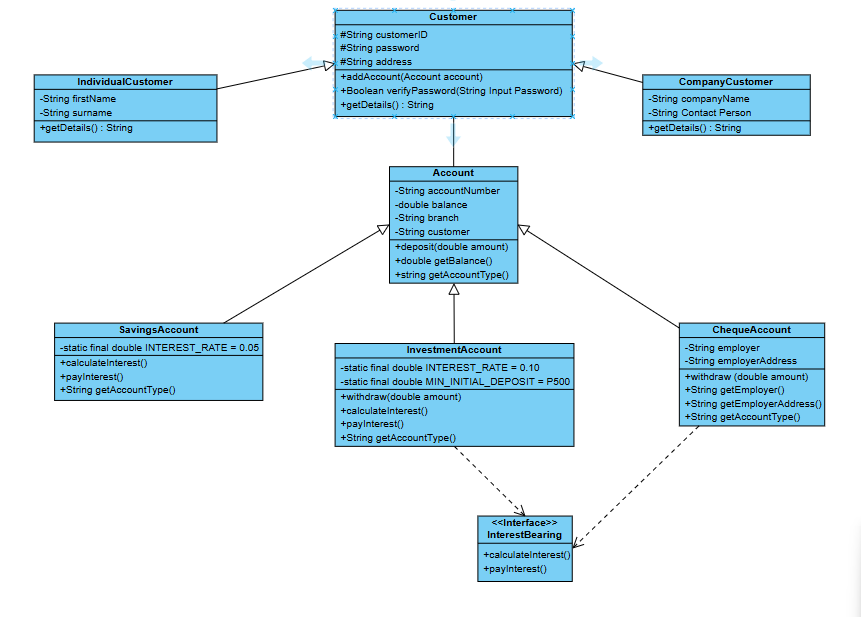
Withdraw Funds

View Account Balance

**«extend» Relationships:** The general Open Account use case is extended by three specific types: Open Savings Account, Open Investment Account, and Open Cheque Account. This means the customer chooses one of these specific paths when opening an account.

**«include» Relationships:** Open Cheque Account requires the customer to perform Provide Employment Information. This is a mandatory step, shown by the «include» relationship.

**Business Rule («extend» for Validation):** The Validate Withdrawal Allowed use case extends Withdraw Funds. This represents the conditional business rule that blocks withdrawals from Savings Accounts. It is an exception to the normal flow.

**Class diagram**  
  
  
**1. The Core Classes & Their Purpose:**

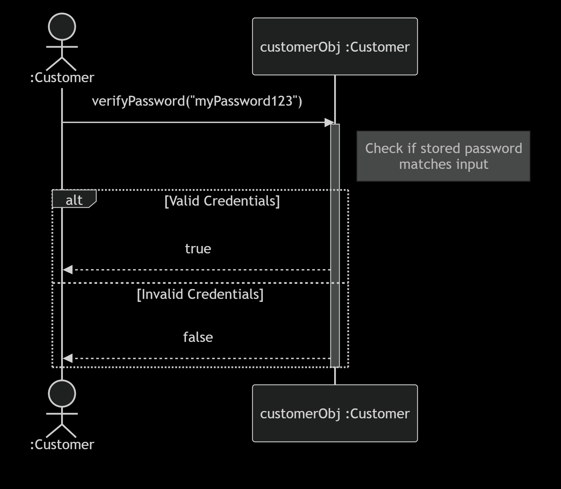
* Customer**:** An abstract class representing a user of the bank. It holds core data like customerId, password, and address.
* IndividualCustomer**&**CorporateCustomer**:** Concrete subclasses that inherit from Customer. They hold specific data for people (firstName, surname) and organizations (companyName, contactPerson).
* Account**:** An abstract class representing a bank account. It holds data like accountNumber, balance, and branch, and defines core methods like deposit() and getBalance().
* SavingsAccount**,**InvestmentAccount**,**ChequeAccount**:** Concrete subclasses that inherit from Account. Each has specific rules:
  + SavingsAccount: Has an interest rate and cannot withdraw.
  + InvestmentAccount: Has a higher interest rate, a minimum deposit, and allows withdrawals.
  + ChequeAccount: Holds employer info and allows withdrawals.
* InterestBearing**:** An interface. It's a contract stating that any class implementing it must have calculateInterest() and payInterest() methods.

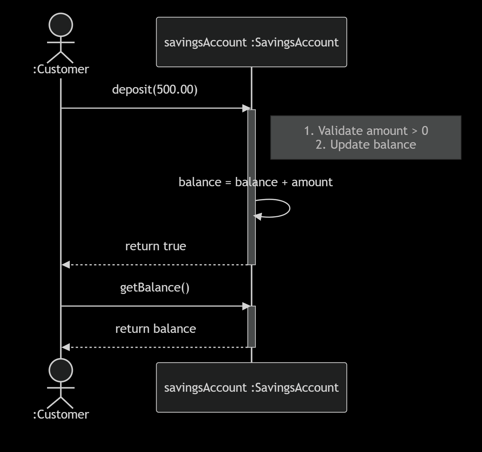
**2. The Relationships (The Glue):**

* **Inheritance (Generalization)**◁---**:**
  + IndividualCustomer**→**Customer, CorporateCustomer**→**Customer
  + SavingsAccount**→**Account, InvestmentAccount**→**Account, ChequeAccount**→**Account
  + **Meaning:** This shows an **"is-a"** relationship. e.g., A SavingsAccount *is a* type of Account.
* **Interface Realization**◁..**:**
  + SavingsAccount**→**InterestBearing, InvestmentAccount**→**InterestBearing
  + **Meaning:** This shows a **"can-do"** relationship. These accounts agree to fulfill the contract of calculating and paying interest.
* **Association**---**:**
  + Customer**"1" -- "\*"**Account
  + **Meaning:** This shows a **"has-a"** relationship. One Customer *has* (owns) many Account objects. This directly models the business rule: "a customer can have one or multiple accounts."

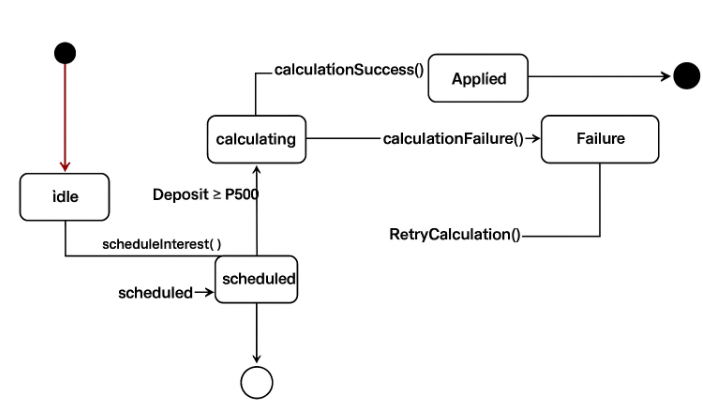
**3. OOP Principles Demonstrated:**

* **Abstraction:** The Account and Customer classes are abstract, providing a general blueprint without complete implementation.
* **Inheritance:** The specialized account and customer types reuse code and structure from their parent classes.
* **Polymorphism:** The getAccountType() or calculateInterest() methods will behave differently depending on whether the object is a SavingsAccount or InvestmentAccount.
* **Encapsulation:** All attributes are private (-) or protected (#), meaning they can only be accessed through public methods (+) like deposit() or getBalance(). This protects the data.
* **Interface:** The InterestBearing interface defines a clear contract without implementation details.

1. **Behavioural UML Modelling  
     
   Sequence Diagrams  
     
   LogIn  
     
     
     
   Actor Initiates: The :Customer (actor) provides their password input and triggers the process by calling the verifyPassword() method on a Customer object (customerObj).**
2. **Object Activation: The customerObj is now active (shown by the activation bar).**
3. **Logic Execution: The object performs the internal logic of retrieving the stored password and comparing it to the input.**
4. **Alternative Flows (alt): This box encapsulates the two possible outcomes:**
5. **Valid Credentials: The method returns true to the actor.**
6. **Invalid Credentials: The method returns false to the actor.**
7. **Object Deactivation: The customerObj finishes its task and is deactivated.**

**Deposit  
  
  
  
Actor Initiates:** The Customer (actor) provides an amount and triggers the process by calling the deposit() method on a SavingsAccount object.

1. **Object Activation & Validation:** The savingsAccount object activates and performs internal steps: it first validates the amount is positive.
2. **State Change:** The object updates its own balance attribute. This is a key concept—objects are responsible for changing their own data.
3. **Confirmation:** The object returns true to confirm success to the actor and deactivates.
4. **Querying State:** The actor then queries the new state of the object by calling the getBalance() method.
5. **Return Result:** The object activates again, returns the current balance, and deactivates.

**State Diagram  
  
  
  
States:**

1. Idle**:** This is the initial, inactive state. The process is waiting to be started. It has done no work yet.
2. Calculating**:** The active state where the interest calculation is being processed. The system is performing the necessary computations.
3. Applied**:** The final success state. The calculation has finished successfully and the interest has been added to the account(s). The process can now end and return to its initial [\*] state.
4. Failure**:** The error state. Something went wrong during the calculation process (e.g., a system error, a network issue).

**Transitions (Events):**

* scheduleInterest()**:** This event triggers the process, moving it from the Idle state to the Calculating state. (e.g., a monthly scheduler kicks off the process).
* calculationSuccess()**:** This event signifies the calculation finished without errors. It moves the process from the Calculating state to the Applied state.
* calculationFailure()**:** This event signifies an error occurred. It moves the process from the Calculating state to the Failure state.
* RetryCalculation()**:** This is a recovery event. While in the Failure state, this event will trigger another attempt, moving the process back to the Calculating state.
* Deposit ≥ 500**:** This is another recovery event. If a deposit of at least 500 is made, it can resolve the failure condition and move the process all the way back to the Idle state to start over. (This is an interesting business rule that suggests a minimum balance might be needed for interest calculation).

**Overall Flow:**

The process starts Idle, gets triggered to start Calculating, and then either succeeds (Applied) and ends, or fails (Failure). From failure, it can either retry or be reset by a qualifying deposit.