**1. Requirements Elicitation**

**1.1. Mock Interview Transcript**

**Me (Atang Mutema):** What types of accounts should the system support?  
**Client (Sir Themba Moeng):** Savings, Investment, and Cheque accounts. Each has rules: Savings only allows deposits, Investment requires BWP 500 minimum and allows withdrawals, and Cheque accounts need proof of employment.

**Me:** Can a customer have more than one account?  
**Client:** Yes, customers may hold multiple accounts of different types.

**Me:** How should interest be handled?  
**Client:** Savings accounts pay 0.05% monthly, Investment accounts pay 5% monthly, and Cheque accounts do not earn interest.

**Me:** How should customer registration work?  
**Client:** Registration should be handled by bank staff with details like full name, ID, date of birth, address, and contact information.

**Me:** How should users access the system?  
**Client:** Customers log in with a username and password, while staff and administrators should have stronger authentication.

**Me:** What about transaction history and statements?  
**Client:** Customers should see the last 12 months of transactions, staff should see complete histories, and account statements should be downloadable in PDF.

**Me:** Should the system recognize different user roles?  
**Client:** Yes, customers, staff, and administrators should have different access permissions.

**1.2 Functional Requirements**

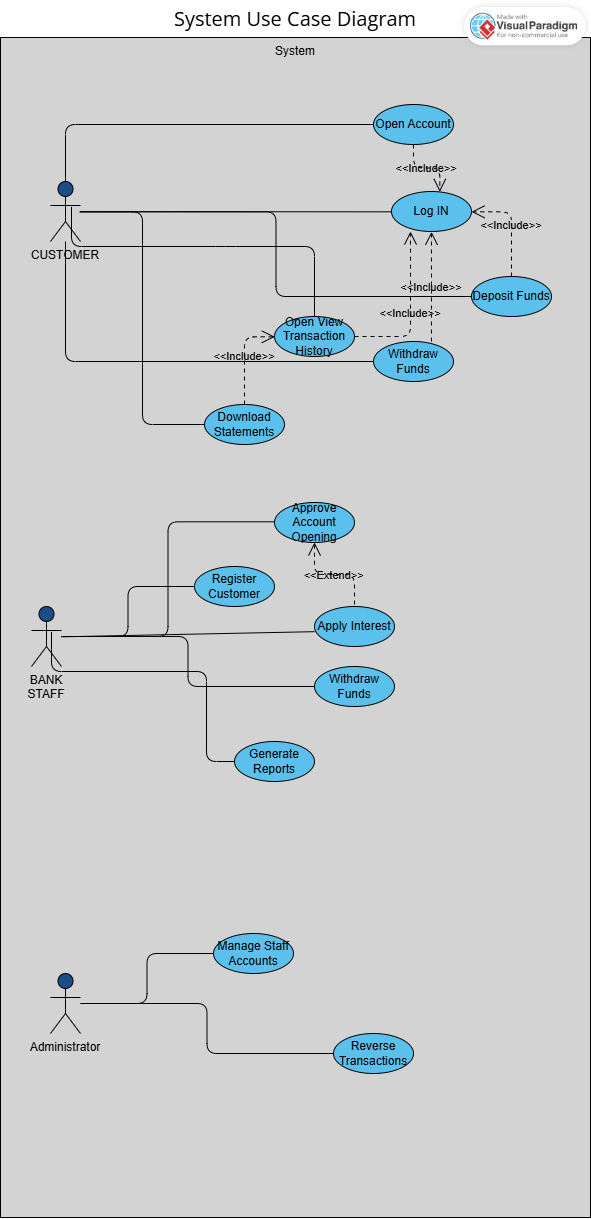
1. The system shall allow administrators to register new customers with their personal details, including first name, surname, address, password, and phone number.
2. The system shall allow each customer to open and hold multiple account types — **Savings, Investment, and Cheque** — with appropriate rules and validations for each type.
3. The system shall allow deposits into all customer accounts.
4. The system shall allow withdrawals from **Cheque** and **Investment** accounts.
5. The system shall allow customers to **transfer money** between their own accounts or to other customers’ accounts within the bank.
6. The system shall display customer account information, including **account balances**, customer details, and **transaction history**.
7. The system shall calculate and apply **monthly interest** to Savings and Investment accounts according to their defined rates.
8. The system shall allow administrators to view **system summaries**, including the total number of customers and accounts registered.
9. The system shall allow customers and administrators to **log out securely** and navigate between pages such as **Dashboard, Profile, Transactions, Withdraw, Transfer, and Admin**.
10. The system shall allow withdrawals to be sent directly to the customer’s registered mobile number.

**1.3 Non-Functional Requirements**

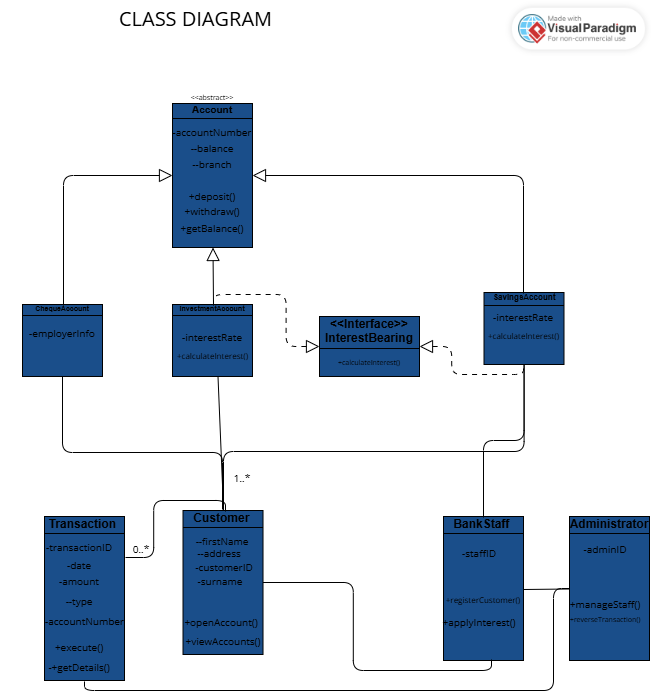
1. **Security**
   * Customer login requires a valid **username and password**.
   * Administrators have restricted access to **customer management and registration functions**.
2. **Performance**
   * Deposits, withdrawals, transfers, and interest calculations shall process **efficiently** within seconds.
   * Loading customer lists, accounts, and transaction history shall be **responsive**.
3. **Usability**
   * The system shall provide a **clean, user-friendly interface** (using JavaFX) for easy navigation.
   * All forms, buttons, and fields shall have **consistent styling** with readable fonts and intuitive layouts.
4. **Data Integrity**
   * All transactions and account updates shall reflect **immediately** in the system to maintain accuracy and consistency.
5. **Maintainability**
   * The system follows a **modular design**, with each FXML page having a separate controller for easier debugging and future enhancements.
6. **Scalability**
   * The system design allows future extension for features such as **online statements, loan processing, or interbank transfers**.

**2. Structural UML Modelling**

**2.1. System Use Case Diagram**

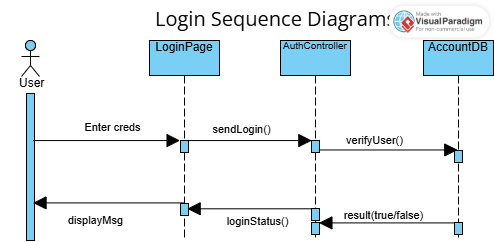


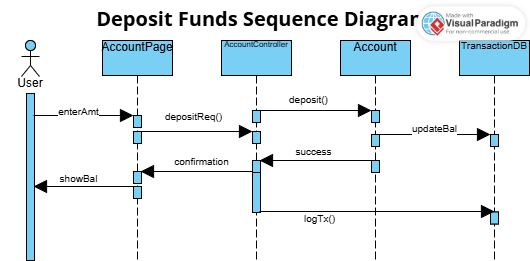
**2.2. Class diagram**



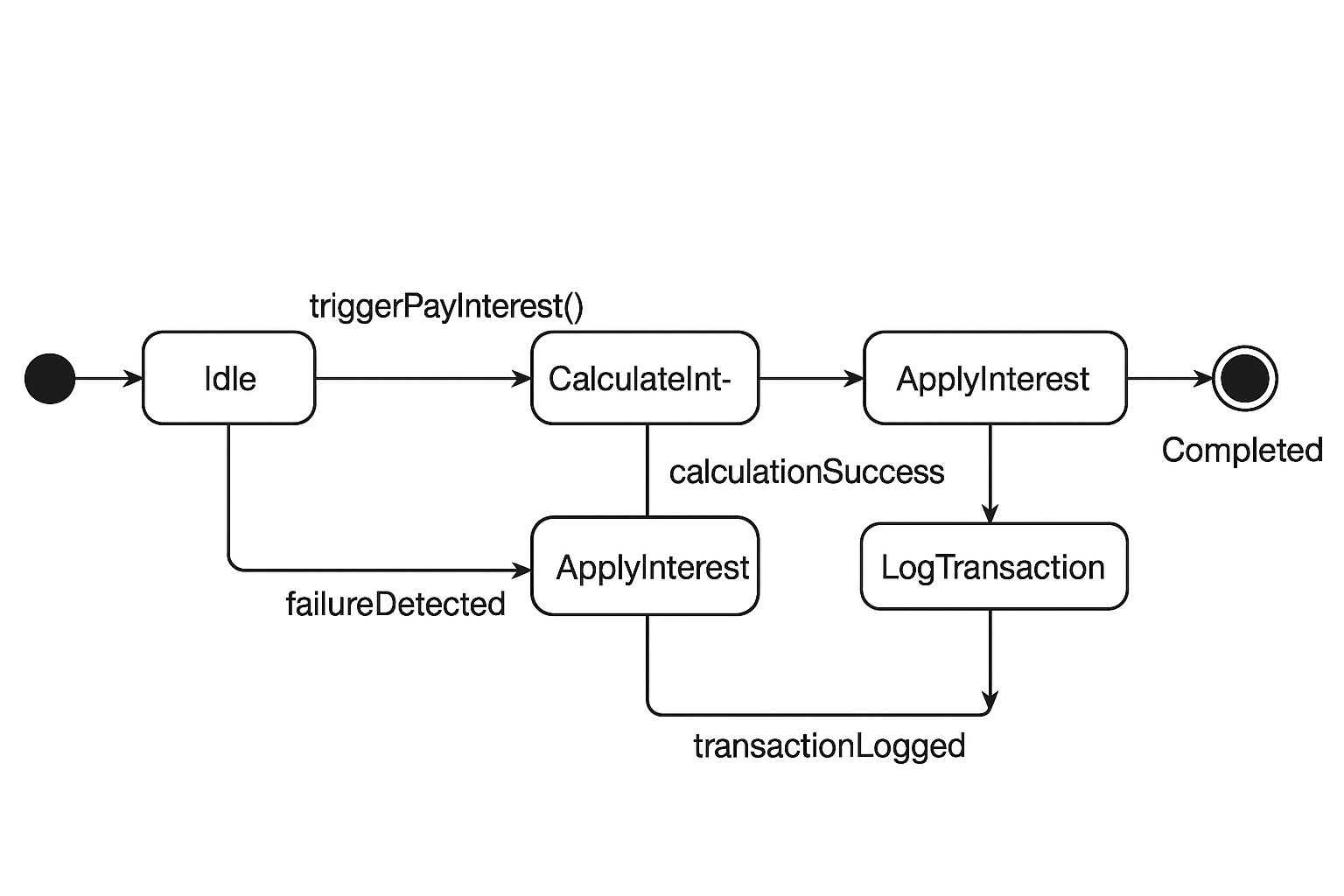
**3. Behavioural UML Modelling)**

**3.1. Sequence Diagrams**





**3.2. State Diagram**



**4. Implementation of Core Model [15 Marks]**

The objective of this stage was to implement the core business logic of the banking system, based on the designed domain model and class diagrams. This phase focused on translating the conceptual model into working Java classes that represent the bank’s customers, accounts, and associated operations.

**Key Implementations:**

* **Account (Abstract Class):**  
  This serves as the base for all account types and contains common attributes such as accountNumber, branch, and balance. Shared methods like deposit(double amount), withdraw(double amount), and getBalance() were implemented to support basic account operations.
* **SavingsAccount, ChequeAccount, and InvestmentAccount:**  
  Each subclass extends Account and implements specific behaviors. For instance, InvestmentAccount enforces a minimum initial deposit. While Cheque and Savings accounts currently have no additional restrictions, this aligns with the assignment requirements.
* **Customer Class:**  
  Represents each bank customer and contains personal details such as firstName, surname, address, password, and phoneNumber. It also maintains a list of Account objects, allowing each customer to hold multiple account types. Methods to add and manage accounts were implemented to support transactions and account retrieval.
* **Business Logic Highlights:**
  + Encapsulation is ensured with private attributes and public getter/setter methods.
  + Core operations such as deposits, withdrawals, and monthly interest calculations for Savings and Investment accounts are implemented.

**Summary:**  
The core model provides a functional foundation for the system. Customers can hold multiple accounts, and the base account operations are fully implemented, making the system ready for GUI integration and user interaction.

**5. GUI Design & Implementation [10 Marks]**

The goal of this stage was to design a user-friendly interface to allow interaction with the banking system while maintaining separation from the core business logic. JavaFX was used to create modular screens for various functionalities.

**Implemented Screens and Features:**

* **Login:** Allows administrators and customers to enter credentials and access the system.
* **Dashboard:** Provides navigation buttons for different modules, including Transactions, Transfer, Withdraw, Profile, and Admin pages.
* **Transactions:** Displays the customer’s transaction history in a clear and organized format.
* **Transfer:** Allows customers to transfer money between their own accounts or to other customers’ accounts within the bank. It validates the amount and target account.
* **Withdraw:** Enables withdrawals from Cheque and Investment accounts directly to the customer’s registered mobile number.
* **Profile:** Displays the customer’s personal information and all associated accounts.
* **Admin:** Allows administrators to register new customers and add Savings, Cheque, and Investment accounts. Also displays total customers and total accounts.

**Design Highlights:**

* Each screen is implemented as an FXML file with a dedicated controller, ensuring modularity and separation of concerns.
* Consistent styling is applied through a shared CSS file (Default.css) for readability and intuitive navigation.
* Logical layouts (AnchorPane, VBox, HBox) and images/icons are used to enhance usability.
* Navigation between screens is smooth, and user input validation is in place for most critical operations.

**Summary:**  
The GUI design allows both customers and administrators to interact with the system efficiently. All implemented screens are functional and align with the assignment’s expectations for usability and modularity.

**6. Controller Implementation**

Controllers serve as the bridge between the GUI and the core model, handling user actions, validating inputs, and updating the model as necessary. Each FXML screen has its dedicated controller to maintain separation of concerns.

**Implemented Controllers and Responsibilities:**

* **TransferController:**  
  Manages money transfers. Users can transfer funds between their own accounts (using a dropdown) or to other customers’ accounts (typed manually). The controller ensures input validation and updates account balances accordingly.
* **WithdrawController:**  
  Handles withdrawals from Cheque and Investment accounts. Users input an amount, and the controller processes the withdrawal to the customer’s registered mobile number. Input validation and balance checks are enforced.
* **TransactionController:**  
  Retrieves and displays a customer’s transaction history for all accounts. Transactions are presented in a structured format, providing clarity on past activities.
* **ProfileController:**  
  Displays a customer’s personal information and all associated accounts. It allows customers to view their details and account summaries seamlessly.
* **AdminController:**  
  Allows administrators to register new customers and add multiple account types (Savings, Cheque, Investment) for them.  
  It also calculates and applies monthly interest for Savings and Investment accounts and displays system summaries such as total customers and accounts. Input validation ensures proper data entry.

**Highlights of Controller Design:**

* Each controller manages only its specific page and responsibilities, including navigation between screens, input validation, and interaction with the core model.
* Features like profile viewing, transactions, withdrawals, and transfers are fully functional and validated.
* Monthly interest calculation for Savings and Investment accounts is implemented and can be triggered via the admin interface.
* The design supports modularity, maintainability, and scalability for future enhancements such as statements download or interbank transfers.

**Summary:**  
The controller implementation ensures that the system’s business logic and user interface work together seamlessly. All critical functionalities expected by the OOAD assignment — including customer registration, account creation, deposits, withdrawals, transfers, profile viewing, and system summaries — are implemented and operational.