Banking System - UML Design

Software Modelling with Use Case, Class, and Sequence Diagrams

Date: February 2026

Chapter 1

System Overview: The Digital Core of Modern Banking

The Banking System is an intricate digital infrastructure designed to efficiently manage diverse financial operations. It acts as the backbone, enabling seamless transactions and maintaining financial records with precision and security.

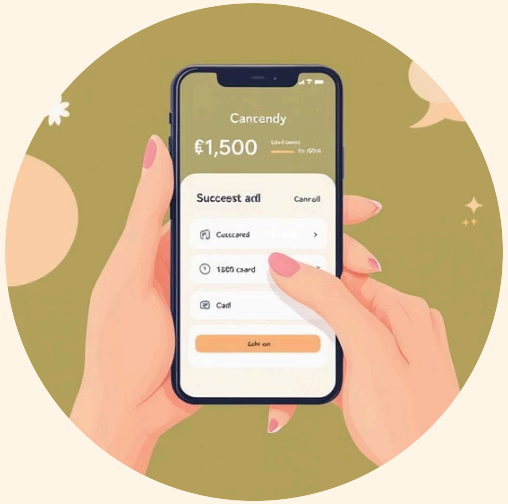
Key functions include:

- Account Management (opening, closing, updating)
- Deposits and Withdrawals (cash and cheque)
- Fund Transfers (between accounts)
- Loan Processing (applications, approvals, repayments)

Chapter 2

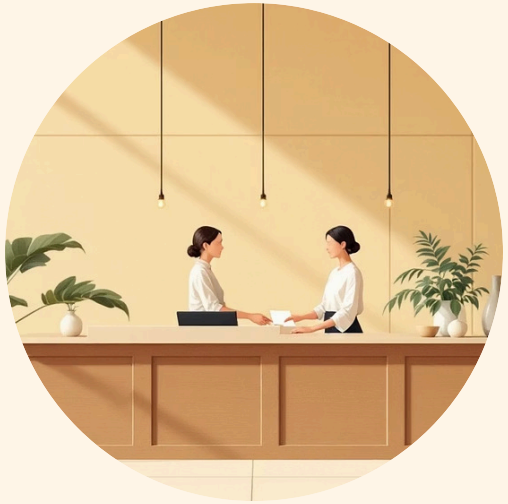
Key Actors: Interacting with the Banking System

Understanding the various individuals and systems that interact with the banking system is crucial for a comprehensive UML design. Each actor has distinct roles and access levels, driving specific use cases within the system.



Customer

Individuals holding accounts and performing transactions via various channels.



Bank Teller

Bank employees who process customer transactions and verify identities.

Administrator

Personnel responsible for system management, configuration, and report generation.

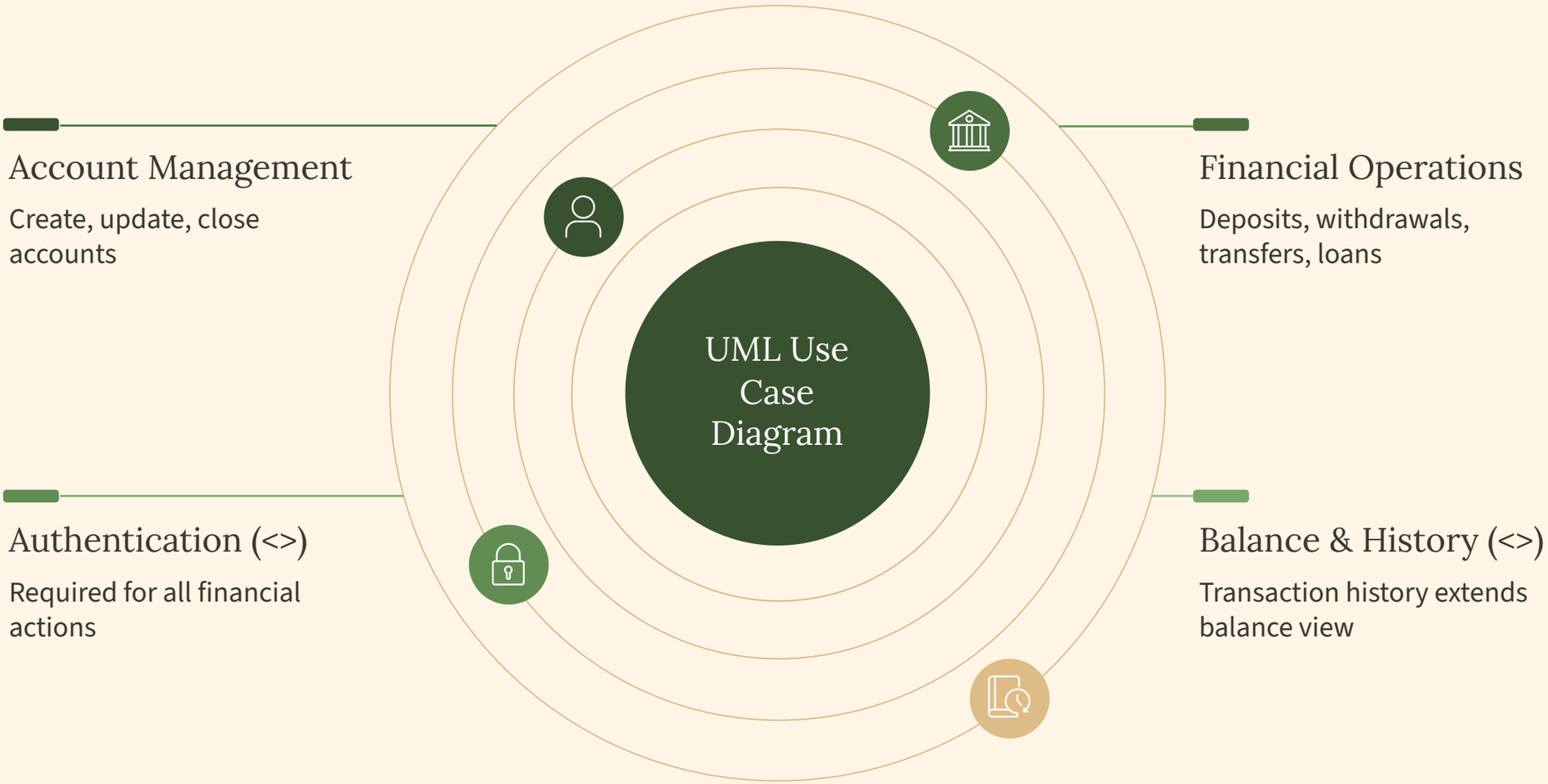
ATM Machine

An automated self-service terminal for basic banking operations.

Chapter 3

Use Case Diagram: Mapping System Functionality

This diagram provides a high-level view of how actors interact with the banking system, detailing the functions they perform. It clearly illustrates the scope and primary functionalities from a user's perspective.



The diagram captures the breadth of interactions, from a customer's basic transactions to an administrator's complex system management tasks. It serves as a foundational blueprint for understanding user requirements.

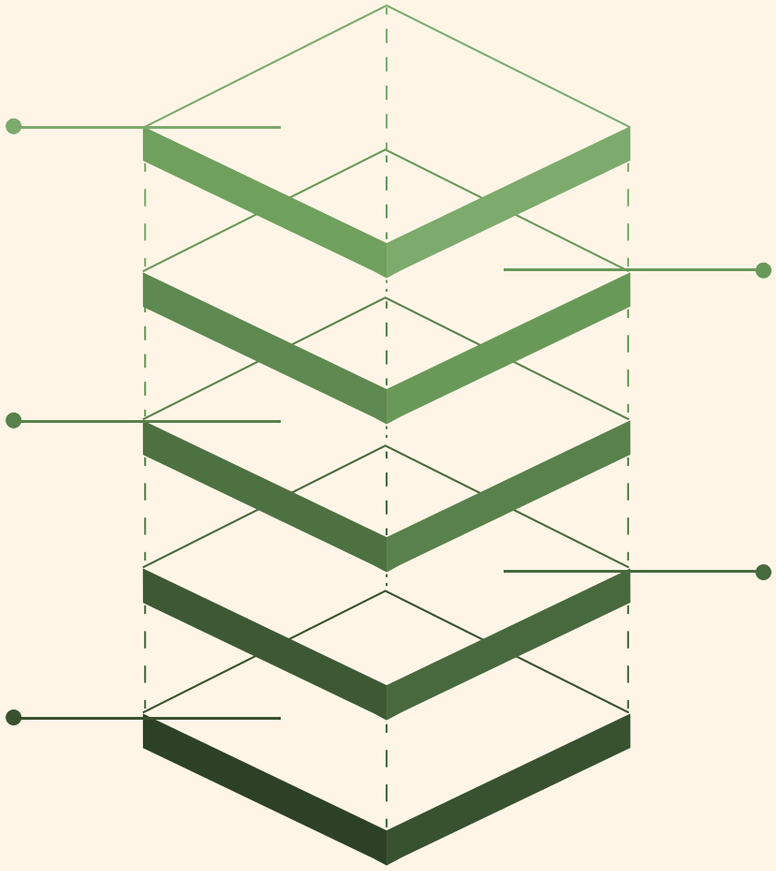
Deeper Dive into Use Case Relationships

- **Security First:** Every financial operation, such as depositing or withdrawing funds, explicitly **includes** an authentication step, ensuring secure access and transaction integrity.
- **ATM Limitations:** The ATM machine supports a specific subset of operations: deposit, withdraw, and balance inquiry, reflecting its self-service nature.
- **Teller Capabilities:** Bank Tellers have the authority to act on behalf of customers, performing various transactions and providing assistance.
- **Administrative Control:** Administrators engage in distinct management-focused use cases, separate from direct customer interactions, such as managing user accounts and generating system reports.
- **Enhanced Features:** The Transaction History use case **extends** the Balance View, offering a more detailed look into past financial activities.

Chapter 4

Class Diagram: Structuring the Banking System

The Class Diagram outlines the static structure of the banking system, identifying the classes, their attributes, methods, and relationships. It is essential for defining the system's data model and foundational architecture.



Key Associations

Customer 1..* Accounts; Account 1..* Transactions

Bank Entity

Bank aggregates branches and services

Transactions & Loans

Transaction and Loan classes shown

Account Hierarchy

Account, SavingsAccount, CheckingAccount

User Hierarchy

User, Customer, BankTeller, Admin

This diagram serves as a blueprint for developers, guiding the implementation of the system's core components and ensuring a robust, scalable design.

Class Diagram: Deconstructing Relationships

User Hierarchy

The **User** class serves as a base, with specialised subclasses like **Customer**, **BankTeller**, and **Admin** inheriting common properties and behaviours. This ensures code reusability and a clear organisational structure.

Account Specialisation

Similarly, the **Account** class is extended by **SavingsAccount** and **CheckingAccount**, each with unique attributes and rules specific to their account types.

Customer-Account Linkage

A crucial **Association** exists where a **Customer** can own multiple **Accounts** (a one-to-many relationship), meticulously tracking financial assets.

Account-Transaction Recording

Each **Account** accurately records numerous **Transactions** (also a one-to-many relationship), forming a comprehensive audit trail of all financial activities.

These relationships form the backbone of the banking system's data integrity and operational logic, directly mapping to how data might be stored in database tables.

Chapter 5

Sequence Diagram: Visualising Fund Transfer

The Sequence Diagram illustrates the dynamic behaviour of the system by modelling the interactions between objects in a time-ordered manner. Here, we focus on the complex, yet common, "Fund Transfer" process.

This diagram provides a step-by-step visualisation, revealing how different system components collaborate to complete a single, critical operation, from initial request to final confirmation.

Dissecting the Fund Transfer Flow

The fund transfer process is a multi-stage operation, carefully orchestrated across several system participants to ensure security, accuracy, and timely completion.

1

1. Authentication Phase

The customer initiates the transfer by logging into the mobile app. Their credentials are sent to the **Authentication Service** for validation, ensuring only authorised users can proceed.

2

2. Validation Phase

The **Account Service** verifies the sender's balance and checks the validity of the recipient's account. This prevents transactions with insufficient funds or incorrect details.

3

3. Execution Phase

Upon successful validation, the **Transaction Service** debits the sender's account and credits the recipient's account. These operations are typically atomic to maintain data consistency.

4

4. Confirmation & Notification

Finally, the mobile app displays a transaction receipt, and the **Notification Service** dispatches an email or SMS to both sender and recipient, confirming the successful transfer.

5

5. Error Handling

In cases like **insufficient funds**, an alternative flow is triggered. The customer is notified, and the transaction is gracefully aborted, ensuring system integrity.

Chapter 6

Conclusion: The Power of UML in Software Design

UML offers a visual, structured approach to software design, crucial for clarity and collaboration.

By employing various diagram types, we gain multi-faceted insights into complex systems.

- **Use Case Diagrams:** Define functional requirements from an actor's perspective, answering "what" the system does.
- **Class Diagrams:** Illustrate the system's static structure, answering "how" it's organised internally.
- **Sequence Diagrams:** Model dynamic behaviour and object interactions over time, detailing event sequences.
- **Preventative Measures:** Modelling before coding significantly reduces errors and saves valuable development time.

Mastering UML enables robust, maintainable software solutions.