Lecture 4

Dr. Umair Rehman

Agenda

- Requirements Gathering
- Functional vs Non-Functional Requirements
- Techniques for Requirements Gathering
- UML
  - Use Case Modelling
  - UML Class Diagrams
  - Sequence Diagrams

Introduction to Requirements Gathering

- Critical step in the software development lifecycle
- Foundation for a successful project
- Object-Oriented Design and Analysis (OOD&A) framework
  - Translate user needs into functional and non-functional requirements
  - Identify key system objects, relationships, and behaviors

Requirements Analysis

- Natural Language Requirements

- Modelling Requirements
  - Visualizations: UML Diagram
  - Formulas
  - Code

- Artifacts
  - Goals
  - Stakeholders
  - Constraints

Functional Requirements

- Focus on what the system should do

- Examples:
  - “User can create a new account”
  - “System should send an email after registration”

- Requirements map directly to objects and their behaviors (methods)

Functional Requirements (Example)

- Online bookstore, the functional requirement "User can add books to the shopping cart" maps to:

- Objects
  - `Book,`
  - `ShoppingCart,`

- methods
  - `addToCart().`

Non-Functional Requirements

- Define the quality of the system, aspects like
  - Performance
  - Security
  - Reliability

- Examples:
  - “System must support 500 transactions per minute”
  - “Response time should not exceed 2 seconds”

Non-Functional Requirements Example

- “System should handle 10,000 concurrent users” may affect the design of the underlying
  - Infrastructure
  - Class distribution
  - Architecture choices like multi-threading.

Techniques for Gathering Requirements

- Interviews:
  - One-on-one or group sessions
  - Stakeholders provide detailed insights

- Extract
  - Nouns (potential objects)
  - Verbs (potential methods)

- A stakeholder might say, "The customer places an order."
  - This points to objects like `Customer`, `Order`, and behaviors like `placeOrder()`.

Techniques for Gathering Requirements

- Structured questionnaires
  - Gather input from a large group of stakeholders

- Identify common objects and methods

- From a survey, you might see repeated requests for a "login" feature
  - Pointing to objects like `User` and methods like `authenticate()`

Techniques for Gathering Requirements

- Observation
  - Watching how users interact with existing systems

- Understand implicit requirements by observing
  - Users expect the system to behave

- Observing users placing items in a physical shopping cart
  - `ShoppingCart` object
  - interactions like `addItem()` in the software.

UML

- UML is short for Unified Modeling Language
  - A standard set of notations for use in modeling object-oriented systems

- We will encounter UML in the form of
  - Class diagrams
  - Sequence/collaboration diagrams
  - State diagrams
  - Activity diagrams, use case diagrams, etc

Understanding the OO Paradigm

- OO technique's view software systems as
  - Networks of communicating objects
- Each object is an instance of a class
- All objects of a class share similar features
  - Attributes
  - Methods
- Classes can be specialized by subclasses
- Objects communicate by sending messages

Object Communication

- In response to a message, an object may
  - Update its internal state
  - Return a value from its internal state
  - Perform a calculation based on its state and return the calculated value
  - Create a new object (or set of objects)
  - Delegate part or all of the task to some other object

Objects

- We would like objects to be
  - Highly Cohesive: Have a single purpose; make use of all features
  - Loosely Coupled: Be dependent on only a few other classes

Use Case Modeling: Tying Requirements to Objects

- Help bridge the gap between
  - What the system needs to do (requirements)
  - How the system will achieve it (design)

- Structured way to model the interactions in the system between
  - System
  - Actors
  - Use Cases
  - Relationships

Use Case Modeling: System

- System: What you are developing
  - Game
  - App
  - Website
  - Software

- Every System has a goal

Use Case Modeling: Actors

- Actor: Uses the system to achieve a goal
  - Person (usually)
  - Organization
  - External Device

- Actors are external objects

- Actors are categorical
  - Person (not Michael)
  - Bank (not RBC)

Use Case Modeling: Actors

- Primary Actor: Initiates System Use

- Secondary Actor: Reactionary

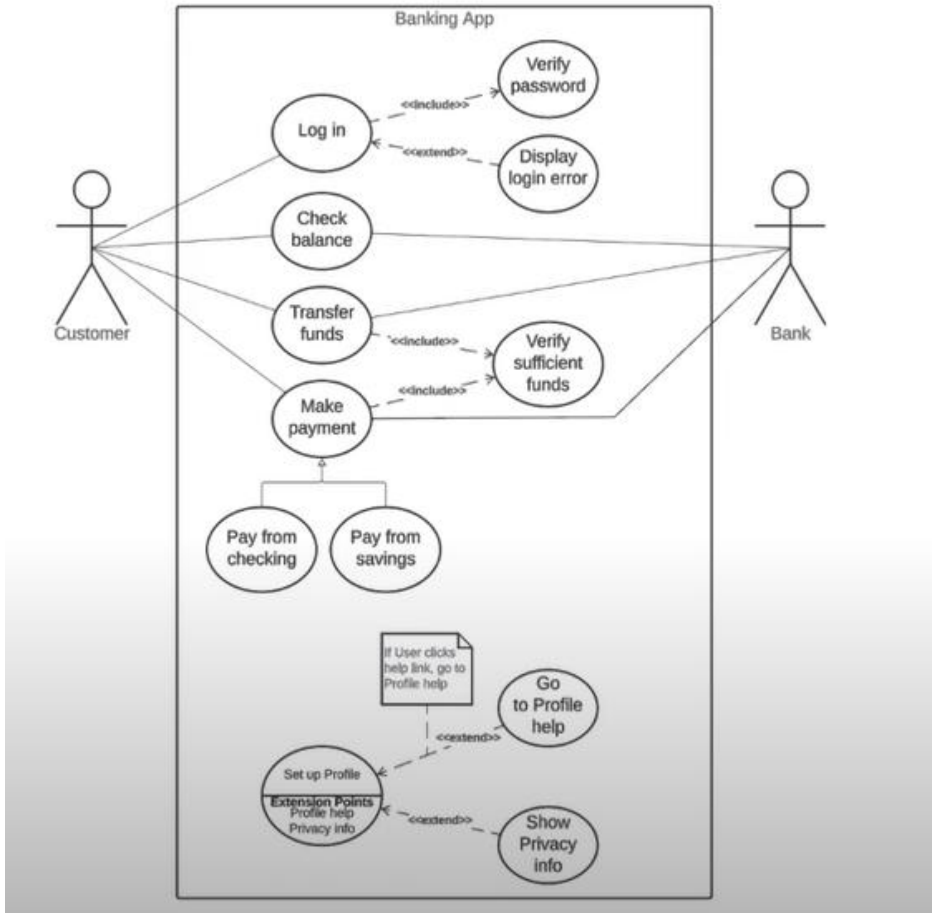
Use Case Modeling: Use Cases

- Signifies an action to accomplish a certain task within a system

Use Case Modeling: Relationships

- Association: A connection between an actor and a use case.
- Include: A use case always includes another.
- Extend: A use case optionally adds behavior to another.
- Generalization: A specialized use case inherits from another.

Use Case Diagrams:



Techniques for Identifying Use Cases:

- Analyze user interactions with the system
  - From gathered requirements

- Focus on user goals: What does the user want to achieve?
  - E.g., "Borrow Book", "Return Book"

- Look for recurring activities that involve the system

Prioritization

- Focus on the most important, high-impact use cases first
  - E.g., "Borrow Book" is more critical than "Renew Book"

- Consider risk:
  - High-risk or complex use cases may need to be addressed earlier

UML Class Diagram Notation

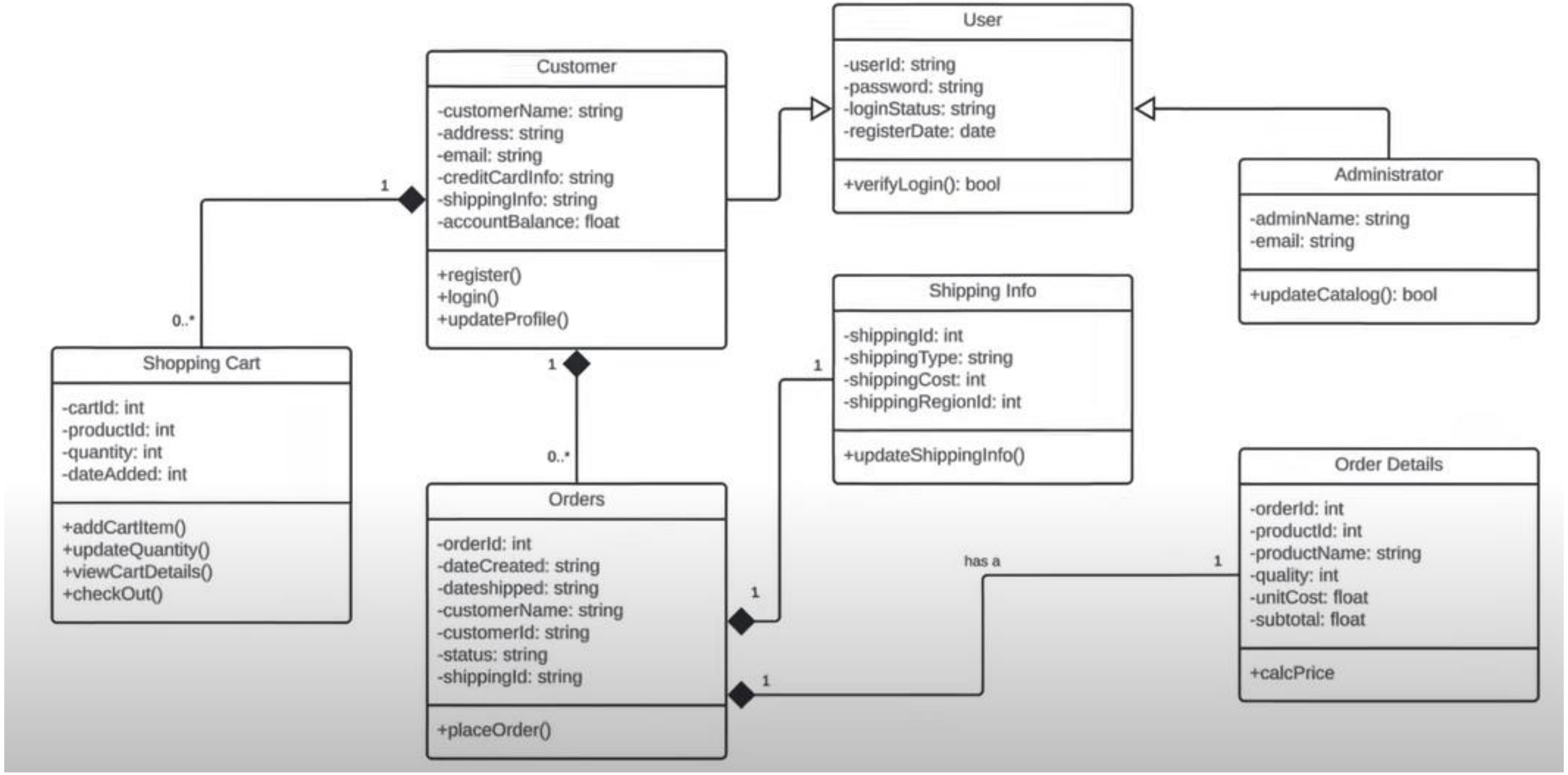
- Notation:
  - Classes are represented as boxes with three sections:
    - Name
    - Attributes
    - Methods

- Lines between classes represent relationships:
  - Solid Line for Association
  - Arrow for Inheritance
  - Diamond for Composition

Classes Relationships

- Class Relationships:
  - Association: A "has-a" relationship (e.g., `teacher` has a `student`)
  - Inheritance: A "is-a" relationship (e.g., `Admin` is a type of `User`)
  - Composition: A “strong has-a" relationship (e.g., `Car` contains an `Engine`)

UML Example



User
- -userId: string
- -password: string
- -loginStatus: string
- -registerDate: date
- +verifyLogin(): bool

Customer
- -customerName: string
- -address: string
- -email: string
- -creditCardInfo: string
- -shippingInfo: string
- -accountBalance: float
- +register()
- +login()
- +updateProfile()

Administrator
- -adminName: string
- -email: string
- +updateCatalog(): bool

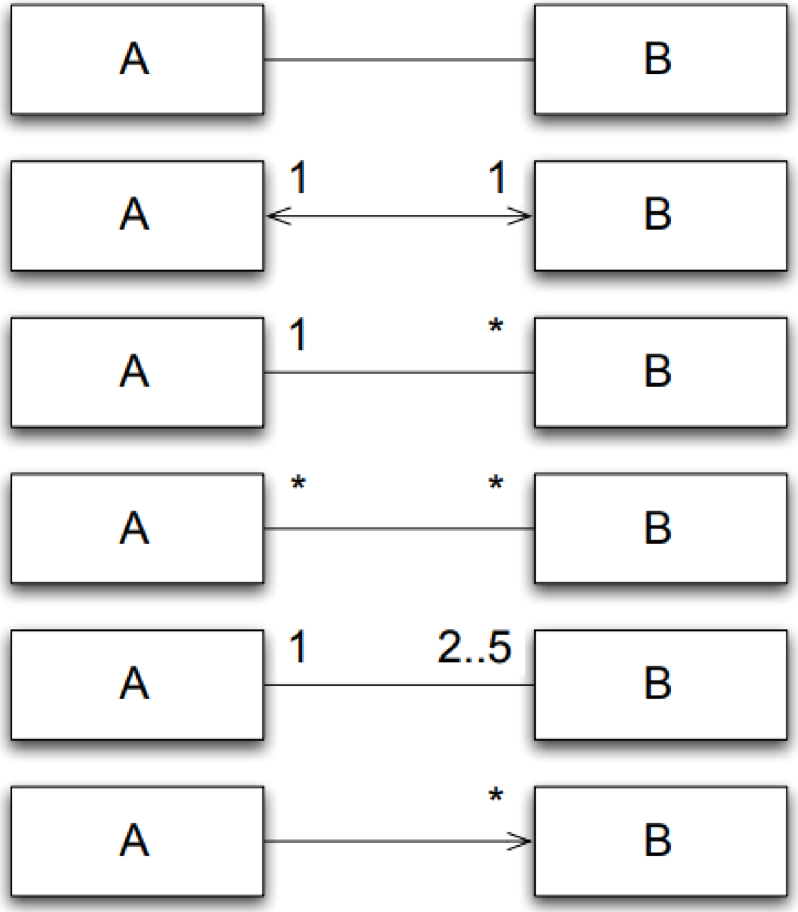
Shopping Cart
- -cartId: int
- -productId: int
- -quantity: int
- -dateAdded: int
- +addCartItem()
- +updateQuantity()
- +viewCartDetails()
- +checkOut()

Shipping Info
- -shippingId: int
- -shippingType: string
- -shippingCost: int
- -shippingRegionId: int
- +updateShippingInfo()

Orders
- -orderId: int
- -dateCreated: string
- -dateshipped: string
- -customerName: string
- -customerId: string
- -status: string
- -shippingId: string
- +placeOrder()

Order Details
- -orderId: int
- -productId: int
- -productName: string
- -quality: int
- -unitCost: float
- -subtotal: float
- +calcPrice

UML Example



One B with each A; one A with each B

Same as above

Zero or more Bs with each A; one A with each B

Zero or more Bs with each A; ditto As with each B

Two to Five Bs with each A; one A with each B

Zero or more Bs with each A; B knows nothing about A

Refining

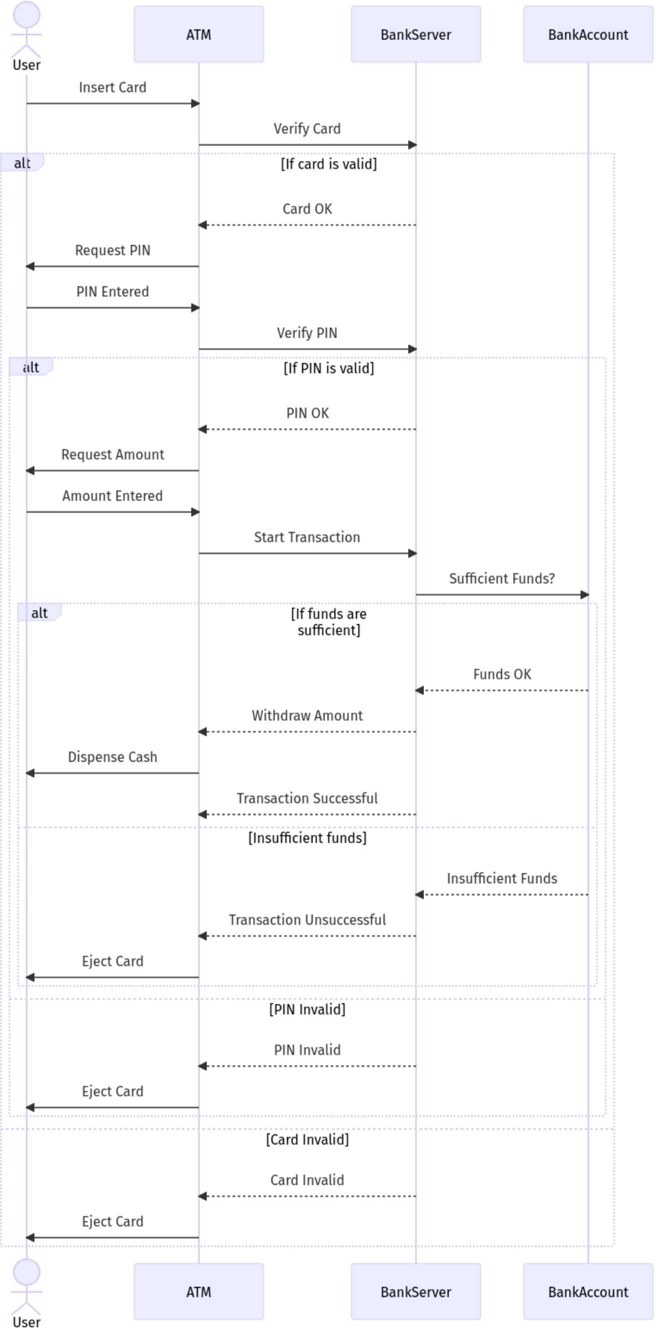
- Iterative Design
- Avoid overcomplicating diagrams
- Each class should represent a clear, distinct concept

Object Interactions with Sequence Diagrams

- Shows the flow of messages (method calls) between objects during a particular scenario or use case
- Over time Interactions
- Sequence of Events
- They complement use case diagrams by modeling the dynamic behavior of objects.

Object Interactions with Sequence Diagrams

- Actors: External entities interacting with the system
- Objects/Participants: Entities or objects involved in the interaction
- Messages: Arrows showing communication between participants
- Conditions: Control flow structures like alternatives (alt) and loops


User
ATM
BankServer
BankAccount
Insert Card
Verify Card
alt
[If card is valid]
Card OK
Request PIN
PIN Entered
Verify PIN
alt
[If PIN is valid]
PIN OK
Request Amount
Amount Entered
Start Transaction
Sufficient Funds?
alt
[If funds are sufficient]
Funds OK
Withdraw Amount
Dispense Cash
Transaction Successful
[Insufficient funds]
Insufficient Funds
Transaction Unsuccessful
Eject Card
[PIN Invalid]
PIN Invalid
Eject Card
[Card Invalid]
Card Invalid
Eject Card
User
ATM
BankServer
BankAccount

Modeling Object Interactions with Sequence Diagrams

- Clarity Over Completeness
- Iterative Design
- Avoiding Pitfalls:
  - Too many classes
  - Making diagrams too detailed for the audience

Conclusion

- Requirements Gathering
- Functional vs Non-Functional Requirements
- Techniques for Requirements Gathering
- UML
  - Use Case Modelling
  - UML Class Diagrams
  - Sequence Diagrams