OOA to OOD Transition

Instructor: Mehroze Khan

Introduction to OOA and OOD

- Object-Oriented Analysis (OOA): Focuses on **what the system must do**, identifying the key **requirements** and **objects** that are needed for the system.
  - Activities include requirements gathering, identifying entities and relationships, and defining use cases.
  - Key Outputs: Problem domain model, Use case diagrams, Class diagrams (high-level).
- Object-Oriented Design (OOD): Focuses on **how the system will achieve the requirements identified** in OOA. It transforms analysis models into detailed designs ready for implementation.
  - Activities include defining classes, methods, and their interactions.
  - Key Outputs: Detailed class diagrams, interaction diagrams (sequence diagrams), package diagrams.

Transition from OOA to OOD

- The transition from OOA to OOD involves refining the **abstract models** created during analysis and preparing them for **detailed system design and coding**. This process can be broken down into several steps

Step 1: Refining the Analysis Model

From Conceptual to Concrete:

- In OOA, entities are often abstract. In OOD, these become more concrete with details like attributes, methods, and access specifiers.
- Example: In OOA, you may have a high-level class called Account, but in OOD, you define its attributes (e.g., balance, accountType) and methods (e.g., deposit(), withdraw()).

Step 2: Relationships Refinement

Association, Aggregation, Composition:

- During OOD, relationships between objects are refined. Simple associations from OOA are turned into more specific relationships like composition or aggregation.

Step 3: Interaction and Collaboration Design

Dynamic Behavior:

- Define how objects collaborate to fulfill system requirements using interaction models like sequence diagrams or communication diagrams.
- Example: A sequence diagram for the Login use case would show how the User object interacts with AuthenticationService.

Step 4: Defining Class Interfaces and Responsibilities

Responsibility-Driven Design (RDD):

- The transition requires distributing responsibilities among classes, defining their public interfaces, and specifying how they interact with other objects.
- Design Principles: Use SOLID principles to ensure a clean, maintainable design.