Unified Modeling Language (UML)

Object Oriented Modeling and Programming

Learning Objectives

- Explain the role of UML in Object-Oriented Analysis and Design (OOAD) by understanding its significance in software modeling and system architecture.
- Model object-oriented system structure using UML structural diagrams, including Class, Object, Component, and Deployment diagrams, to represent relationships and hierarchies.
- Analyze dynamic system behavior through UML behavioral diagrams such as Use Case, Activity, and State Machine diagrams, ensuring alignment with real-world processes.
- Design and refine object interactions using UML interaction diagrams, including Sequence, Communication, and Timing diagrams, to define message flow and collaboration.
- Use UML as a design and communication tool for object-oriented software development, using industry-standard UML tools for documentation and implementation.

What is UML

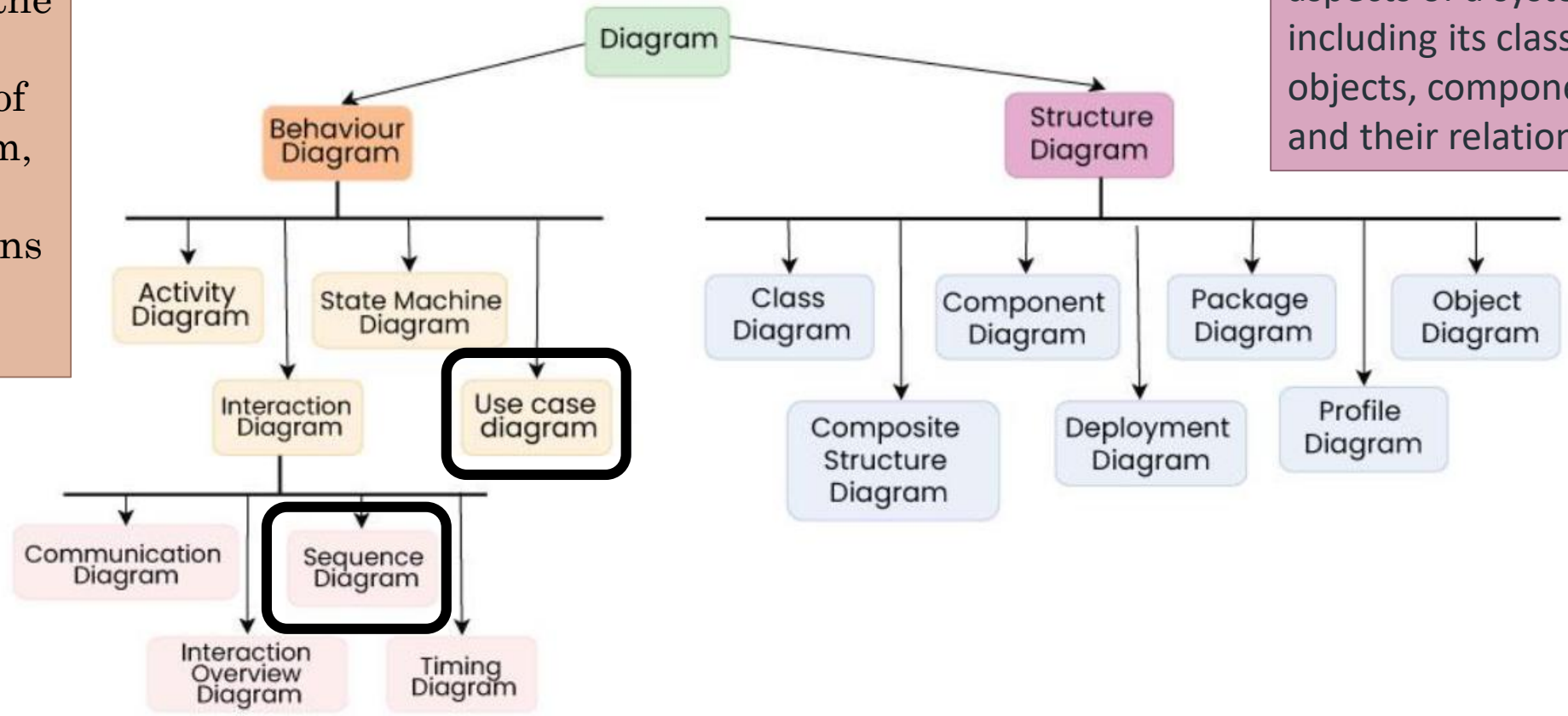
- A general-purpose modeling language (Standard language for object-oriented modeling)
- Define a standard way to visualize how a system has been designed.
- UML is not a programming language, but a visual language
- Introduced by, Booch, Rumbaugh and Jacobson (2004,2005)
- The UML has 13 diagram types
- Supports the creation of many different types of system models.

Why UML?

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non-programmers about essential requirements, functionalities, and processes.
- Time is saved: Teams can visualize processes, user interactions, and the static structure of the system.

Types of UML Diagrams

Describe the dynamic behavior of the system, including interactions between elements.



Visual representations that depict the static aspects of a system, including its classes, objects, components, and their relationships

Tools for creating UML Diagrams

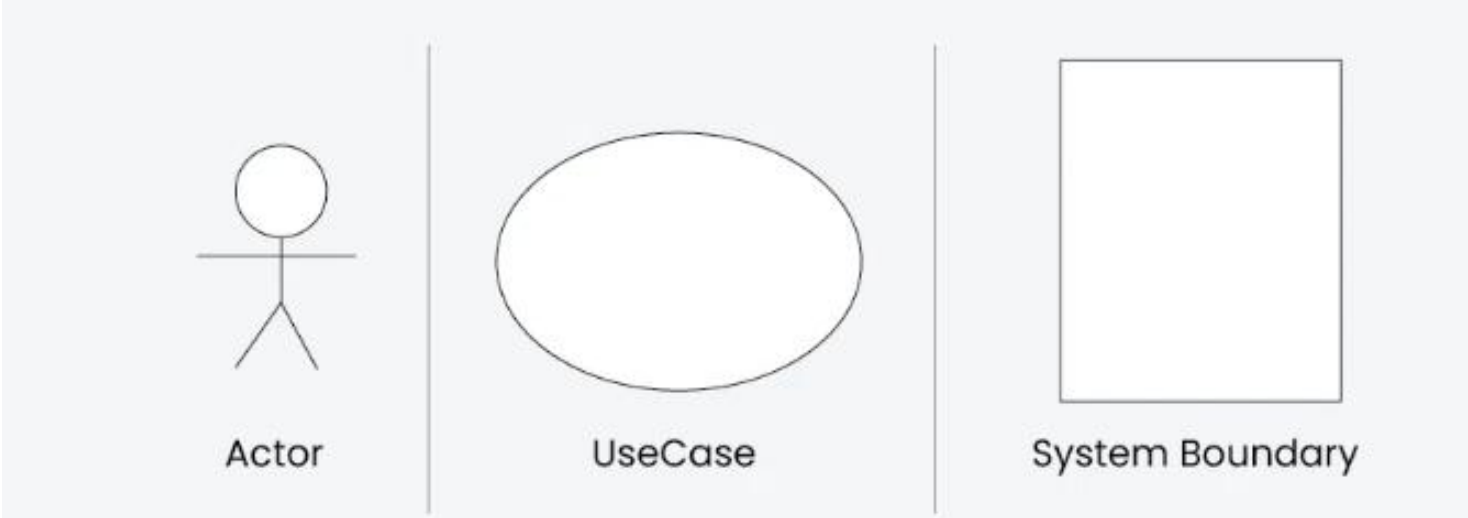
- **Lucidchart:** A web-based diagramming tool, user-friendly and collaborative
- **Draw.io:** A free, web-based diagramming tool
- **Visual Paradigm**: Offers both online and desktop versions and supports a wide range of UML diagrams.
- **StarUML:** An open-source UML modeling tool with a user-friendly interface. It supports the standard UML 2.x diagrams and allows users to customize and extend its functionality through plugins.
- **ArgoUML, Rational Rose**…etc

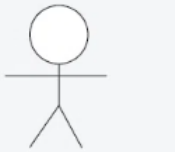
Use-case Diagrams

Defines what the system should do (user perspective).

Use-Case Diagrams

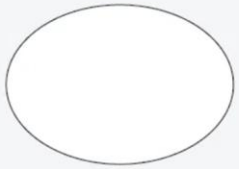
- Represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals.





Actors are typically represented by stick figures

Use Case Diagram Notations



Use cases are represented by ovals.

Actors

- External entities that interact with the system.
- Named by noun.
- Plays a role in the business
- Include users, other systems, or hardware devices.
- Initiate use cases.

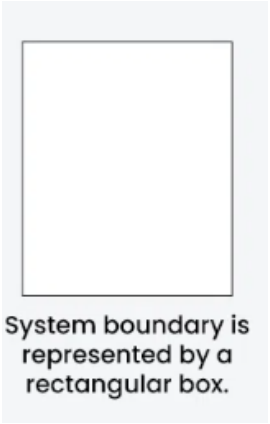
Use Cases

- Represent specific things the system can do.
- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
- Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

Use Case Diagram Notations

System Boundary

- A visual representation of the scope or limits of the system
- Defines what is inside the system and what is outside.
- Establish a clear distinction between the elements that are part of the system and those that are external to it.



System boundary is represented by a rectangular box.

Relationships

- Represent specific things the system can do.
- Depicts the interactions between actors and use cases.

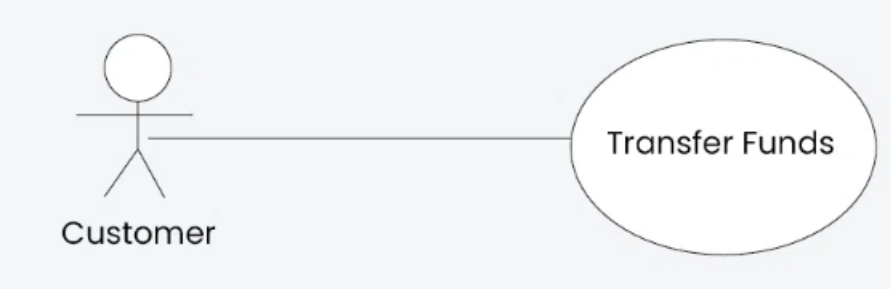
Relationships

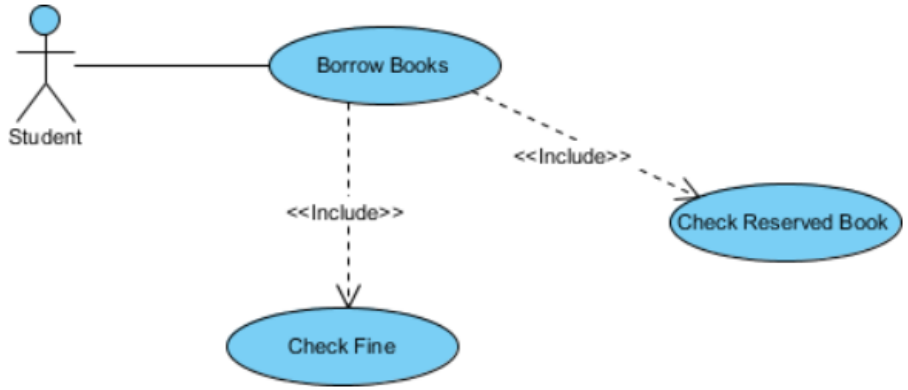
Use Case Diagrams

Relationships

Association Relationship

- Represents a communication or interaction between an actor and a use case.
- Depicted by a line connecting the actor to the use case.
- Signifies that the actor is involved in the functionality described by the use case.

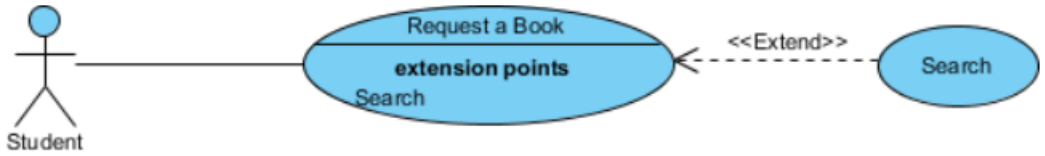




Include Relationship

- Indicates that a use case includes the functionality of another use case.
- Denoted by a dashed arrow pointing from the including use case to the included use case.
- The stereotype "<<include>>" identifies the relationship as an include relationship.

Relationships

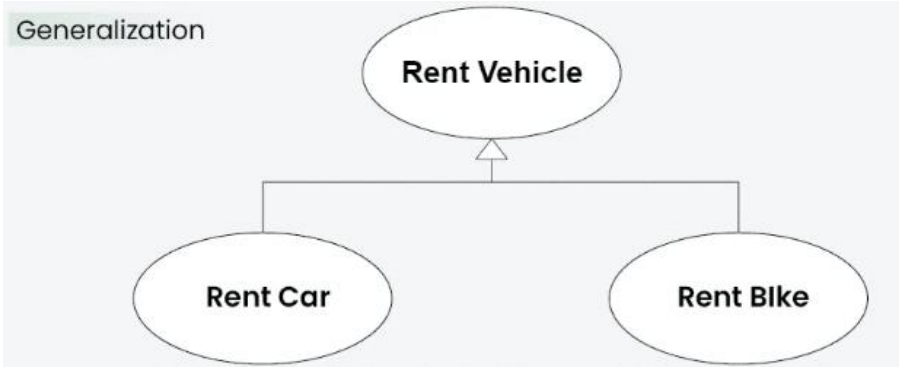


Extend Relationship

- Illustrates a use case that can be extended by another use case under specific conditions.
- Represented by a dashed arrow with the keyword "extend."
- Useful for handling optional or exceptional behavior.
- The stereotype "<<extends>>"

Generalization Relationship

- Establishes an "is-a" connection between two use cases, indicating that one use case is a specialized version of another.
- Represented by an arrow pointing from the specialized use case to the general use case.



How to Draw?

- **Step 1: Identify Actors**: Determine who or what interacts with the system (users, other systems, or external entities)
- **Step 2: Identify Use Cases**: Identify the main functionalities or actions the system must perform. Each use case should represent a specific piece of functionality.
- **Step 3: Connect Actors and Use Cases**: Draw lines (associations) between actors and the use cases they are involved in. This represents the interactions between actors and the system.
- **Step 4: Add System Boundary**: Draw a box around the actors and use cases to represent the system boundary. This defines the scope of the system.
- **Step 5: Define Relationships**: If certain use cases are related or if one use case is an extension of another, indicate the relationships with appropriate notations.
- **Step 6: Review and Refine**: Ensure that it accurately represents the interactions and relationships. Refine as needed.
- **Step 7: Validate**: Share the use case diagram with stakeholders and gather feedback.

Activity

Scenario: Airline Reservation System

How to Identify Actor (Schneider and Winters - 1998)

- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- What other systems use this system?
- Who gets information from this system?
- Who provides information to the system?
- Does anything happen automatically at a present time?

How to Identify Use Cases?

- What functions will the actor want from the system?
- Does the system store information? What actors will create, read, update or delete this information?
- Does the system need to notify an actor about changes in the internal state?
- Are there any external events the system must know about? What actor informs the system of those events?