MUST

Wisdom & Virtue

MIRPUR UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF SOFTWARE ENGINEERING

Software design & Architecture

(Lecture # 1)

UML Activity Modeling

Saba Zafar
(Lecturer)

Date: 31-10-2024

LECTURE CONTENTS

1. Activity Diagram
2. Activity Diagram Symbols
3. Examples of Activity Diagram
4. Swim-lane Activity Diagram

Activity diagram

- An activity diagram is a type of behavioral diagram in the Unified Modeling Language (UML) that illustrates the workflow or flow of control in a system, process, or algorithm. It visually represents the activities, actions, and flows of the system.
- Here are the main components and concepts of an activity diagram:
- Activities: Activities represent operations or actions that take place in the system. They can be atomic actions or complex processes.
- Actions: Actions are specific steps or tasks that are performed within an activity. They can be represented by rounded rectangles in the diagram.
- Transitions: Transitions show the flow of control from one activity to another. They indicate the sequence of actions and the order in which they are executed. Transitions are usually represented by arrows.

CONTD...

- Decisions: Decisions are used to represent conditional statements or branching in the flow of control. They help in illustrating the conditions under which different paths or actions are taken. Decisions are often depicted using diamond shapes.
- Initial and Final Nodes: The initial node indicates the starting point of the activity diagram, while the final node represents the end or completion of the process.
- Forks and Joins: Forks and joins are used to split the flow of control into multiple concurrent paths and merge them back, respectively. Forks are depicted by a single incoming transition splitting into multiple outgoing transitions, while joins merge multiple incoming transitions into a single outgoing transition.

Example’s

- Here's a simple example to illustrate these components:
- Initial Node: Start
- Activity: Enter Login Credentials
- Action: Enter Username
- Action: Enter Password
- Decision: Validate Credentials?
- Yes: Proceed to Dashboard
- No: Display Error Message
- Activity: Dashboard (if credentials are valid)
- Final Node: End

Scenario: A user wants to purchase an item from an online shopping website.

- Activity Diagram:
- Initial Node: Start
- Activity: Browse Products
- Action: Search for Product
- Decision: Product Found?
- Yes: Select Product
- No: Continue Browsing
- Activity: Add to Cart
- Activity: Proceed to Checkout

CONTD...

- Decision: Logged In?
- Yes: Enter Payment Details
- No: Login/Register
- Activity: Enter Login Credentials
- Decision: Valid Credentials?
- Yes: Proceed to Payment
- No: Display Error Message
- Activity: Confirm Order
- Final Node: End

Scenario: A customer wants to withdraw cash from an ATM.

- Initial Node: Start
- Activity: Insert ATM Card
- Decision: Card Valid?
- Yes: Enter PIN
- Decision: PIN Correct?
- Yes: Select Withdrawal Amount
- Decision: Sufficient Funds?
- Yes: Dispense Cash
- No: Display Insufficient Funds Message
- No: Retry/Cancel
- No: Eject Card
- Final Node: End

Scenario: A software development team is working on a new feature implementation.

- Initial Node: Start
- Activity: Requirement Analysis
- Activity: Design Solution
- Decision: Design Approved?
- Yes: Implement Feature
- Activity: Write Code
- Activity: Perform Unit Testing
- Decision: Tests Passed?

CONTD...

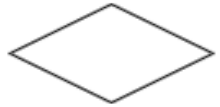
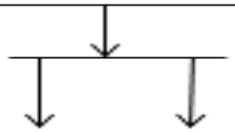
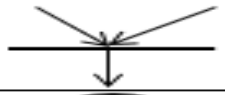
- Yes: Merge Code to Main Branch
- No: Debug and Fix Issues
- No: Revise Design
- Activity: Document Changes
- Final Node: End

ACTIVITY DIAGRAM SYMBOLS

- several symbols are used to represent different elements and actions within the diagram. Here's a list of the commonly used symbols:
- Initial Node:
- Symbol: Filled circle
- Description: Indicates the starting point of the activity diagram.
- Final Node:
- Symbol: Hollow circle (or filled circle with a border)
- Description: Represents the end or completion of the process.
- Activity:
- Symbol: Rounded rectangle
- Description: Represents an operation or action that takes place in the system. It can be atomic actions or complex processes.
- Action:

- Symbol: Rounded rectangle with a small label at the bottom
- Description: Represents specific steps or tasks performed within an activity.
- Decision:
- Symbol: Diamond shape
- Description: Represents a conditional statement or branching in the flow of control. It helps illustrate the conditions under which different paths or actions are taken.
- Transition (Flow):
- Symbol: Arrow
- Description: Indicates the flow of control from one activity or action to another, representing the sequence and order of execution.

- Fork:
- Symbol: Black bar with one incoming transition splitting into multiple outgoing transitions
- Description: Represents the splitting of the flow of control into multiple concurrent paths.
- Join:
- Symbol: Black bar with multiple incoming transitions merging into one outgoing transition
- Description: Represents the merging of multiple concurrent paths back into a single flow of control.

Sr. No	Name	Symbol
1.	Start Node	
2.	Action State	
3.	Control Flow	
4.	Decision Node	
5.	Fork	
6.	Join	
7.	End State	

Difference between Standard activity & swim lane activity

- Both swim lane activity diagrams and standard activity diagrams are used to model workflows and processes within a system, but they differ in the way they organize and represent the flow of activities and responsibilities among different entities or roles involved. Here's a comparison to highlight the differences between the two:

- Standard Activity Diagram:
- Basic Structure: Represents the flow of activities, actions, and transitions without explicitly defining the roles or responsibilities of different entities.
- Elements: Uses standard UML symbols such as activities (rounded rectangles), actions, decisions (diamonds), transitions (arrows), etc., to depict the workflow.
- Scope: Focuses on illustrating the sequence of actions and interactions within a system or process, without specifying which entities or roles are responsible for each activity.

CONTD...

- Swim lane Activity Diagram:
- Structure: Organizes activities and actions into lanes or swim lanes, representing different entities, roles, or departments involved in the process.
- Lanes: Each lane corresponds to a specific entity, role, or department, and activities within a lane indicate the responsibilities or tasks associated with that entity.
- Interaction: Shows how different entities interact with each other and contribute to the overall workflow. Arrows crossing between lanes indicate interactions or handoffs between different entities.
- Clarity: Provides a clearer and more structured view of the responsibilities and interactions among different entities, making it easier to understand the flow of activities and identify potential bottlenecks or issues.

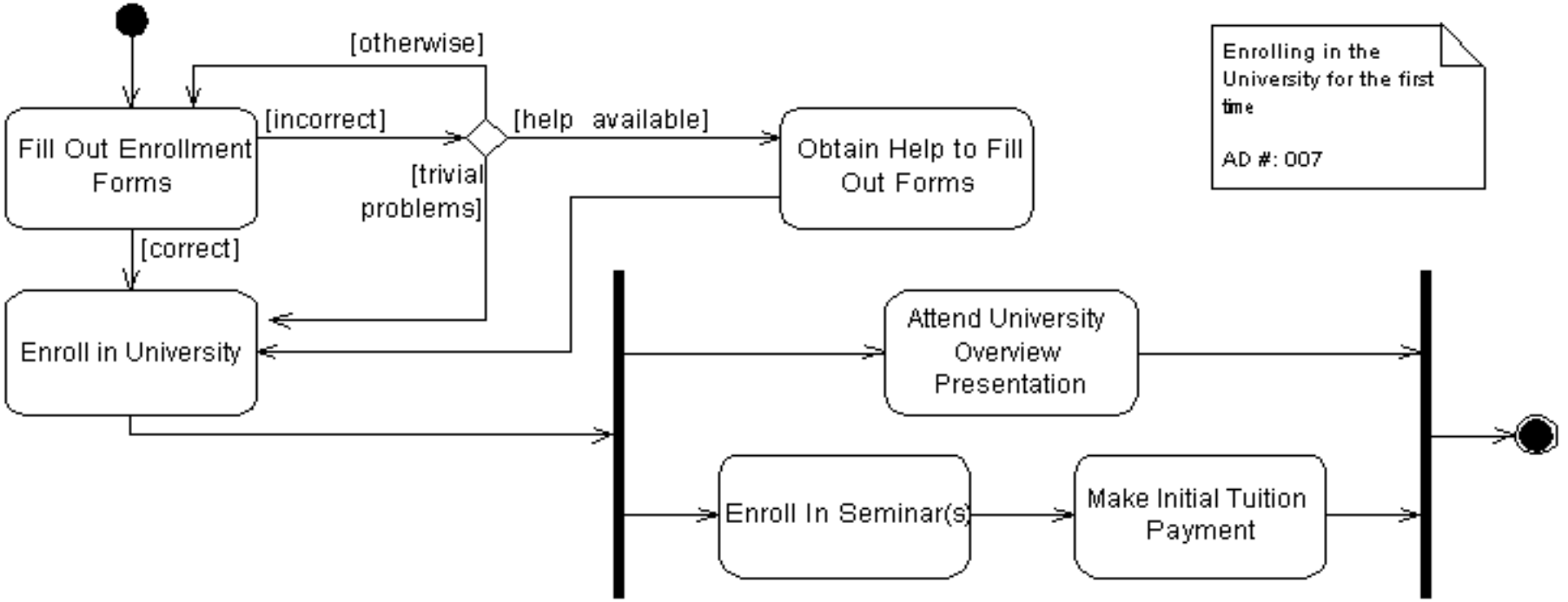
Example:
Standard Activity Diagram:

- Activity: Enter Login Credentials
- Action: Enter Username
- Action: Enter Password
- Decision: Validate Credentials?
- Yes: Proceed to Dashboard
- No: Display Error Message

Swim lane Activity Diagram:

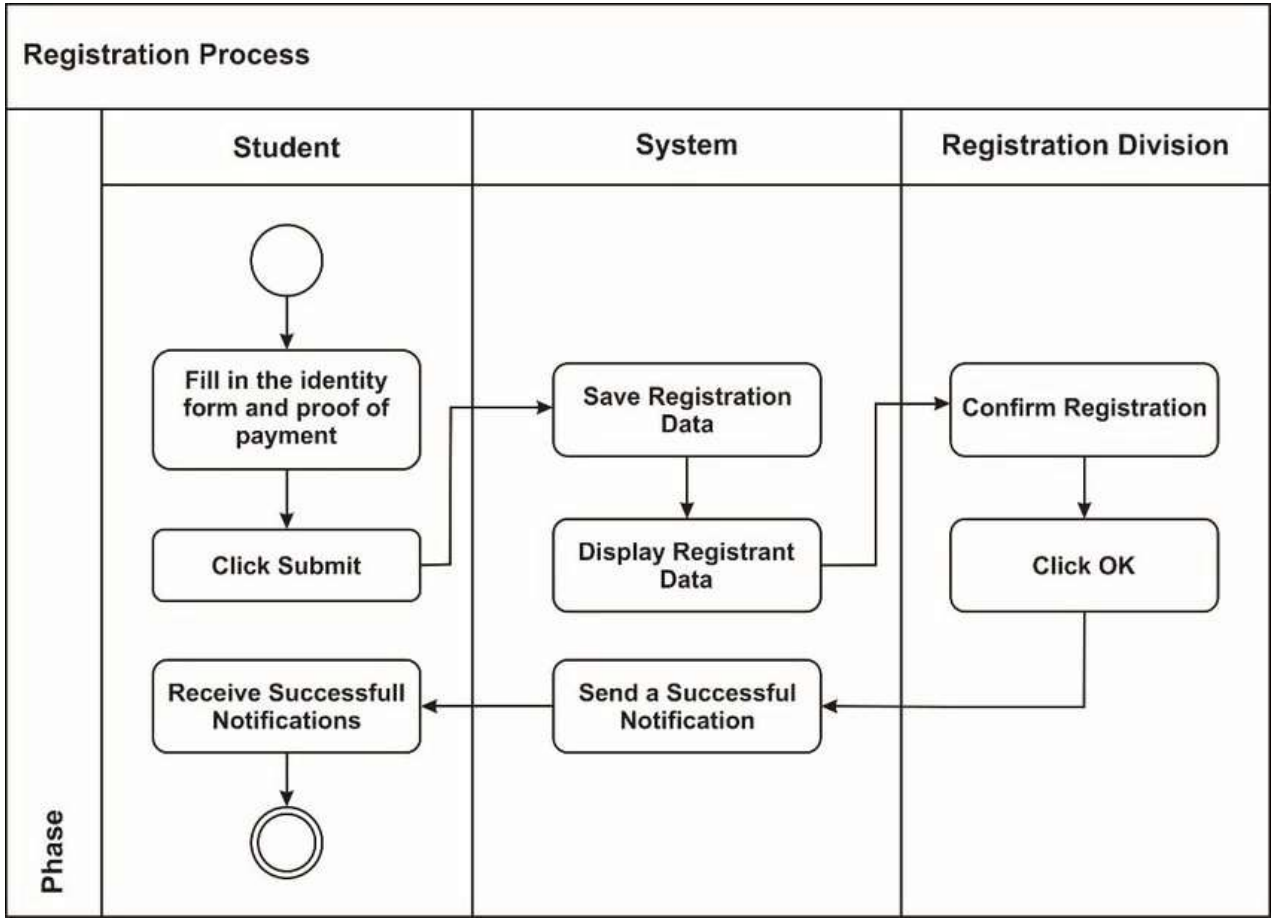
- Lane A (User):
- Activity: Enter Login Credentials
- Action: Enter Username
- Action: Enter Password
- Lane B (System):
- Decision: Validate Credentials?
- Yes: Proceed to Dashboard
- No: Display Error Message
- In the swimlane activity diagram, the responsibilities of the user and the system are clearly separated into different lanes, providing a more organized and structured representation of the login process.

ACTIVITY DIAGRAM :EXAMPLE 1



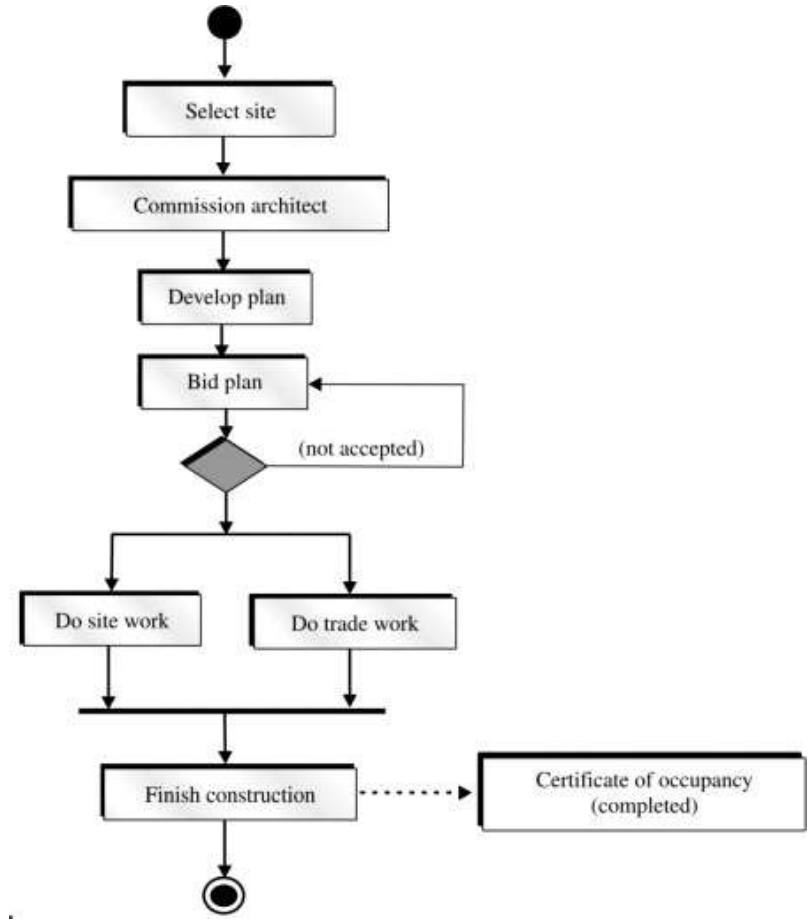
1. Student registration process activity diagram

- This activity diagram shows the series of actions performed by the student, the student registration system and the registration division to complete the student registration process.

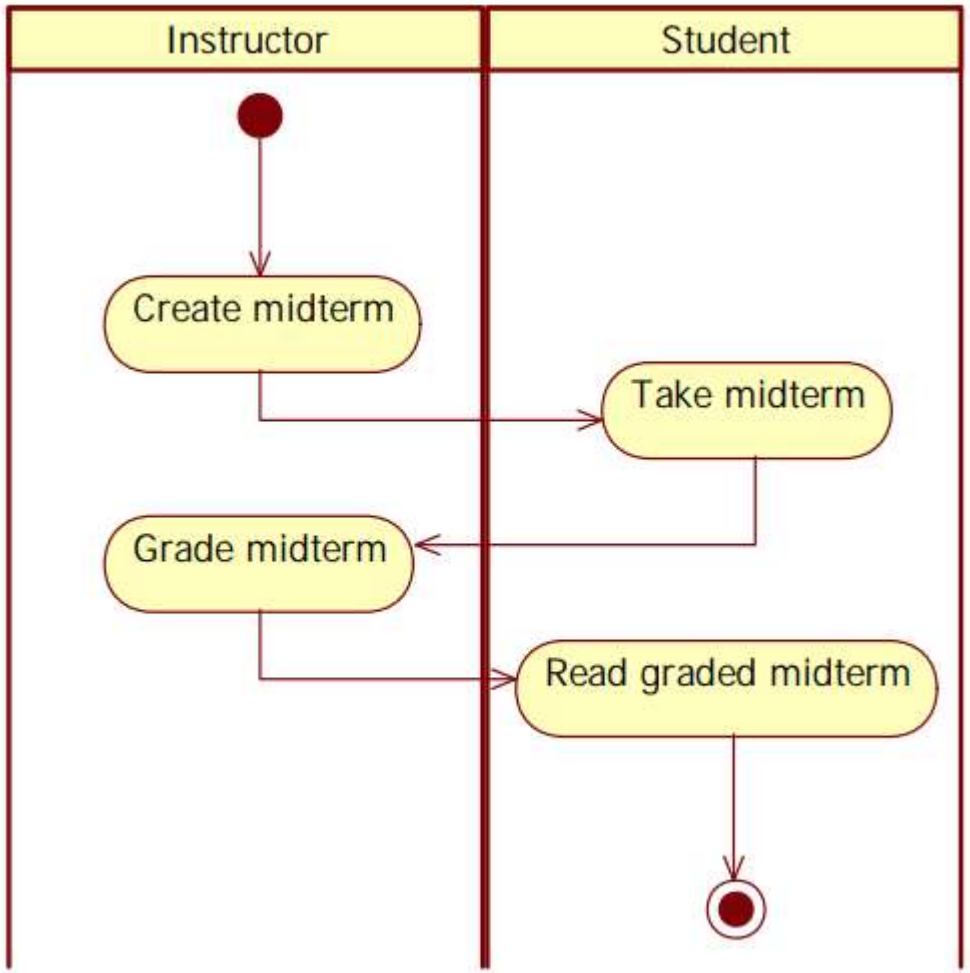

Registration Process
Student
System
Registration Division
Phase
Fill in the identity form and proof of payment
Click Submit
Save Registration Data
Display Registrant Data
Confirm Registration
Click OK
Receive Successfull Notifications
Send a Successful Notification

2. Construction process activity diagram

- Here’s another example of an activity diagram that is used to outline the steps of a business process. Utilizing different activity diagram symbols, this example makes use of a fork and join node to start and end a concurrent flow:


Select site
Commission architect
Develop plan
Bid plan
(not accepted)
Do site work
Do trade work
Finish construction
Certificate of occupancy
(completed)

ACTIVITY DIAGRAM (SWIMLANE DIAGRAM) : EXAMPLE 1



LECTURE CONTENTS

1. Static Object Modeling
2. Class Diagram
3. Composition Relationship
4. Aggregation Relationship

STATIC OBJECT MODELING

- *Focus is on the static structure of the system.*
- ***UML Notation**: Class Diagram*

The UML Class Diagram:

- *A class diagram describes the static view of a system in terms of classes and relationships among the classes*
- *The UML includes class diagrams to illustrate classes and their associations.*

Class Diagram

- A class diagram is a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system or application. It provides a high-level overview of the classes, their attributes, methods, and relationships with other classes.

CONTD...

- Class: Represents a blueprint for objects. It typically consists of three compartments:
- **Class Name**: The name of the class.
- **Attributes**: Properties or data members of the class.
- **Methods**: Functions or operations that the class can perform.
- Association: Represents a relationship between two classes. It indicates that objects of one class are connected to objects of another class in some way. Associations can be one-to-one, one-to-many, or many-to-many.
- Multiplicity: It's used on association lines to indicate how many instances of one class are related to instances of another class. For example, 1..* means "one or more" and 0..1 means "zero or one".

CONTD...

- **Inheritance (Generalization)**: Represents an "is-a" relationship between classes, where one class is a specialization of another. The arrow points from the subclass (child) to the superclass (parent).
- **Dependency**: Represents a relationship where one class depends on another in some way. It's usually denoted by a dashed arrow.
- **Aggregation**: Represents a "whole-part" relationship between classes, where one class is a container for other classes. It's denoted by a hollow diamond at the container end.
- **Composition**: Similar to aggregation but with stronger ownership. It represents a "stronger" whole-part relationship, where the part cannot exist without the whole. It's denoted by a filled diamond at the container end.

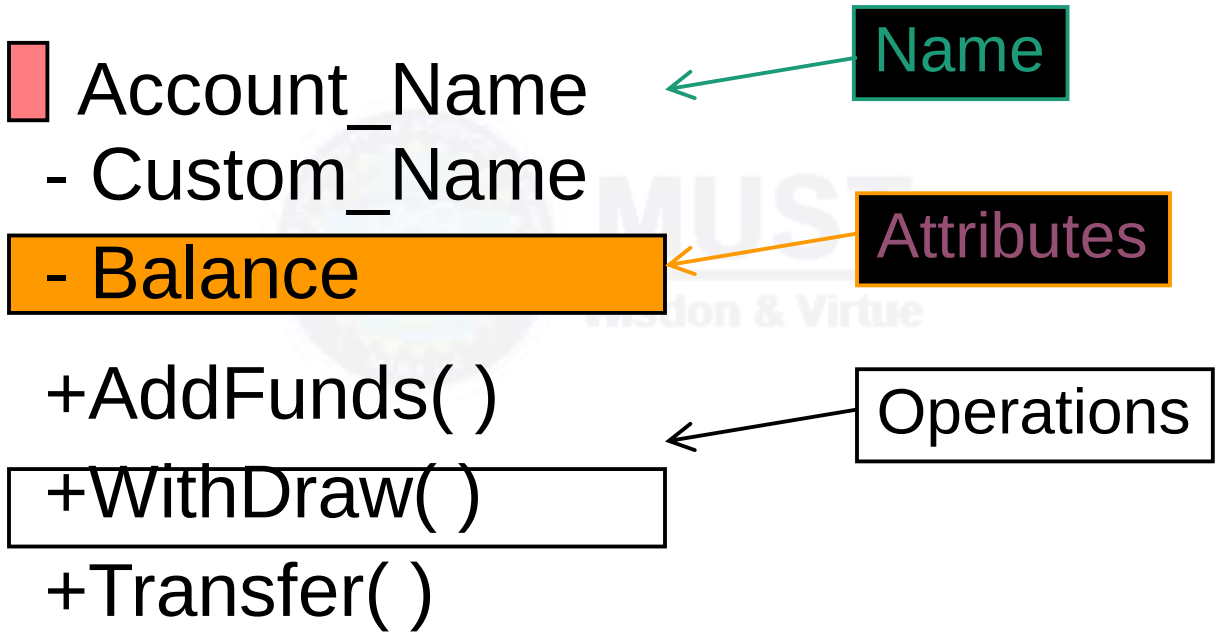
CONTD...

- Aggregation Example:
- Imagine we have classes for University and Department. A University contains multiple Departments, but a Department can exist independently of the University, meaning it can be associated with different universities over time.
- Composition Example:
- Let's consider a University and Student relationship. A University has multiple Students, but if the University ceases to exist, the Students associated with it should also cease to exist. This stronger form of association is composition.

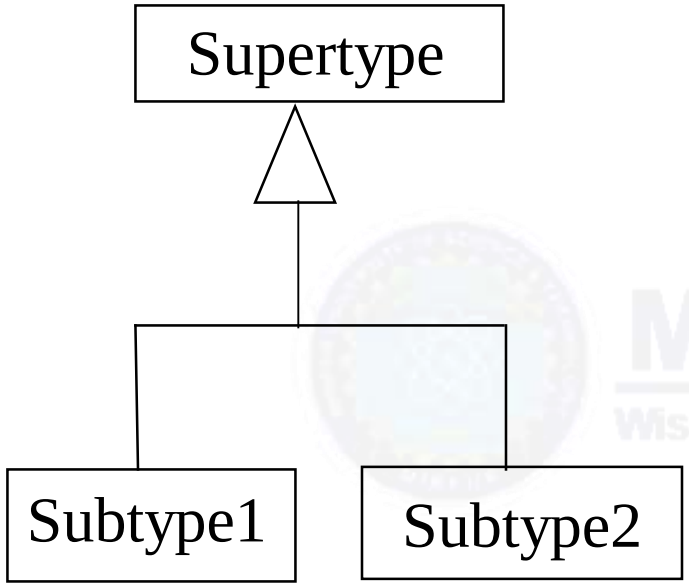
CLASS DIAGRAMS

- *Each class is represented by a rectangle subdivided into three compartments*
  - *Name*
  - *Attributes*
  - *Operations*
- *Modifiers are used to indicate visibility of attributes and operations.*
  - *'+' is used to denote Public visibility (everyone)*
  - *'#' is used to denote Protected visibility (friends and derived)*
  - *'-' is used to denote Private visibility (no one)*
- ***By default**, attributes are hidden and operations are visible*
- *The last two compartments may be omitted to simplify the class diagrams*

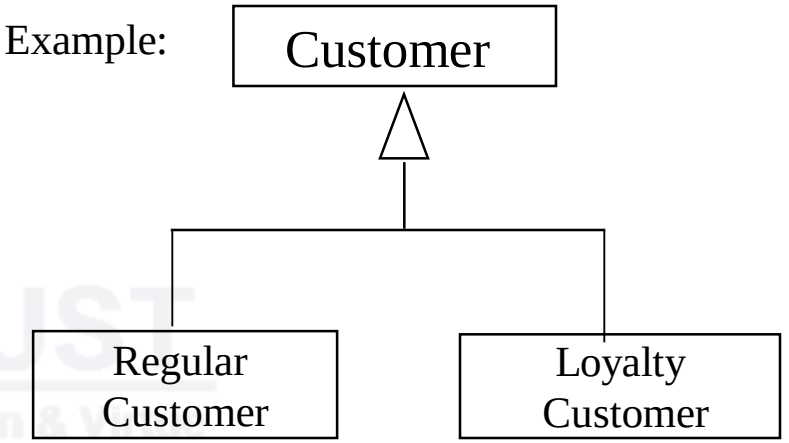
AN EXAMPLE OF CLASS



NOTATION OF CLASS DIAGRAM: GENERALIZATION

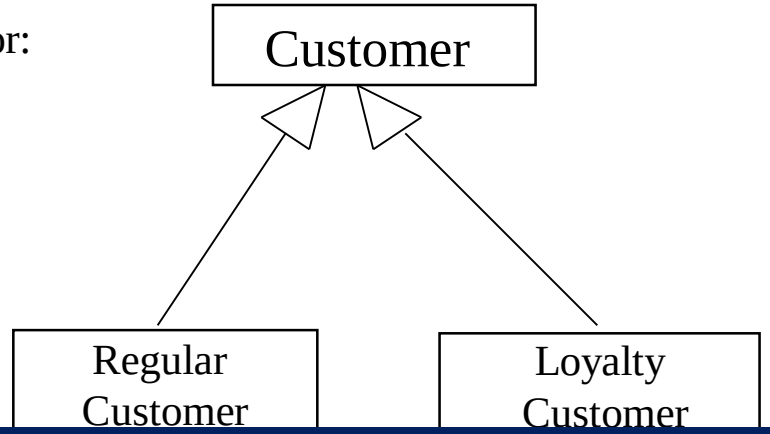


Example:

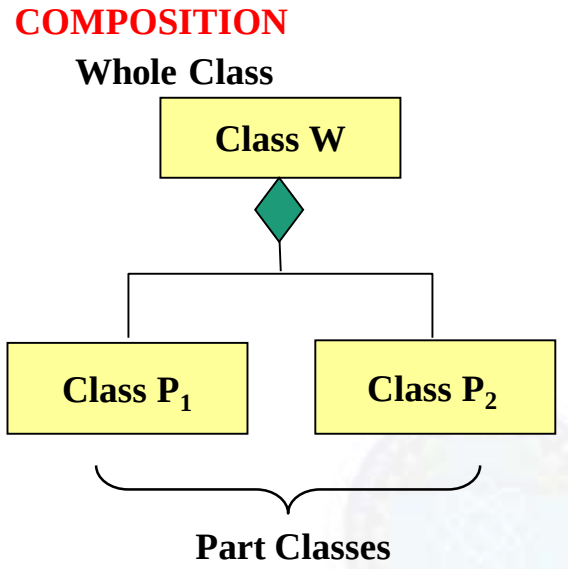


Generalization expresses a relationship among related classes. It is a class that includes its subclasses.

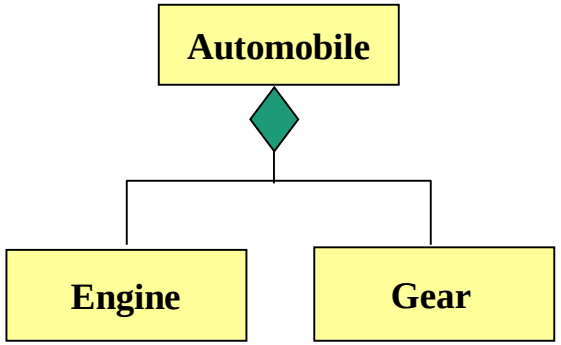
or:



NOTATION OF CLASS DIAGRAM :COMPOSITION



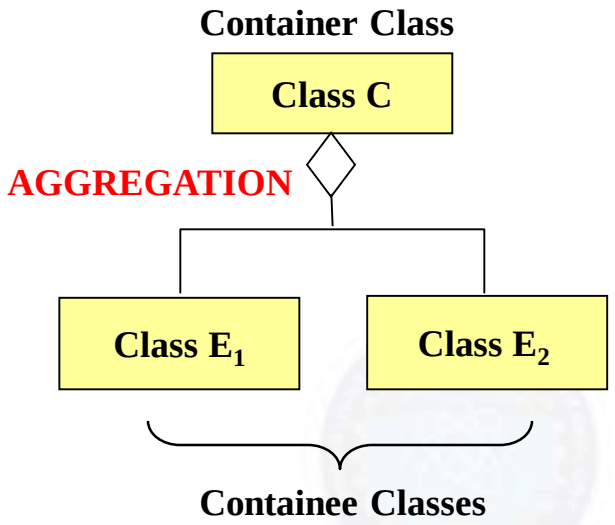
Example



Composition: Expresses a relationship among instances of related classes. It is a specific kind of **Whole-Part** relationship.

- Composition should be used to express a relationship where the behavior of Part instances is undefined without being related to an instance of the Whole.
- And, conversely, the behavior of the Whole is ill-defined or incomplete if one or more of the Part instances are undefined.

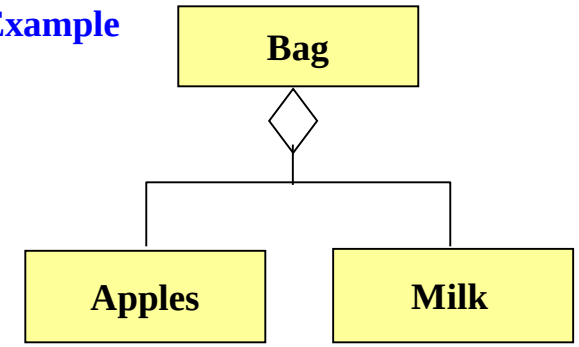
NOTATION OF CLASS DIAGRAM :AGGREGATION



***Aggregation**: Expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.*

Aggregation is an appropriate relationship where the Container and its Containees can be manipulated independently.

Example



AGGREGATION VS COMPOSITION

Composition *is really a strong form of* ***aggregation***

- *Components cannot exist independent of their owner; both have coincident lifetimes*
- *Components live or die with their owner*

e.g. (1)Each car has an engine that cannot be shared with other cars

(2) If the polygon is destroyed, so are the points

Aggregations *may form "part of" the aggregate, but may not be essential to it. They may also exist independent of the aggregate. Less rigorous than a composition*

e.g. (1)Apples may exist independent of the bag

(2)An order is made up of several products, but the products are still there even if an order is cancelled

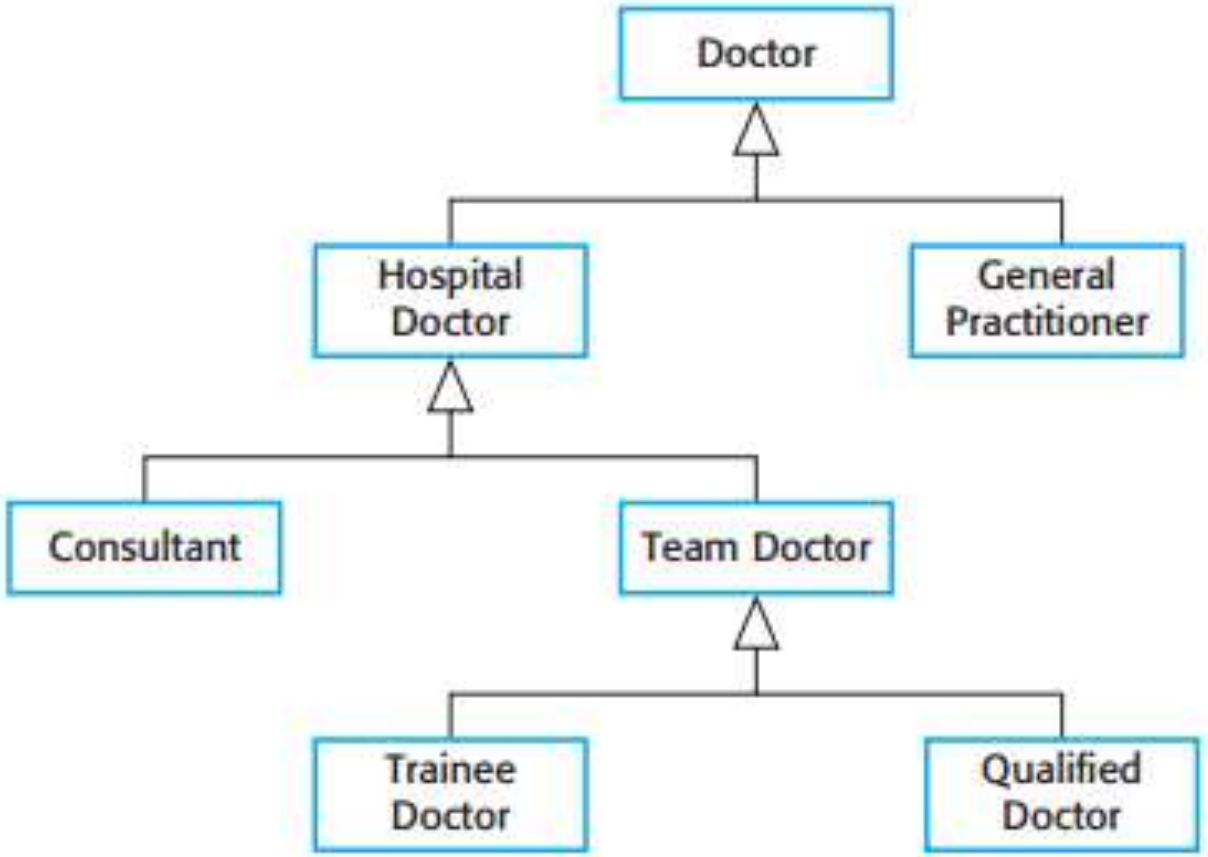
- Aggregation:
- Aggregation represents a "has-a" relationship where one class (the whole) contains or is composed of other classes (the parts).
- It's often depicted as a diamond shape on the containing class's end, connected to the contained class(es).
- Aggregation implies a relationship where the contained class(es) can exist independently of the containing class.
- The lifecycle of the contained objects is not tightly bound to the lifecycle of the containing object.
- Example: A university contains departments. A department can exist independently of the university (though in reality, the relationship might be more complex).

- **Composition**:
  - Composition is a stronger form of aggregation, representing a whole-part relationship where the part cannot exist without the whole.
  - It's also depicted with a diamond shape, but with a filled diamond, indicating strong ownership.
  - Composition implies that the lifetime of the contained objects is controlled by the containing object. If the containing object is destroyed, all its contained objects are typically destroyed as well.
  - Example: A car contains an engine. Without the car, the engine has no purpose, and it's typically destroyed along with the car.

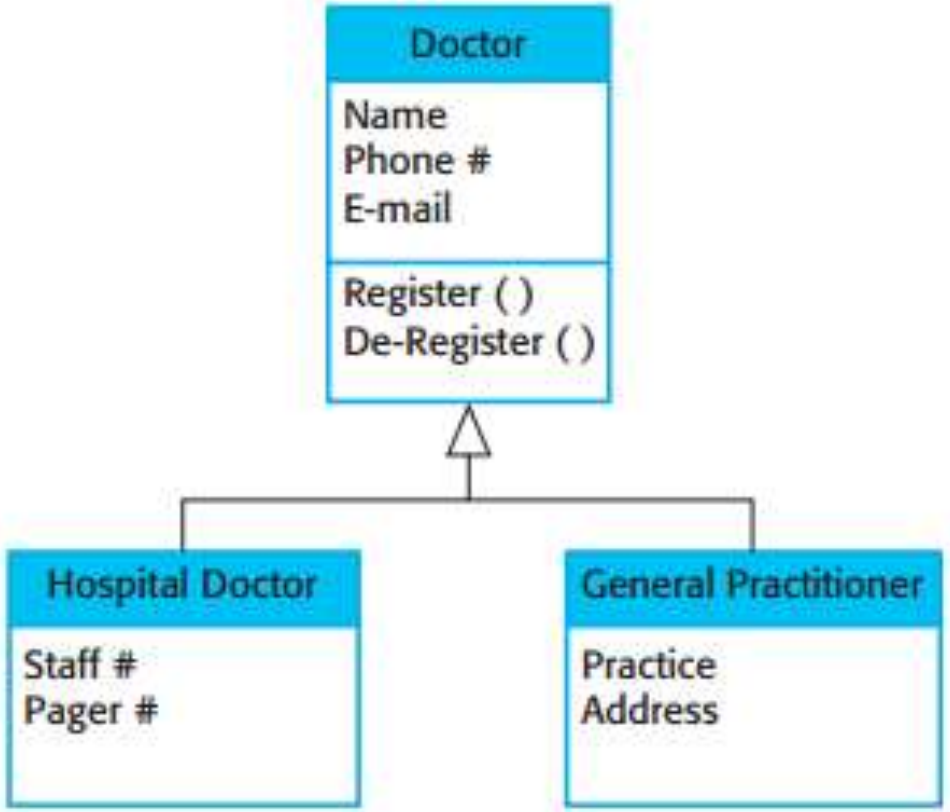
CLASS MODEL : GENERALIZATION (INHERITANCE)

- *In object-oriented languages, such as C++ and Java, generalization is implemented using the class inheritance mechanisms built into the language.*
- *In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.*
- *The lower-level classes are subclasses inherit the attributes and operations from their super-classes.*
- *These lower-level classes then add more specific attributes and operations.*

A GENERALIZATION HIERARCHY



A GENERALIZATION HIERARCHYWITH ADDED DETAIL



ENCAPSULATION

- When an object carries out its operations, those operations are hidden
- When most people watch a TV show, they usually don't know or care about:
  - The complex electronics components that sit in back of the TV screen
  - All operations that have to occur in order to paint the image on the screen
- The TV does what it does and hides the process from us
- In the software world, encapsulation helps cut down on the potential for bad things to happen

PURPOSE OF CLASS DIAGRAMS

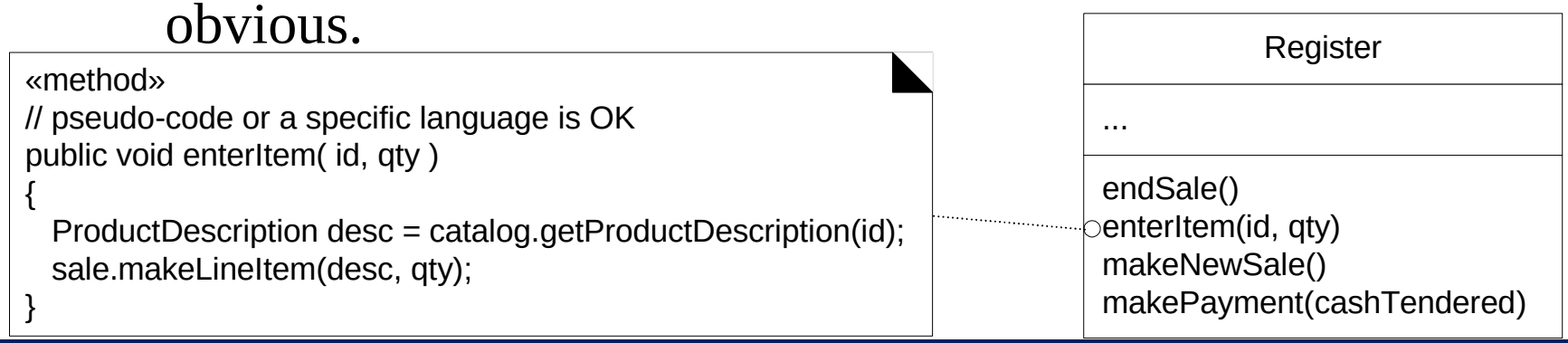
- *The purpose of class diagram is to model the static view of an application*
- *Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.*

The purpose of the class diagram can be summarized as −

- *Analysis and design of the static view of an application*
- *Describe responsibilities of a system*
- *Base for component and deployment diagrams*
- *Forward and reverse engineering.*

NOTE CAN ALSO BE USED

- *Note symbols can be used on any UML diagram, but are especially common on class diagrams*
- *A UML note symbol is displayed as a dog-eared rectangle with a dashed line to the annotated element*
- **Clarification**: Class diagrams can become complex, especially in larger systems or when modeling intricate relationships. Notes allow designers to clarify certain aspects of the diagram that might not be immediately obvious.



CONTD...

- **Documentation**: Notes serve as a form of documentation within the diagram itself. They provide insights into design decisions, constraints, assumptions, or any other relevant information that might not be apparent from the structure of the diagram alone. This can be helpful for developers who come across the diagram later or for communicating with stakeholders.
- **Annotations**: Notes can be used as annotations to highlight specific points of interest or to explain the rationale behind certain design choices. This makes the diagram more informative and helps stakeholders understand the underlying design principles.

Dependency

- **Concept**:
  - Dependency indicates that one class, known as the client, uses or depends on another class, known as the supplier, in some manner.
  - It is a directional relationship, indicating that changes in the supplier class may affect the client class.
- **Usage**:
  - Dependencies are often used to model relationships where one class uses another class as a parameter type, return type, local variable type, or in some other way, but does not own or contain instances of the other class.
  - They are also used to represent temporary relationships or interactions between classes that are not part of the main structure of the system.

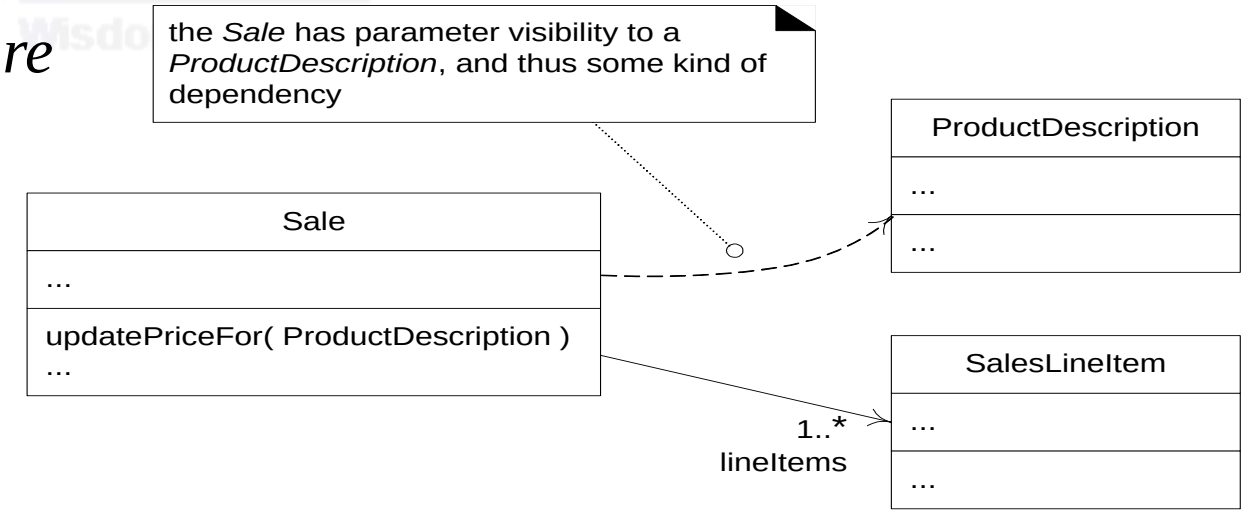
Contd...

- **Representation**:
  - In a class diagram, dependencies are typically represented by a dashed arrow pointing from the client class to the supplier class.
  - The arrow indicates the direction of the dependency, showing which class depends on the other.
  - No specific notation is used on the supplier class end; the arrow itself indicates the relationship.
  - Example:
  - Suppose you have a Car class that depends on a Engine class. The Car class may use the Engine class for powering the car.
  - This dependency is represented in the class diagram by a dashed arrow pointing from the Car class to the Engine class.

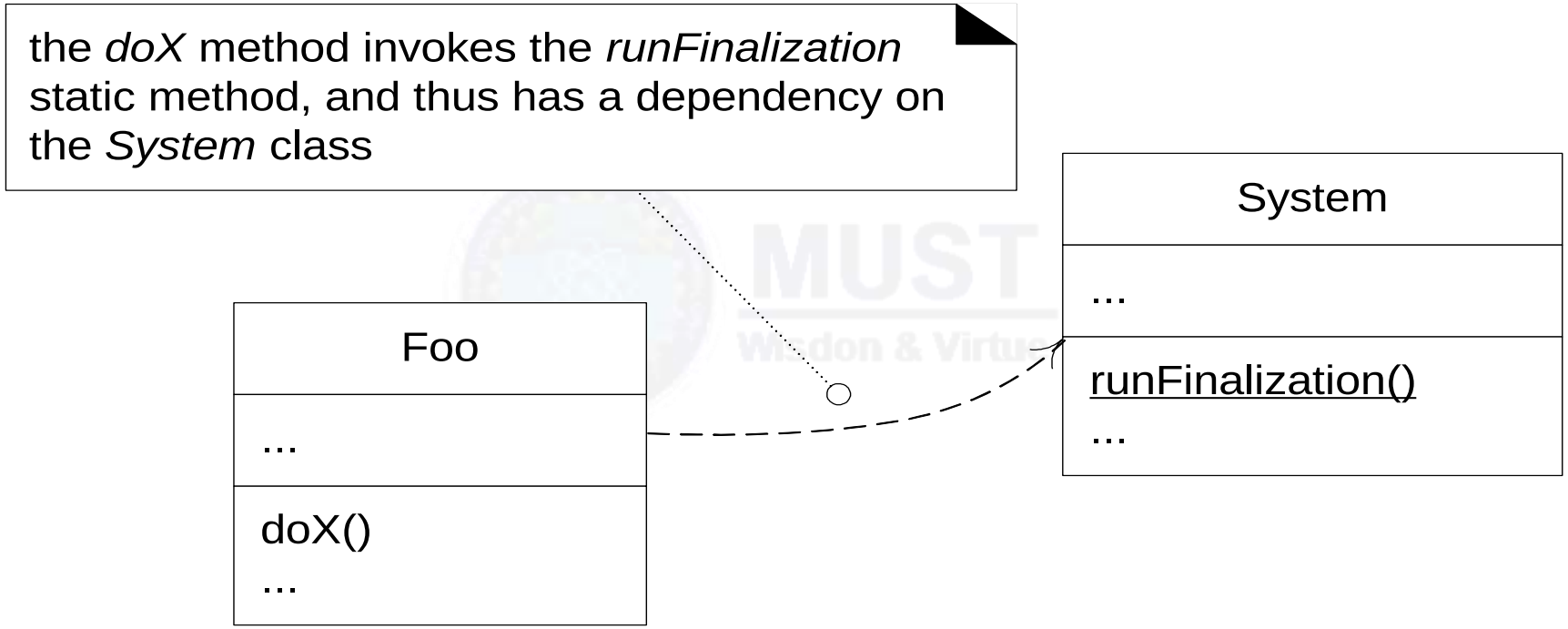
- **Impact**:
  - Changes in the supplier class may impact the client class, potentially requiring modifications in the client class to accommodate those changes.
  - However, the dependency is relatively loose compared to stronger relationships like aggregation or composition. The client class does not own or control instances of the supplier class; it merely relies on it.

SHOWING DEPENDENCY

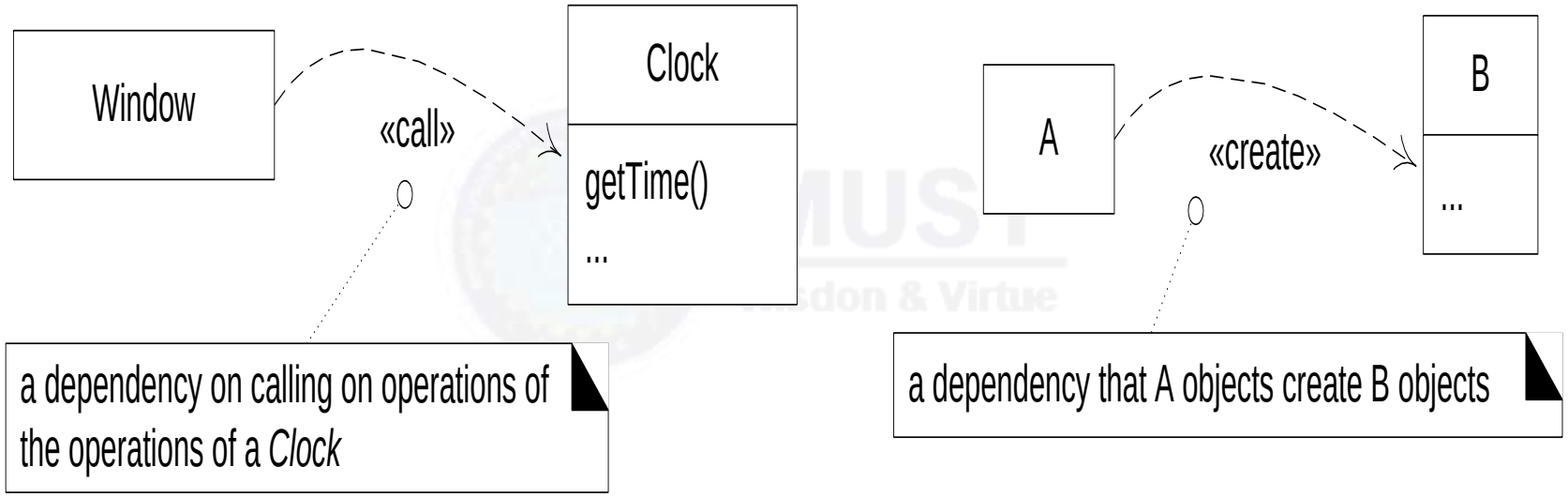
- *The UML includes a general dependency relationship that indicates that a **client element** (classes) has knowledge of another **supplier element** (other class) and that a change in the supplier could affect the client.*
- *Dependency is illustrated with a dashed arrow line from the client to supplier.*
- *There are many kinds of dependency; here are some common types:*
  - *Having an attribute of the supplier type*
  - *Sending a message to a supplier*
  - *Receiving a parameter of the supplier type*



SHOWING DEPENDENCY



OPTIONAL DEPENDENCY LABLES



Optional dependency labels

- In UML class diagrams, optional dependency labels can be used to denote the conditions under which a dependency exists or to specify additional information about the dependency relationship. However, it's important to note that optional dependency labels are not standard UML notation; instead, they are used in some modeling tools or methodologies to provide extra clarity or detail.

Contd...

- Conditional Dependency:
- Sometimes, a dependency between classes might only exist under certain conditions or scenarios. An optional dependency label can be used to indicate these conditions.
- For example, if a class A depends on class B only when a certain feature is enabled, you could annotate the dependency arrow with a label like "when Feature X is enabled".

Contd...

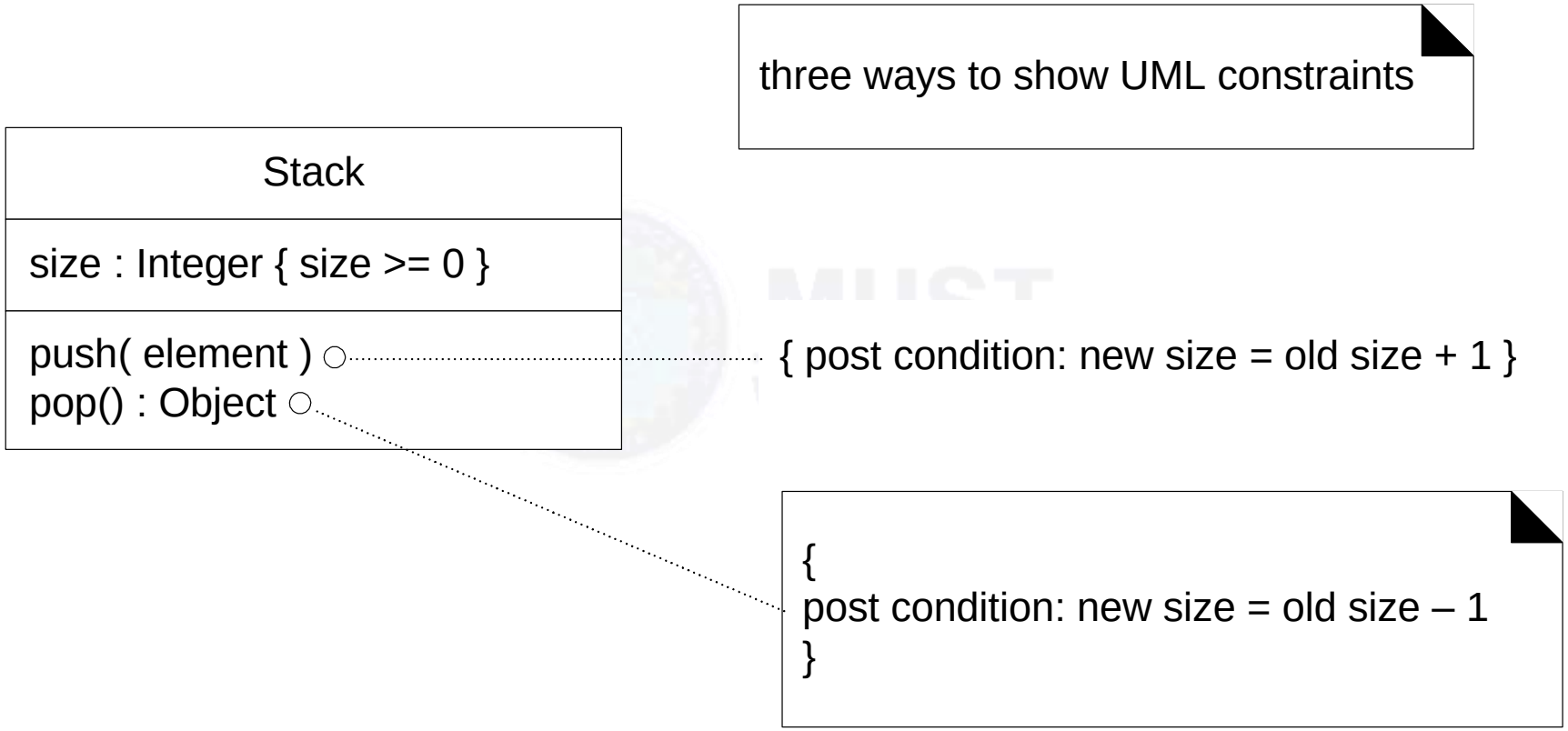
- Additional Information:
- Optional dependency labels can also be used to provide additional information about the nature or purpose of the dependency.
- This might include explanations about why the dependency exists, how it's utilized, or any constraints associated with it.
- For instance, you might label a dependency arrow with "Used for logging purposes" to clarify the purpose of the dependency.
- Tool-specific Extensions:
- Some modeling tools or methodologies may offer extensions or customizations to standard UML notation, allowing users to add labels or annotations to elements like dependency arrows.
- These optional labels might be supported through the tool's interface or by manually adding text to the diagram.

Types of Constraints

- Semantic Constraints: These constraints define the semantic rules or business logic that govern the behavior of the system. They ensure that the system operates according to the specified requirements.
- Structural Constraints: These constraints define the structure and relationships between classes. They specify conditions such as cardinality, multiplicity, or inheritance rules.
- Behavioral Constraints: These constraints define the behavior of the classes, including their methods, operations, and interactions.

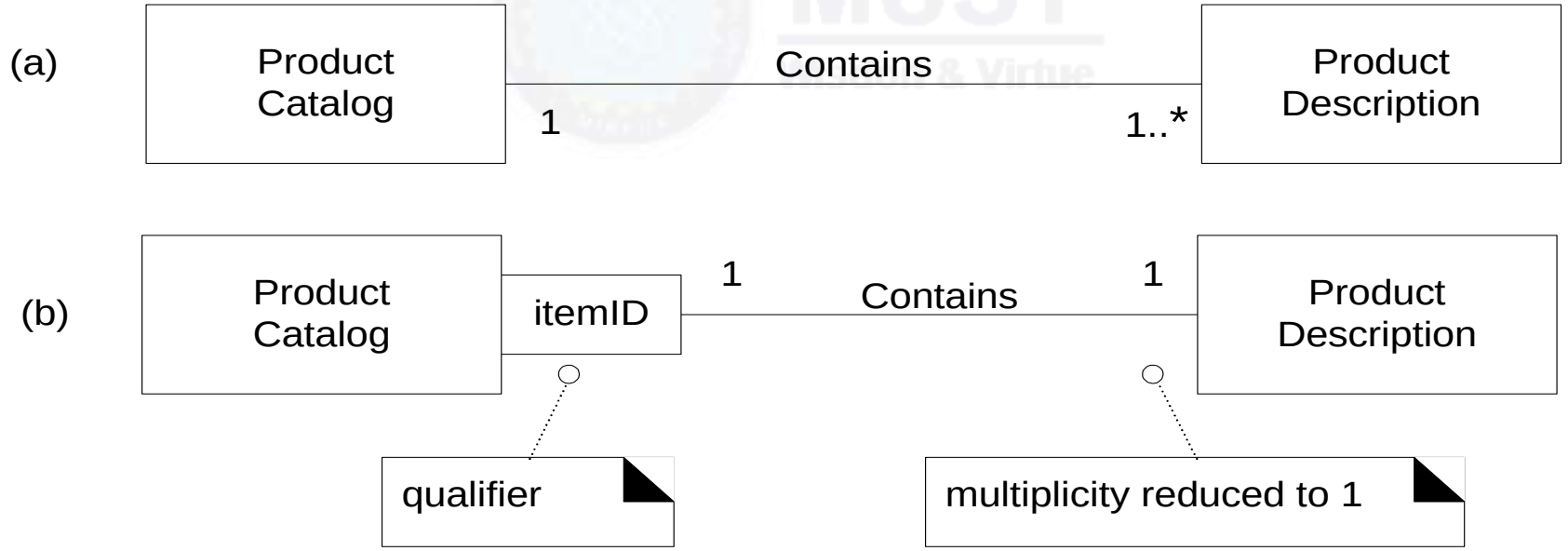
- Examples:
- Semantic Constraint: {age > 18} - Specifies that the 'age' attribute of a 'Person' class must be greater than 18.
- Structural Constraint: {0..*} - Specifies that a class has zero or more instances of a related class.
- Behavioral Constraint: {void update()} - Specifies that the 'update' method of a class does not return any value.

CONSTRAINTS



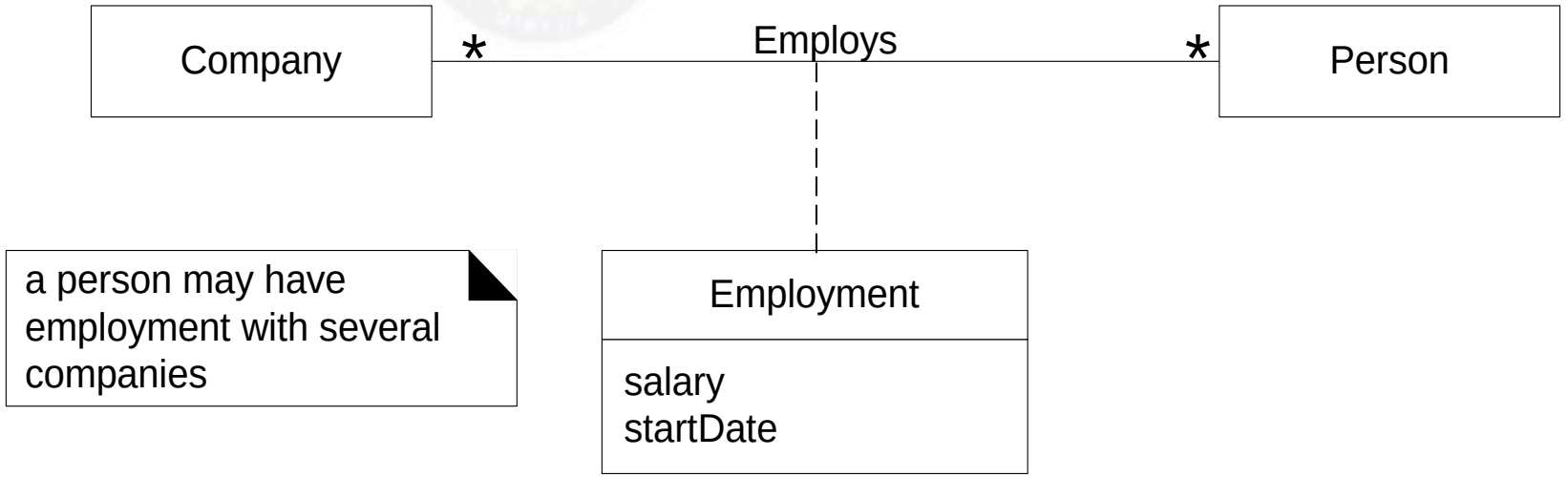
QUALIFIED ASSOCIATION

- *A **qualified association** has a **qualifier** that is used to select an object from a larger set of related objects, based upon the **qualifier key***
- *For example, if a **ProductCatalog** contains many **ProductDescriptions,** and each one can be selected by an **itemID,** then the UML notation in Figure can be used to depict this*

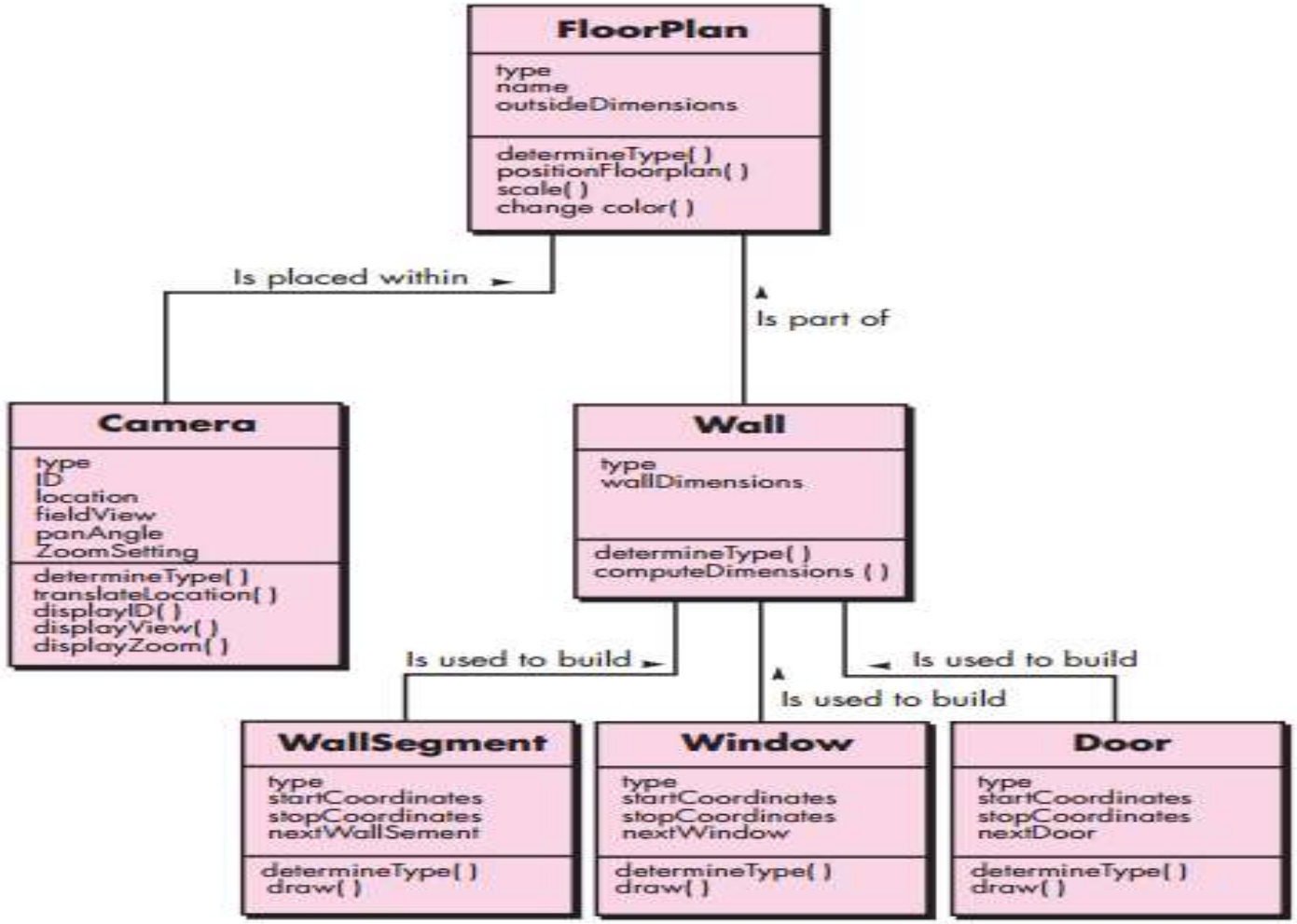


ASSOCIATION CLASS

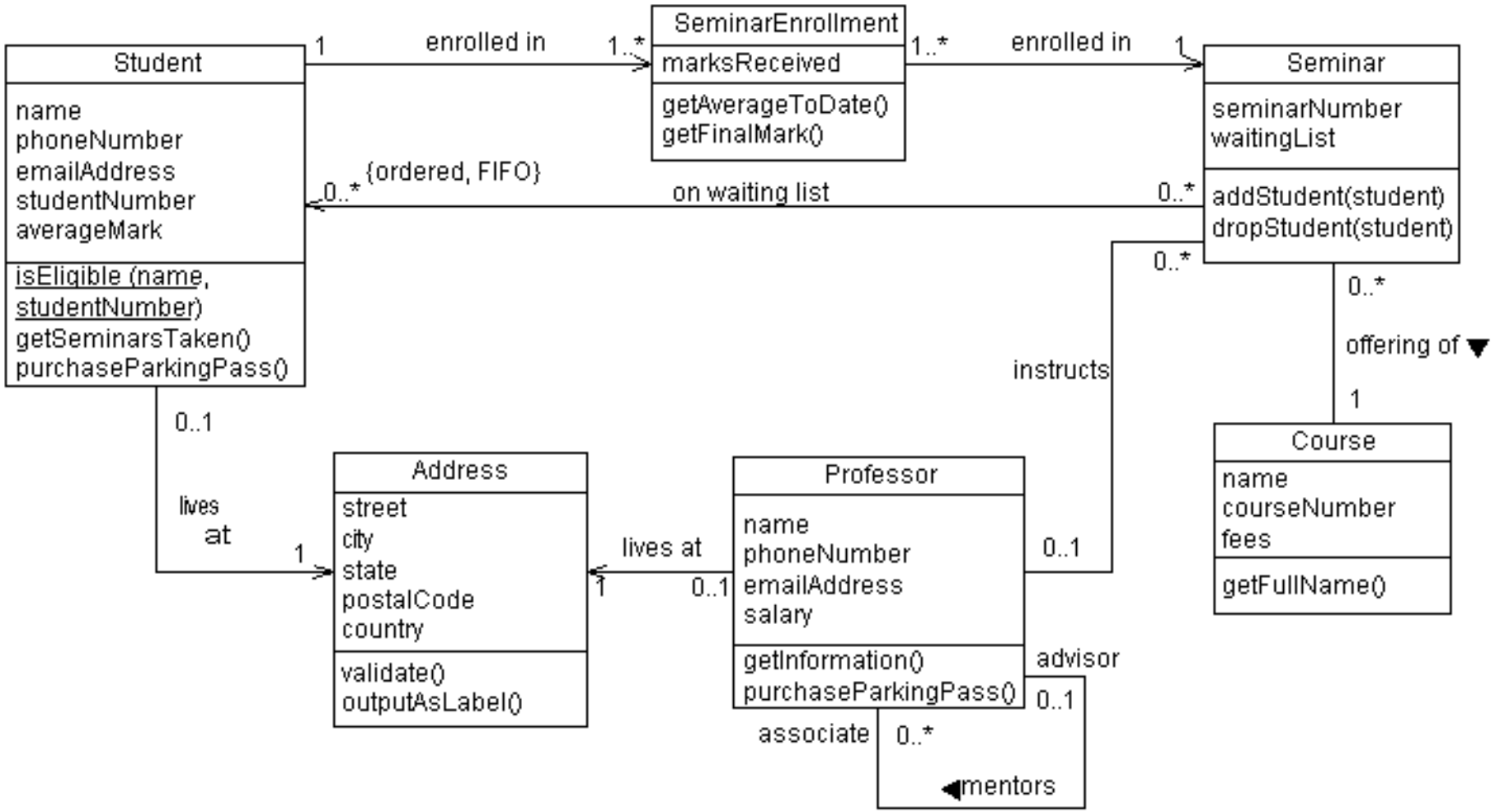
- *An **association class** allows you treat an association itself as a class, and model it with attributes, operations, and other features*
- ***For example**, if a Company employs many Persons, modeled with an Employs association, you can model the association itself as the Employment class, with attributes such as startDate*
- *In the UML, it is illustrated with a dashed line from the association to the association class.*



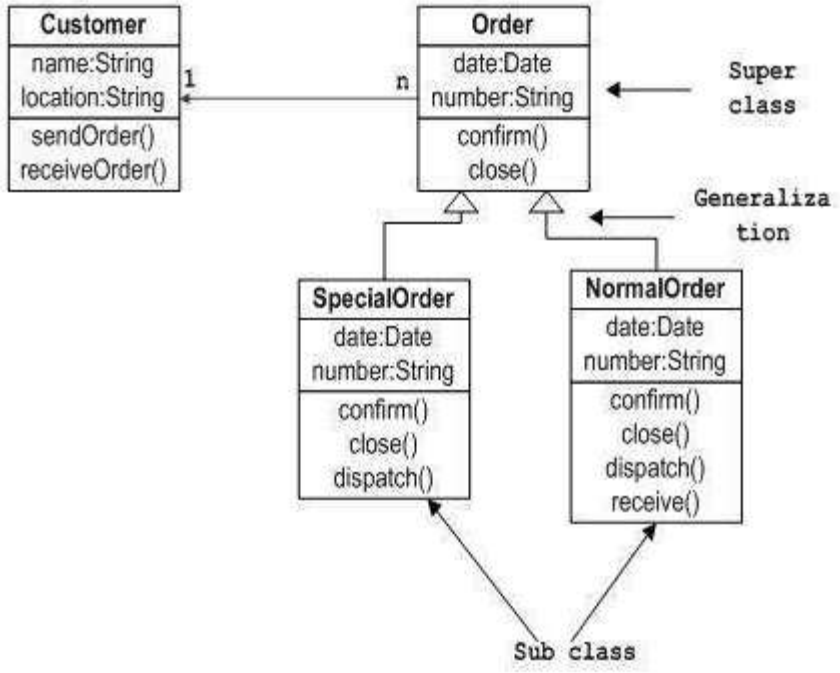
CLASS MODEL EXAMPLE 1



CLASS MODEL EXAMPLE 2



CLASS MODEL EXAMPLE 3



Task 1(Activity Diagram)

- In an online food ordering system, a user begins by logging into the app, where they either successfully access their account or receive an error message and try again. Once logged in, they browse the menu, filtering items by categories such as appetizers or main courses. As they select food items, each choice is added to their cart, where they can review their selections, adjust quantities, or remove items if needed. Satisfied with the cart, the user proceeds to the payment section. If the cart is empty, they are prompted to add items before continuing. After choosing a payment method and entering payment details, the system validates the transaction. If successful, the user receives an order confirmation and estimated delivery time; if the payment fails, they are notified and can attempt the payment process again. The restaurant then prepares the order and hands it over for delivery, updating the user with real-time tracking. Finally, the user receives the order and has the option to provide feedback or rate the service, completing the transaction process.

Task 2-Class Diagram

- In an online food ordering system, the main entities involved are users, menu items, orders, payments, and deliveries. A **User** logs into the system to browse a **Menu** of available food items, which are organized by categories and displayed through the **Item** class. As the user selects items to purchase, they are added to their **Cart**, where the user can review, update quantities, or remove items. When ready, the user places an **Order**, which stores details of the selected items, the order date, and status. The **Payment** class manages the payment process, validating and confirming transactions, while the **Delivery** class handles the delivery details, including tracking and updates on estimated delivery time. Each order is associated with both a payment and a delivery, and a user can place multiple orders over time, each containing multiple items. This structure defines the core relationships and responsibilities within the system, creating a streamlined flow from user login to order completion.

THANKS