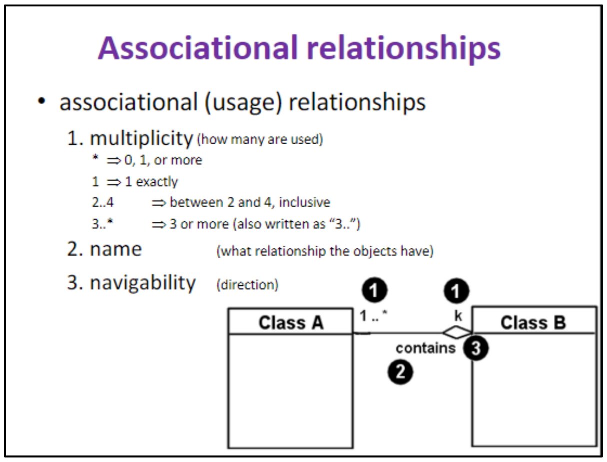
UML Notes:

CLASS NOTES:

19/8/2024:

- Uses for UML:
  - To communicate aspects of a system
  - Used as a blueprint
  - Used, for tools that create code from UML



Associational relationships

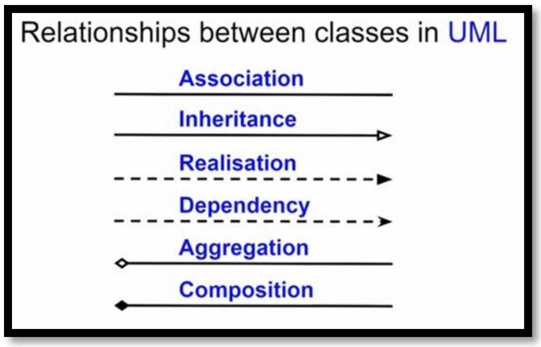
- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more (also written as "3..")
  2. name (what relationship the objects have)
  3. navigability (direction)

Class A
Class B
1..*
k
contains
1
1
2
3

20/8/2024:

Association types (a usage relationship):

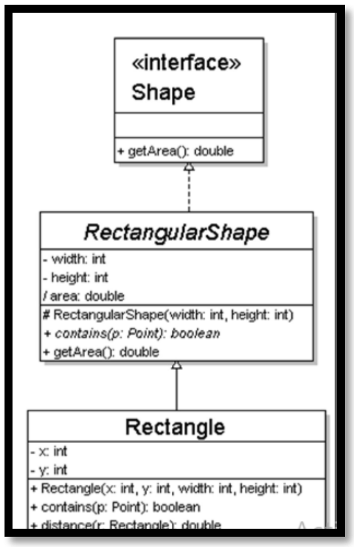
- Aggregation: is part of something
  - Symbolized by: Clear white diamond
- Composition: is fundamentally part of something / is entirely made of
  - This is a stronger connection that aggregation
    - Symbolized by: Black diamond
- Dependency: is used temporarily

- 

Generalization (inheritance) relationships:

- Class: sold line, black arrowhead
- Abstract: solid line, white arrowhead
- Interface: dashed line, white arrowhead

- Hierarchies = drawn top-down
- Arrows = point upward to parent

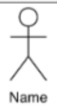
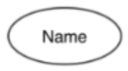
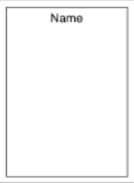

«interface»
Shape
+ getArea(): double
RectangularShape
- width: int
- height: int
/ area: double
# RectangularShape(width: int, height: int)
+ contains(p: Point): boolean
+ getArea(): double
Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double

Associational relationships:

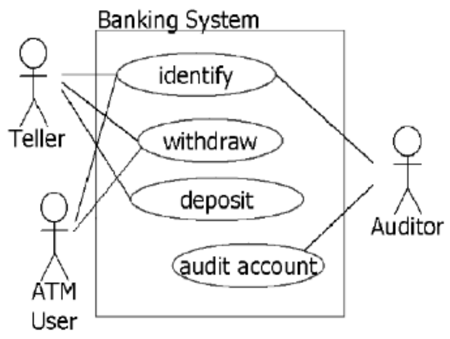
- Multiplicity (how many are used)
- Name
- Navigability

Part 2 Use cases:

- *Use case*: a unit of functionality (requirement), or a service, in the system. A use case isn't a process, program, or function.
    - Nothing OO about use cases.
- A use case is (a generic description of) an entire transaction, involving several objects.
- Purpose: define a piece of behavior of an entity without revealing internal structures of the entity.

Symbol	Name	Interpretation
	Actor	An entity (human or otherwise) external to the system and which interacts with it
	Use case	A service or unit of functionality
	System boundary	Indicates the division between the system being designed and the rest of the world

- Actors: specifies a role played by a user or any other system that interacts with the subject
    - are external to the system; May or may not be human
    - Identified in terms of the roles they play.
    - e.g. Time; A system for devising staff work timetables;
- Actors don't interact with each other, in a use case model.
- Primary actor *accomplishes a goal* via the use case.
    - LHS
- Secondary actor *provides a service* to the use case.
    - RHS
- System boundary: distinguishes the system from the rest of the world.
    - Use Cases are inside the box; Actors are outside.

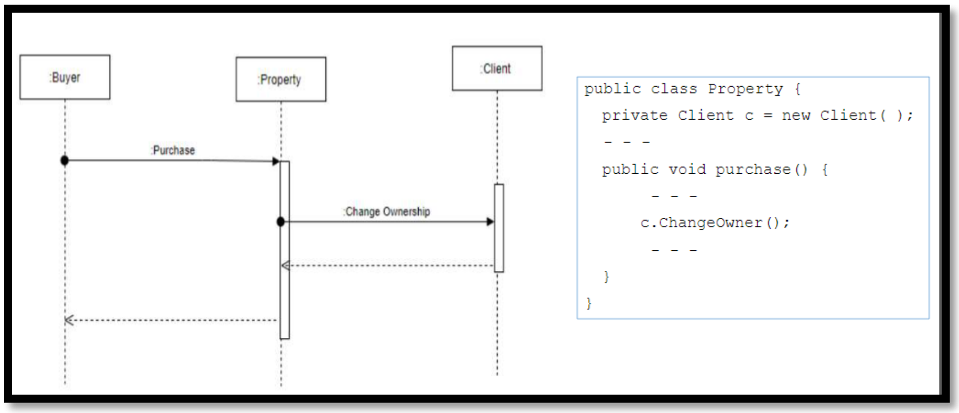
- 

- Use cases are represented by ovals ^:
  - Name of use case inside ellipse/oval
- Stereotype: when one use case is related to another
  - <<includes>> <<uses>> <<extends>>
  - <<extends>> is used for part of use case that will sometimes be used and sometimes not. AKA its use is NOT guaranteed.

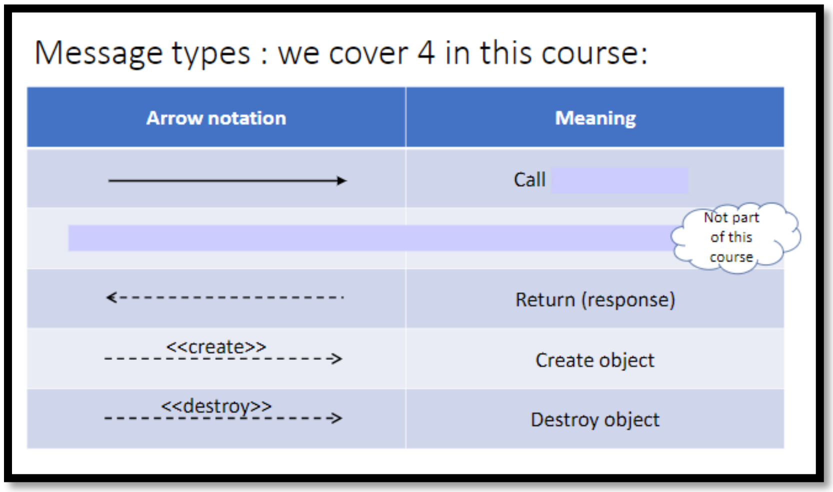
- Use case narratives:
  - Every case a has a detail description
  - Stories of using a system to fulfill a goal
  - If can't describe use-case properly, then look back, rethink whether you have a use case; ask client for more information.
- Use case scenario: a specific sequence of actions, interactions in a use case.
  - A collection of scenario

- Guidelines:
  - Use-cases start with a verb
  - EBP:

(Part 3 – Sequence diagrams):



```
public class Property {
  private Client c = new Client( );
  - - -
  public void purchase() {
       - - -
     c.ChangeOwner();
       - - -
  }
}
```

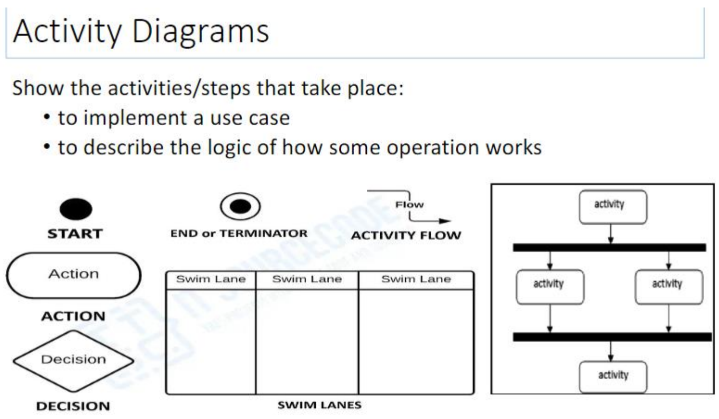


Message types : we cover 4 in this course:

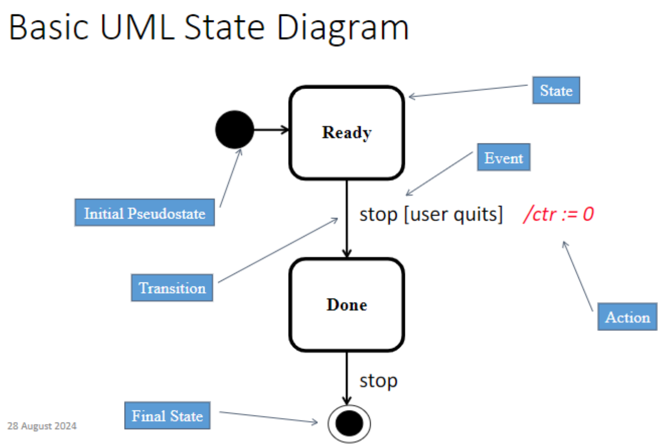
Arrow notation	Meaning
⟶	Call
←- - - - - - - - - - - - - - - -	Return (response)
- - - - <<create>> - - - ->	Create object
- - - - <<destroy>> - - - ->	Destroy object

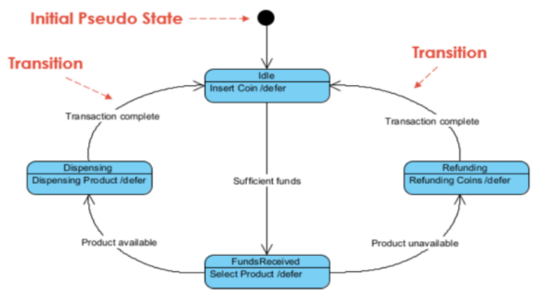
Not part of this course

28/8/2024 (Part 4 –Activity, State & Package Diagrams):

- 
  Activity Diagrams

  Show the activities/steps that take place:
  - to implement a use case
  - to describe the logic of how some operation works

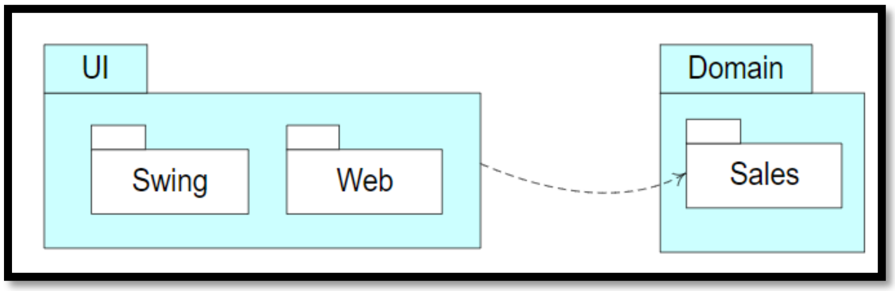
- 
  Basic UML State Diagram

- 

- State machines / state diagrams:
  - Shows how an object changes due to external stress over lifetime
  - Reactive
  - Needs an external diagram

UML Packages:

- Packages group elements
- Used to show the high level organization of a project
- A dashed arrow between packages indicates a dependenc



-

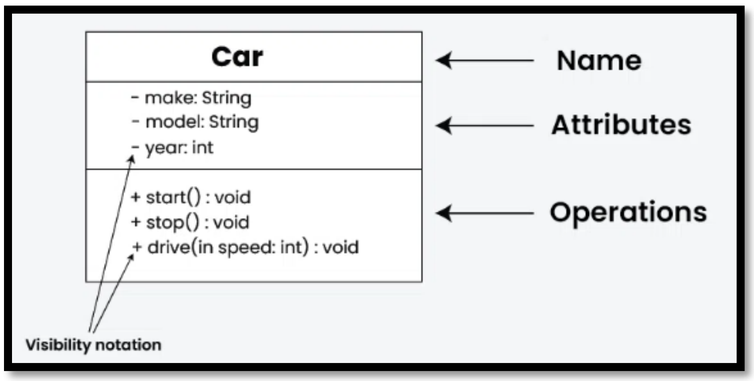
EXTERNAL NOTES:

Recomended resources: https://www.youtube.com/watch?v=WnMQ8HlmeXc

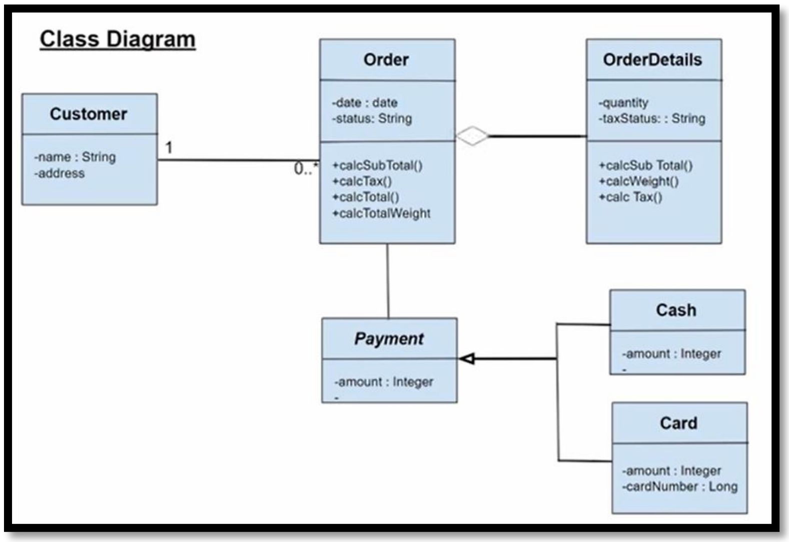
https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/ d

Overview:

- Class: a blueprint or template for creating objects
- Class Diagram: visually represents the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems.
  - Association: represents relations between instances, elements
    - e.g. ~

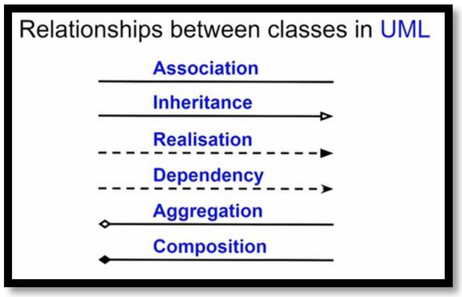


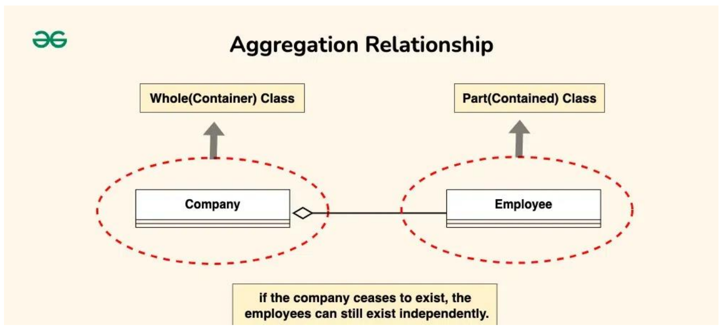
- Classes are depicted as boxes, each containing three compartments for the class: *name*, *attributes*, and *methods*. Lines connecting classes illustrate associations, showing relationships such as one-to-one or one-to-many.

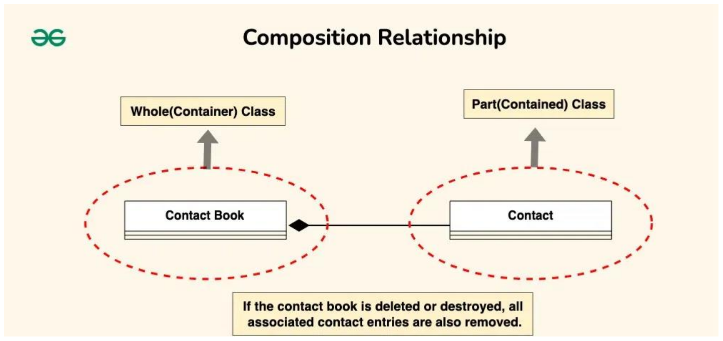


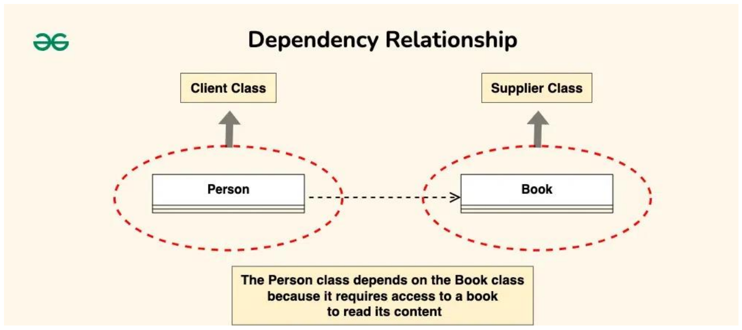
Association types:

- Aggregation: is part of something
- Composition: is fundamentally part of something
    - This is a stronger form of aggregation; indicates more significant ownership.
- Dependency: is used temporarily






Composition Relationship
Whole(Container) Class
Part(Contained) Class
Contact Book
Contact
If the contact book is deleted or destroyed, all associated contact entries are also removed.


Dependency Relationship
Client Class
Supplier Class
Person
Book
The Person class depends on the Book class because it requires access to a book to read its content

The 3 main parameter directionality notations used in class diagrams:

- In (Input):
  - An input parameter is a parameter passed from the calling object (client) to the called object (server) during a method invocation. It is represented by an arrow pointing towards the receiving class (the class that owns the method).
- Out (Output):
  - An output parameter is a parameter passed from the called object (server) back to the calling object (client) after the method execution. It is represented by an arrow pointing away from the receiving class.
- InOut (Input and Output):
  - An InOut parameter serves as both input and output. It carries information from the calling object to the called object and vice versa. It is represented by an arrow pointing towards and away from the receiving class.