CESE4015 Software Systems

Unified Modeling Language: An Introduction

Guohao Lan

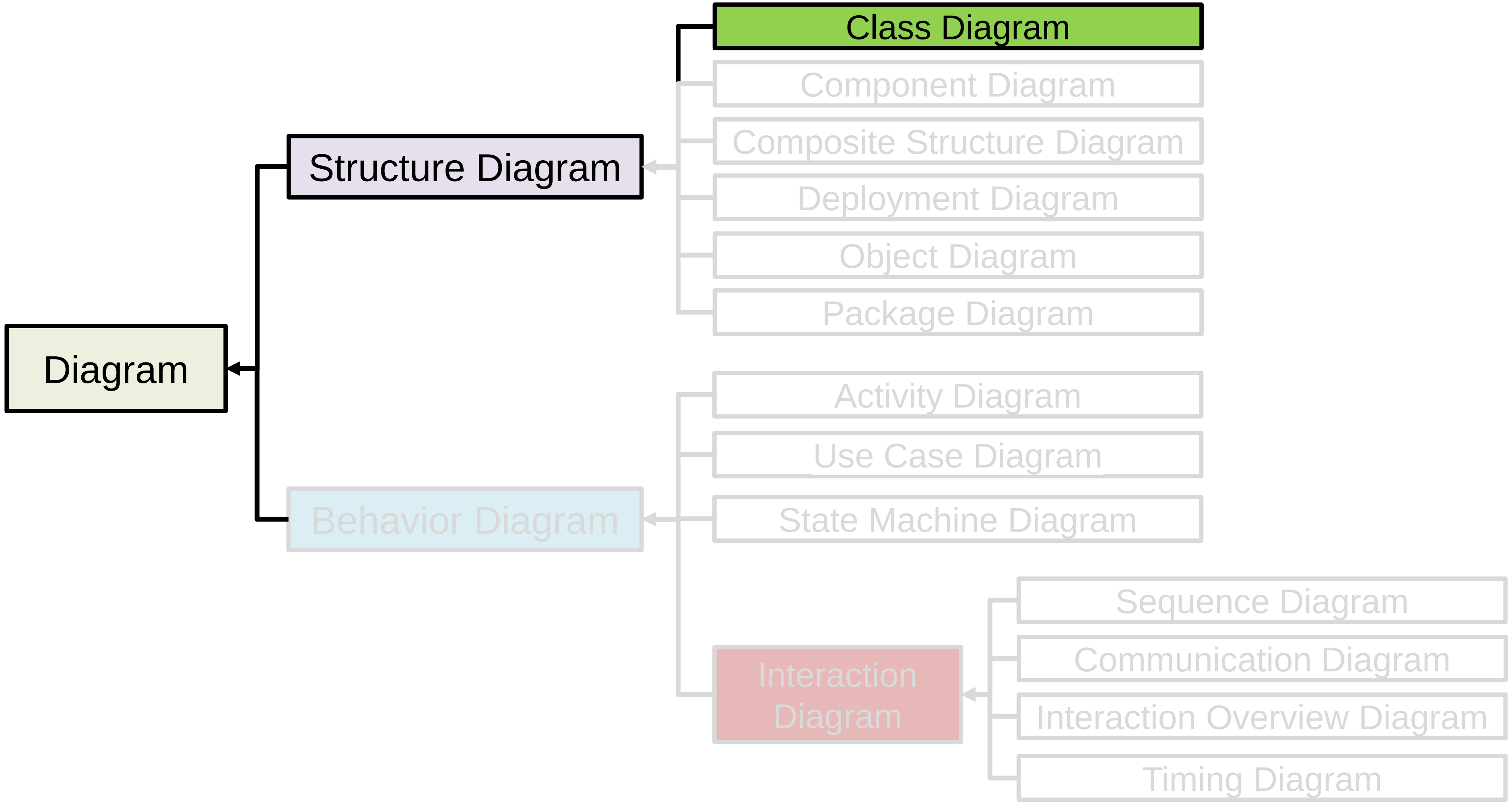
Embedded Systems Group

December 20th 2022

Agenda for UML

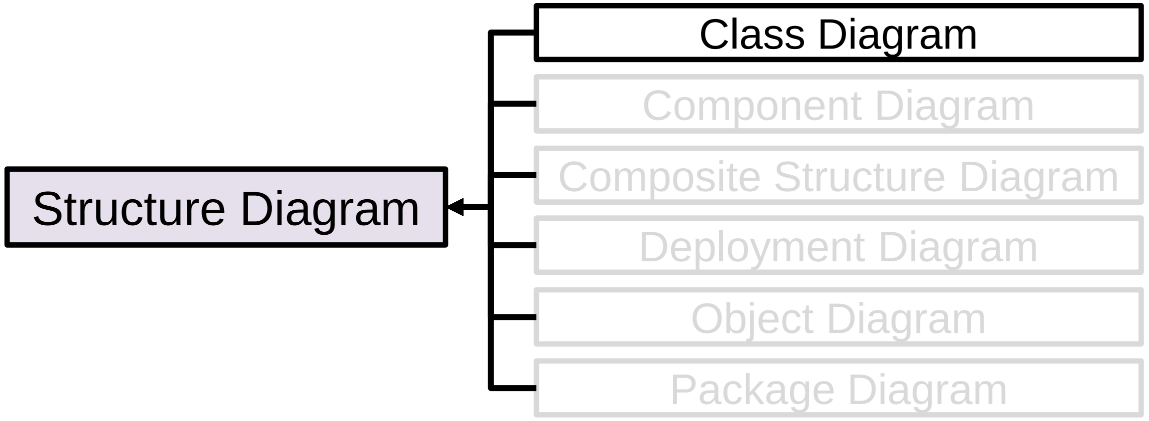
- Week 5 Lecture:
  - Background of UML
  - Use Case, Component, Deployment
- Week 5 Lab:
  - Modeling with UML diagrams (part 1)
- Week 6 Lecture:
  - Class, Sequence
- Week 6 Lab:
  - Modeling with UML diagrams (part 2)

Class Diagram



Class Diagram (cont.)

- Class diagram:
  - Is a type of structural diagram:
    - Emphasizes the static structure of the system and the things that must be presented in the system, including objects, attributes, operations, and relationships.
    - Used extensively in documenting the architecture of the software systems.



Class Diagram (cont.)

- What is a class diagram?

  ▪ **Class Diagram:** describes the classes (types of objects) in the system and the various kinds of static relationships that exist among them.

  – It shows:
    - The static properties and operations of classes and the constraints that apply to the way **objects** are connected.

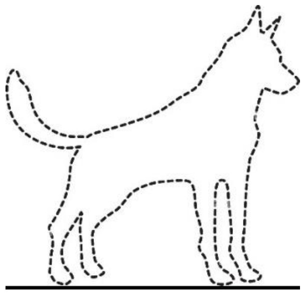
  – It does not show:
    - How the classes are interacted.
    - The implementation details.

Class Diagram (cont.)

- Difference between a Class and an Object?
  - A class represents the type of the object and is a blueprint for an object.
  - A class describes what an object will be, but it is not the object itself.

Properties	**Functions**
Color	Sit
Height	Eat
Length	Shake
Weight	Run

Class



Create instance →

Object

Properties

Color: Flesh

Height: 15cm

Length: 30cm

Weight: 2kg

Class Diagram (cont.)

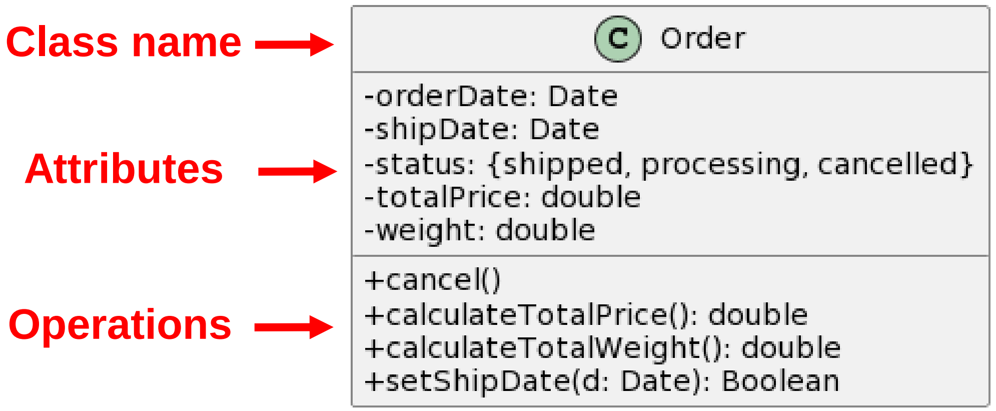
- Difference between a Class and an Object?
  - A class represents the type of the object and is a blueprint for an object.
  - A class describes what an object will be, but it is not the object itself.

  - Object-Orientation “features” in Rust:
    - Using traits to define shared behavior in an abstract way.
    - Using struct to achieve the “purpose of class:
    - References: *https://doc.rust-lang.org/book/ch17-02-trait-objects.html*
    - *https://jimmco.medium.com/classes-in-rust-c5b72c0f0a4c*

Class Diagram (cont.)

- Diagram of one class:
  - **Class notation:** contains three parts - class name, attributes, and operations.

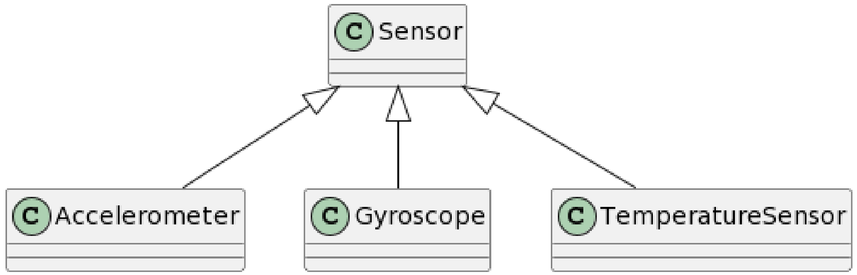
- Class name in top of the box
- Attributes should include all fields of the object
- Operations should not include inherited methods



Class name
Attributes
Operations
C
Order
-orderDate: Date
-shipDate: Date
-status: {shipped, processing, cancelled}
-totalPrice: double
-weight: double
+cancel()
+calculateTotalPrice(): double
+calculateTotalWeight(): double
+setShipDate(d: Date): Boolean

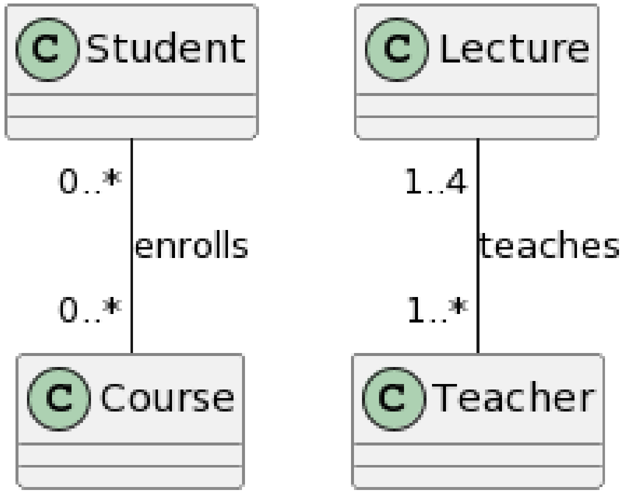
Class Relationship (cont.)

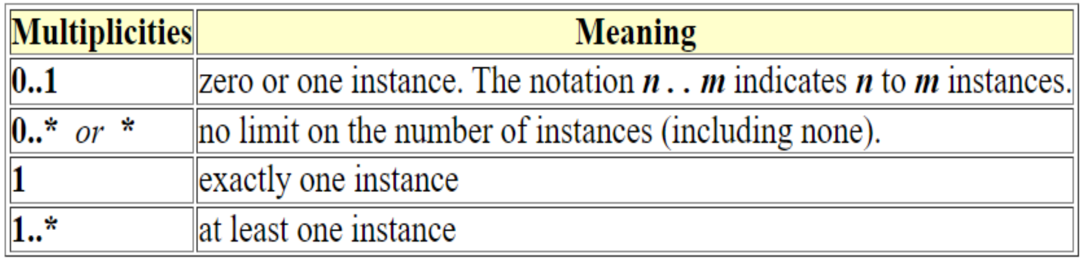
- Class relationships:
  - **Generalization:** an inheritance relationship
    - Represents an “is-a” relationship
    - A solid line with a hollow arrowhead that points from the child to the parent class.
    - An important concept in object-oriented design.
    - The ability of one class to inherit the identical functions or properties of another class.



Class Relationship (cont.)

- Class relationships:
  - **Simple association:**
    - A solid line connects two classes.
    - Different types of cardinality.

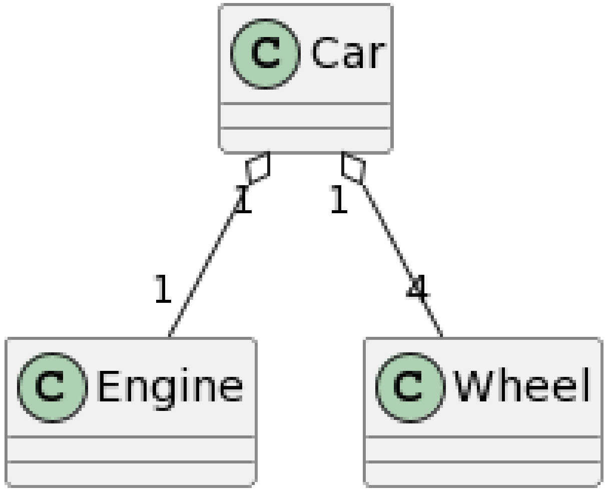




Multiplicities	Meaning
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

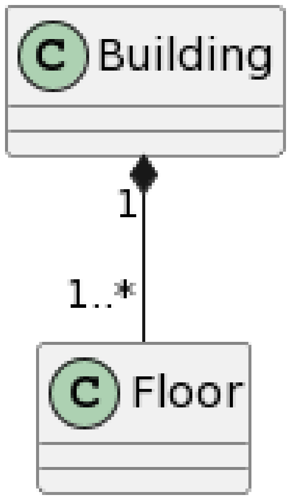
Class Relationship (cont.)

- Class relationships:
    - **Aggregation:** represents a "is part of" relationship
        - A solid line with an **unfilled diamond** at the association end connected to the class of composite.
        - Objects of Class A and Class B have separate lifetimes:
            - *The lifecycle of* ***a part of Class*** *is independent from the* ***whole class's lifecycle****.*



Class Relationship (cont.)

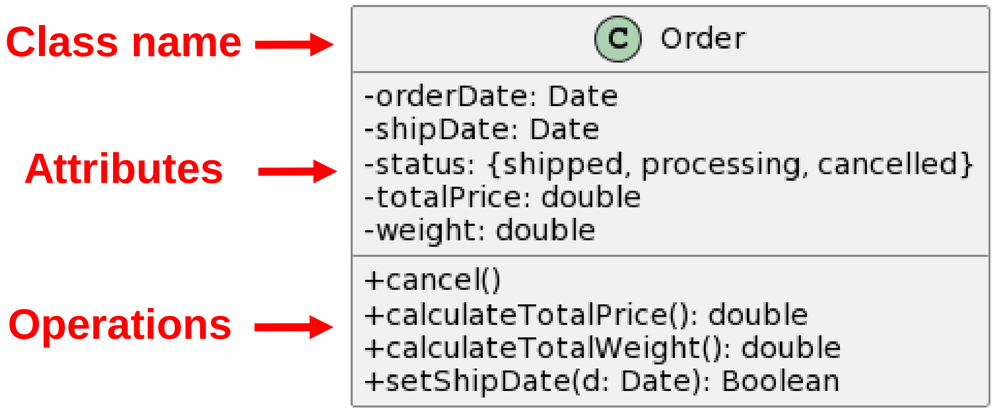
- Class relationships:
  - **Composition:** represents a “is entirely made of” relationship
    - A solid line with a **filled diamond** at the association end connected to the class of composite.
    - Objects of Class A and Class B have the same lifetime.
      - *The lifecycle of **a part of Class** is dependent on the **whole class’s lifecycle**.*



Class Diagram (cont.)

- Diagram of one class:
  - **Class notation:** contains three parts - class name, attributes, and operations.

- Class name in top of the box
- Attributes should include all fields of the object
- Operations should not include inherited methods


Class name
Attributes
Operations
C
Order
-orderDate: Date
-shipDate: Date
-status: {shipped, processing, cancelled}
-totalPrice: double
-weight: double
+cancel()
+calculateTotalPrice(): double
+calculateTotalWeight(): double
+setShipDate(d: Date): Boolean

Class Diagram (cont.)

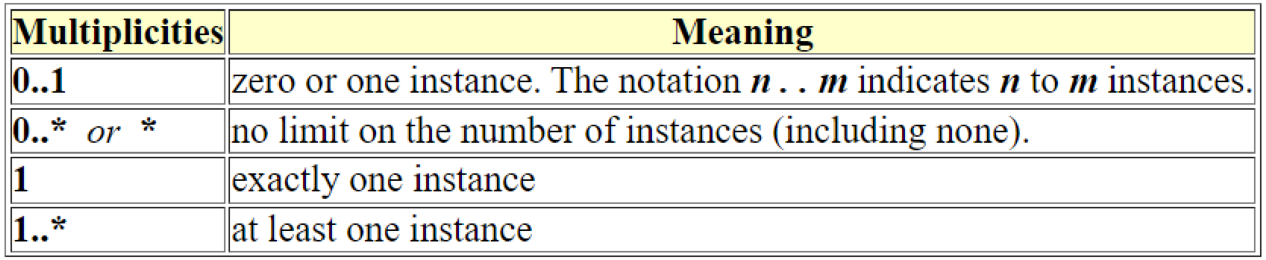
- Class attributes:
  - **Syntax:**
    visibility name : data_type [multiplicity] = default_value

– (1) Visibility:
  - + public: accessible to everything
  - # protected: accessible to class, package, and subclasses
  - - private: accessible to the class only
  - ~ package (default): accessible to class and package

Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

Class Diagram (cont.)

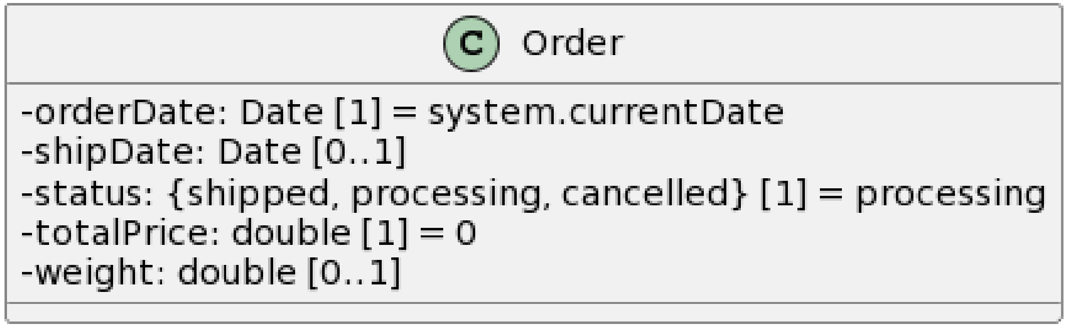
- Class attributes:
  - **Syntax:**
    visibility name : data_type [multiplicity] = default_value
  - (2) Multiplicity:



Multiplicities	Meaning
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

Class Diagram (cont.)

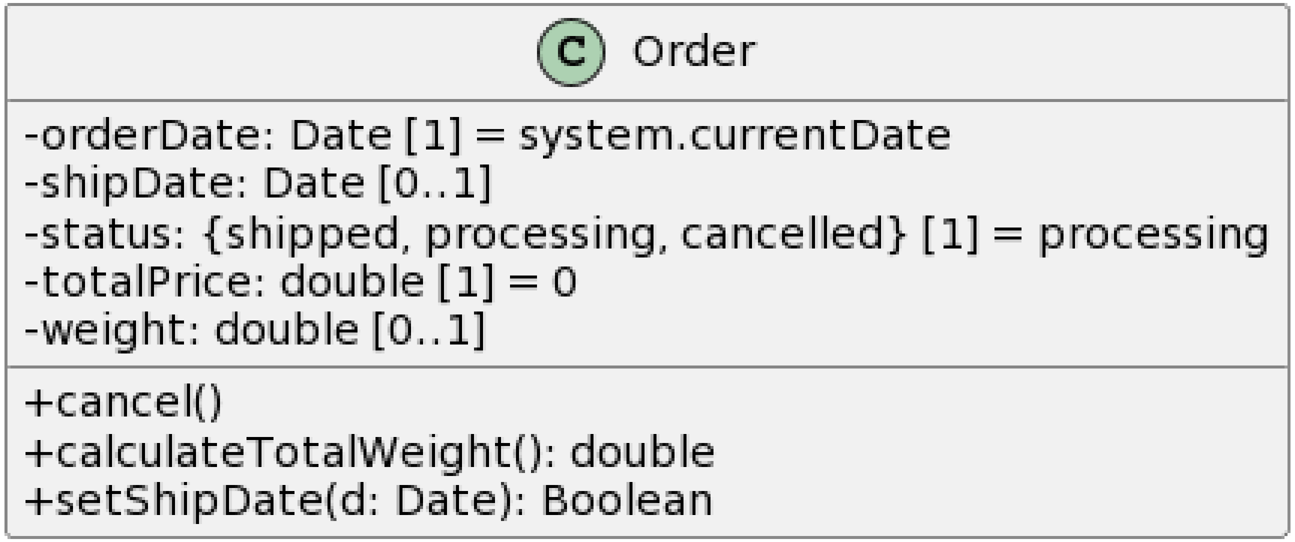
- Class attributes:
  - **Syntax:**
    visibility name : data_type [multiplicity] = default_value
  - An example:



```
-orderDate: Date [1] = system.currentDate
-shipDate: Date [0..1]
-status: {shipped, processing, cancelled} [1] = processing
-totalPrice: double [1] = 0
-weight: double [0..1]
```

Class Diagram (cont.)

- Class operations:
  - **Syntax:**
    visibility name (parameter-list) : return-type
  - An example:

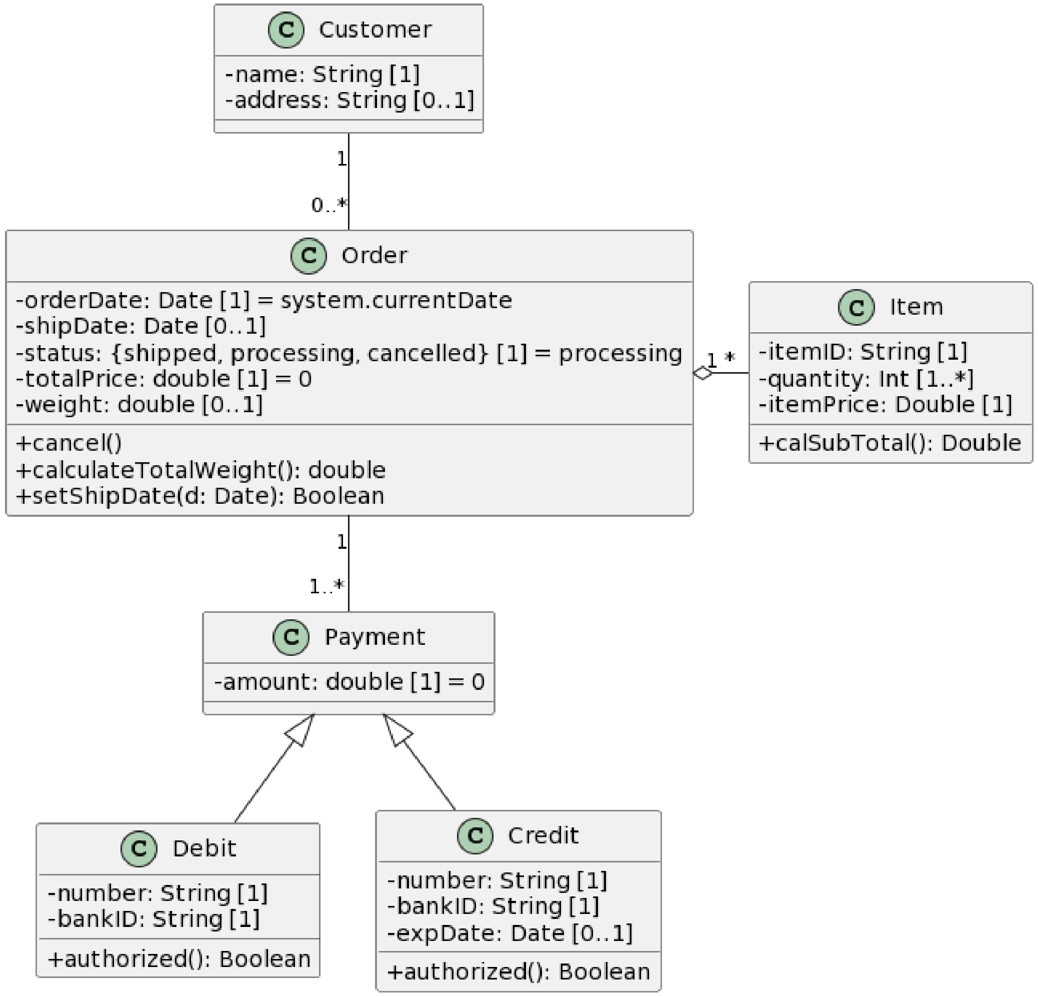


```
-orderDate: Date [1] = system.currentDate
-shipDate: Date [0..1]
-status: {shipped, processing, cancelled} [1] = processing
-totalPrice: double [1] = 0
-weight: double [0..1]

+cancel()
+calculateTotalWeight(): double
+setShipDate(d: Date): Boolean
```

Class Diagram (cont.)

- An example:



Class Diagram

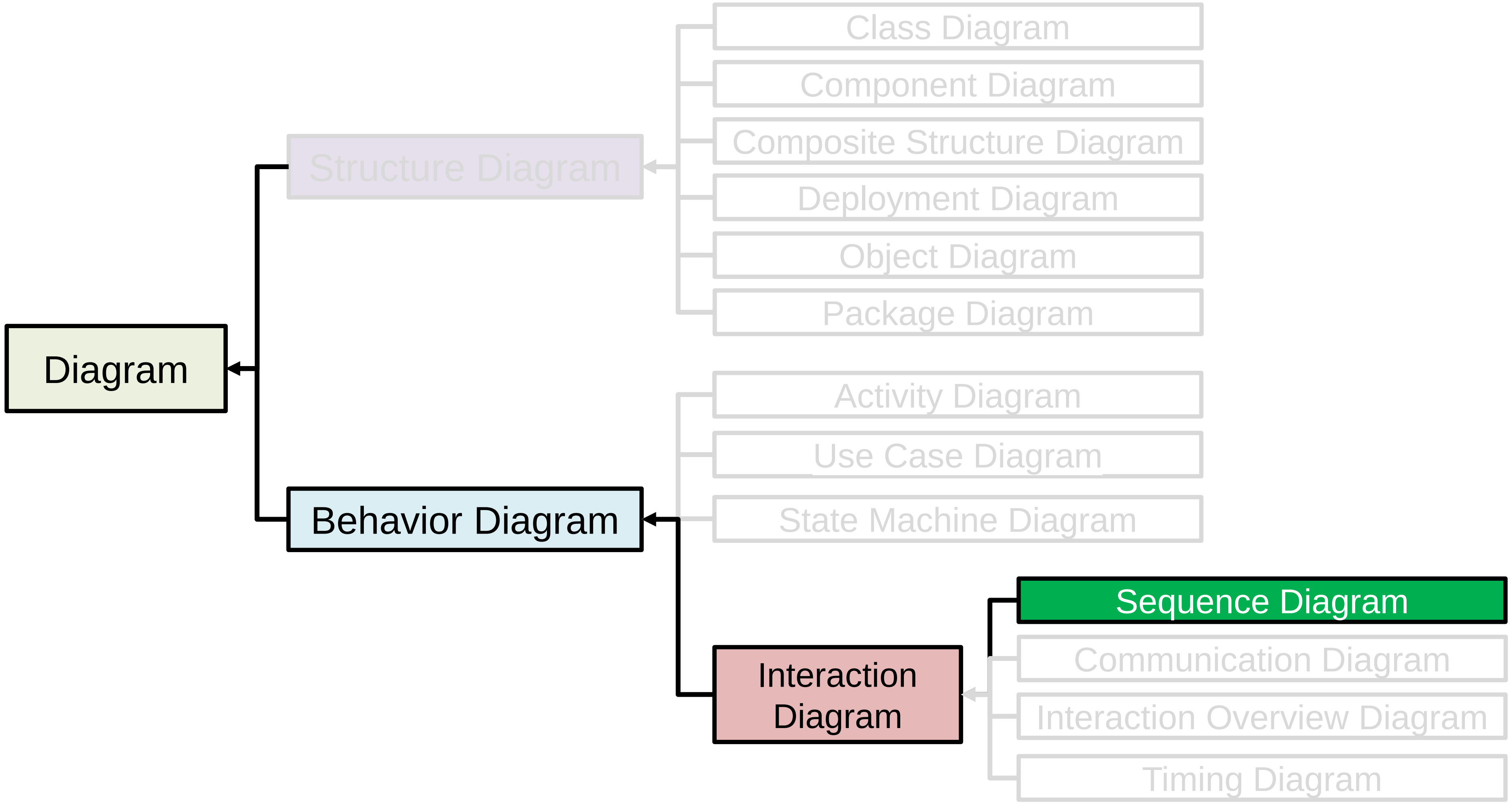
- Short summary:

  ▪ **Class Diagram:** describes the classes (types of objects) in the system and the various kinds of static relationships that exist among them.

  – When to use:
    - Describes the structure of a system by showing its classes (operations and attributes) and the relationships among them.
    - Useful in conceptual modeling of the structure of the system, and helpful in translating the models into programming code.

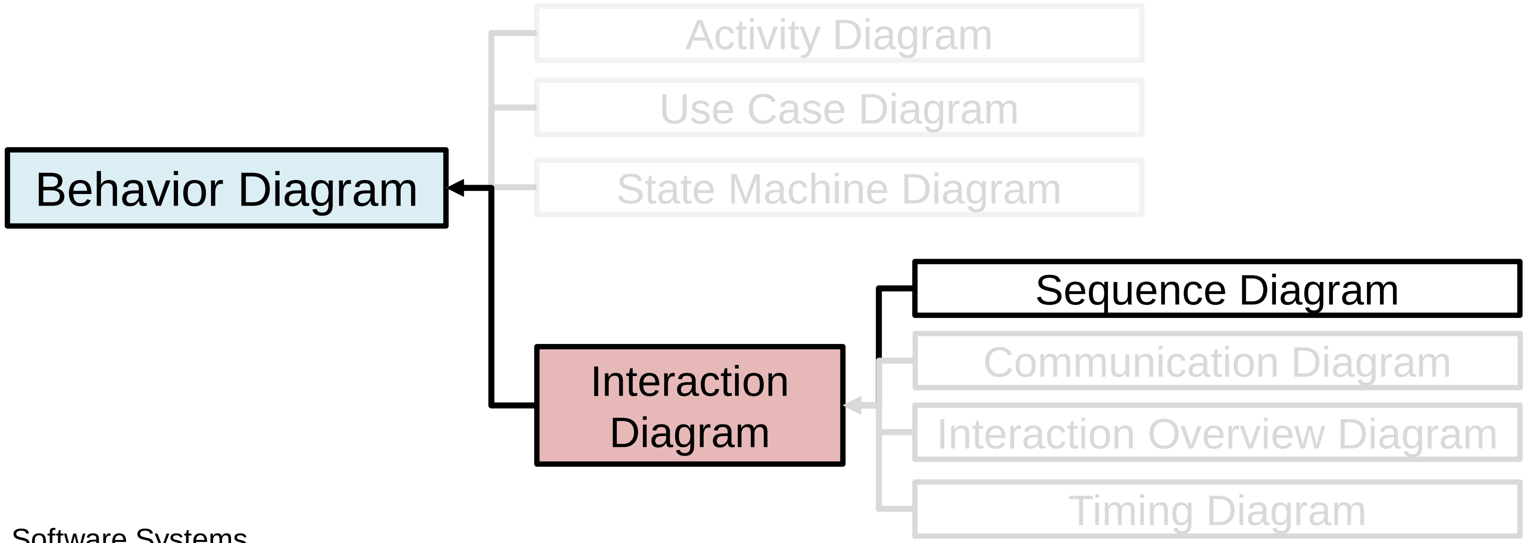
  – It does not show:
    - How the classes are interacted.
    - The implementation details.

Sequence Diagram



Sequence Diagram (cont.)

- Sequence diagram:
  - Focuses on the dynamic behavior of the systems and changes to the internal states of objects.
  - Interaction diagrams:
    - Interaction: emphasize the flow of control, showing collaborations among objects; how objects communicate;



Sequence Diagram (cont.)

- What is the Sequence Diagram?

  ▪ **Sequence Diagram:** an “interaction diagram” that models a single scenario execution in the system. The diagram shows how example objects interact with each other and the messages that are passed between them.

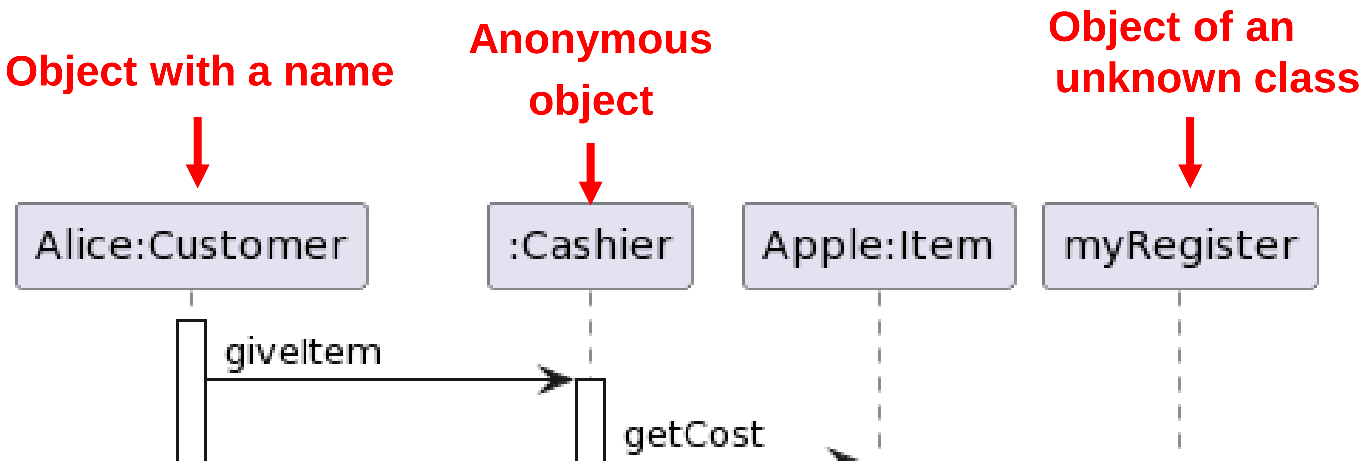
  – Show high-level overview of relationship between use cases, actors, and the system.
  – It is a behavioral diagram.
  – Does not provide a lot of details.

Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Participant:** object that acts in the diagram.
    - Squares with object type, optionally preceded by “name:”

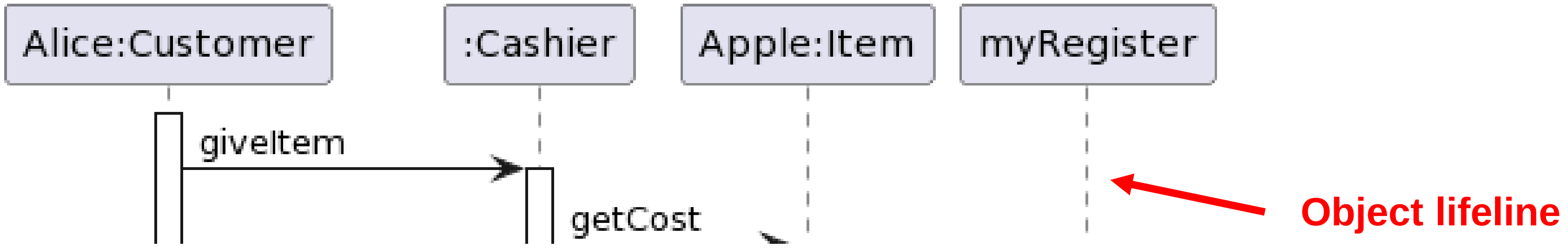
Name syntax: <objectname>:<classname>

- Object can be specify (with a name) or general (without a name to represent any object in that class).



Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Participant:** object that acts in the diagram.
    - Squares with object type, optionally preceded by “name:”
  - **Lifeline:** represents the period of time that an object exists.
    - Represented by dashed vertical line.



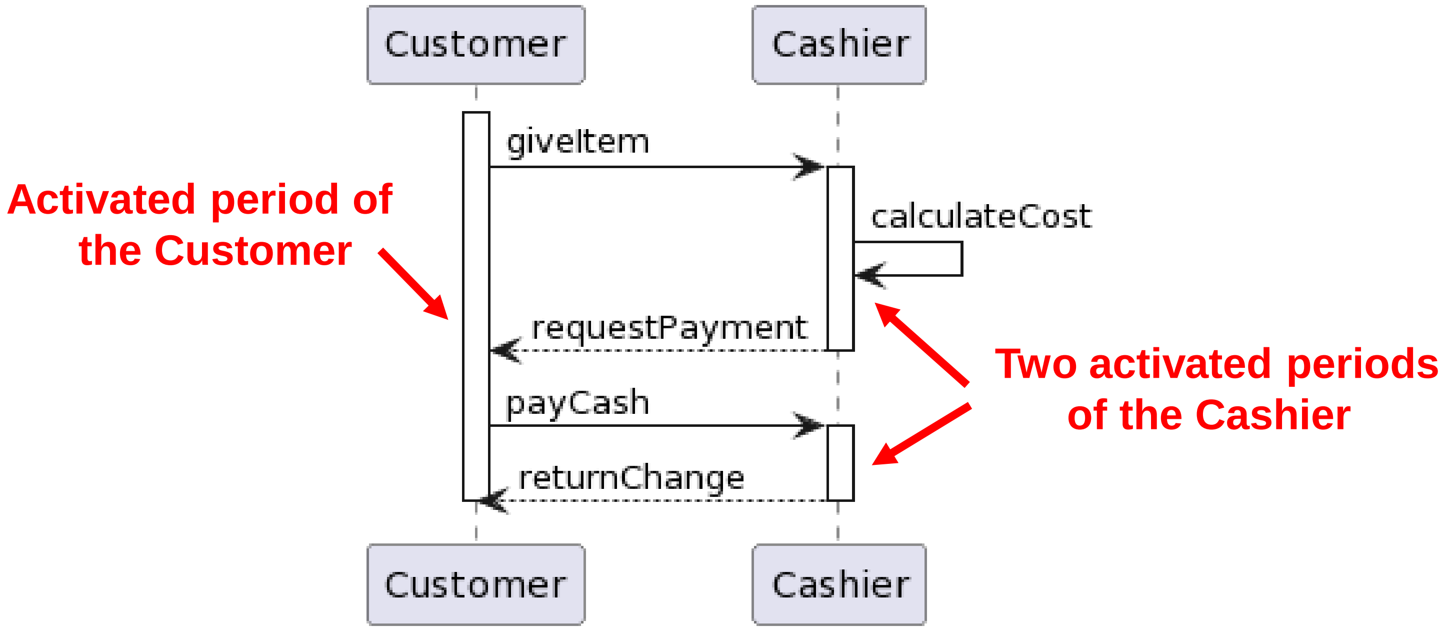
Name syntax: <objectname>:<classname>

Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Participant:** object that acts in the diagram.
    - Squares with object type, optionally preceded by “name:”
  - **Lifeline:** represents the period of time that an object exists.
    - Represented by dashed vertical line.
  - Participants in the system take the responsibility in *managing the data, processing the data, moving data around the system, handling requests, and many other operations*.

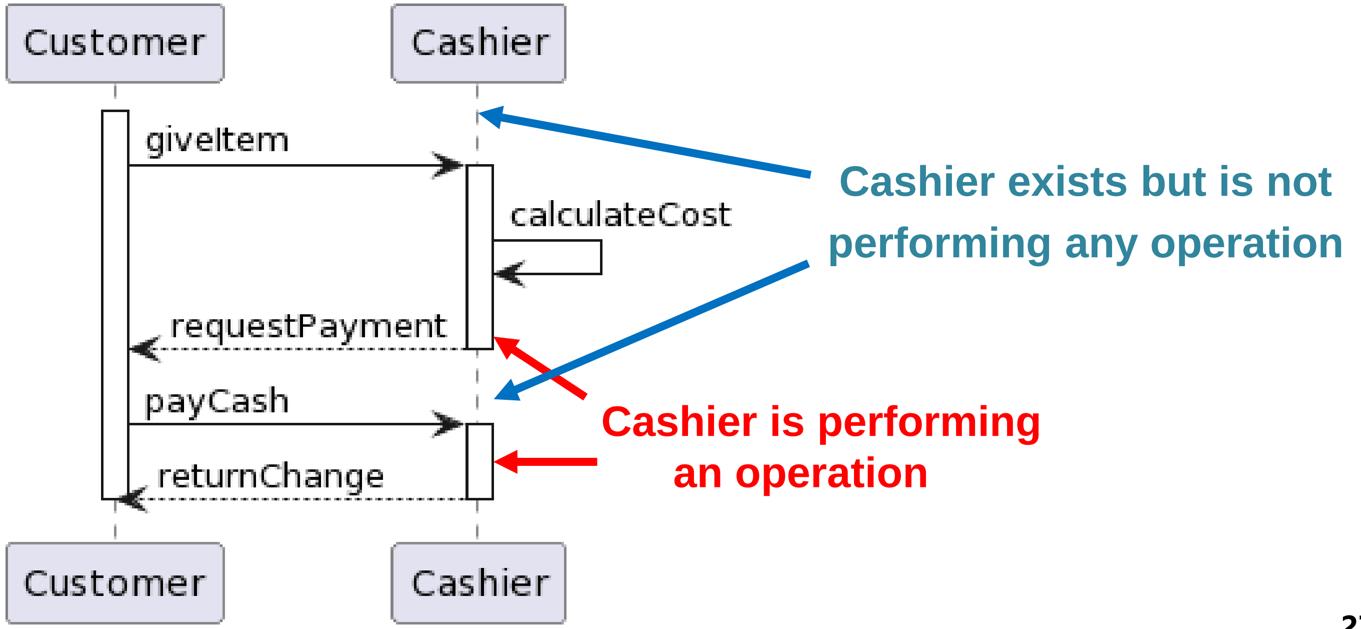
Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Activation**: a thin rectangle on the lifeline that represents the period during which a participant is performing an operation (e.g., running its code or waiting for another participant's method to finish).



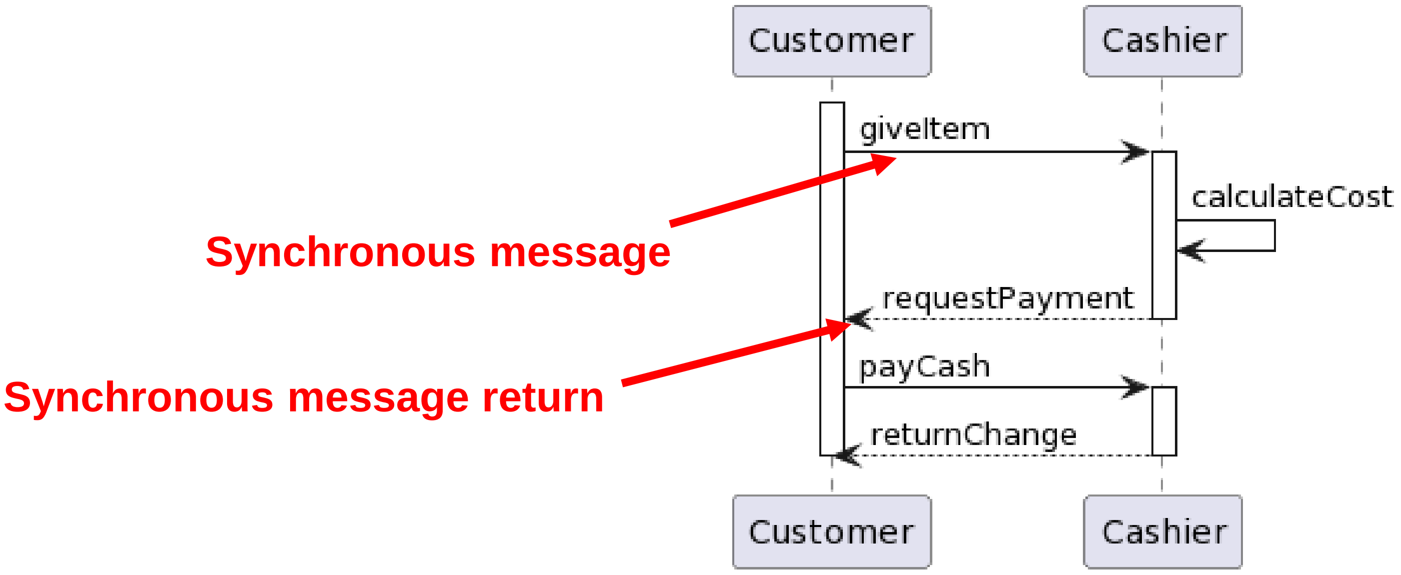
Sequence Diagram (cont.)

- Difference between activation and lifeline?
  - **Activation**: a thin rectangle on the lifeline that represents the period during which a participant is performing an operation (e.g., running its code or waiting for another participant's method to finish).
  - **Lifeline:** represents the time that an object (participant) exists.


Customer
Cashier
giveItem
calculateCost
requestPayment
payCash
returnChange
Customer
Cashier
Cashier exists but is not performing any operation
Cashier is performing an operation

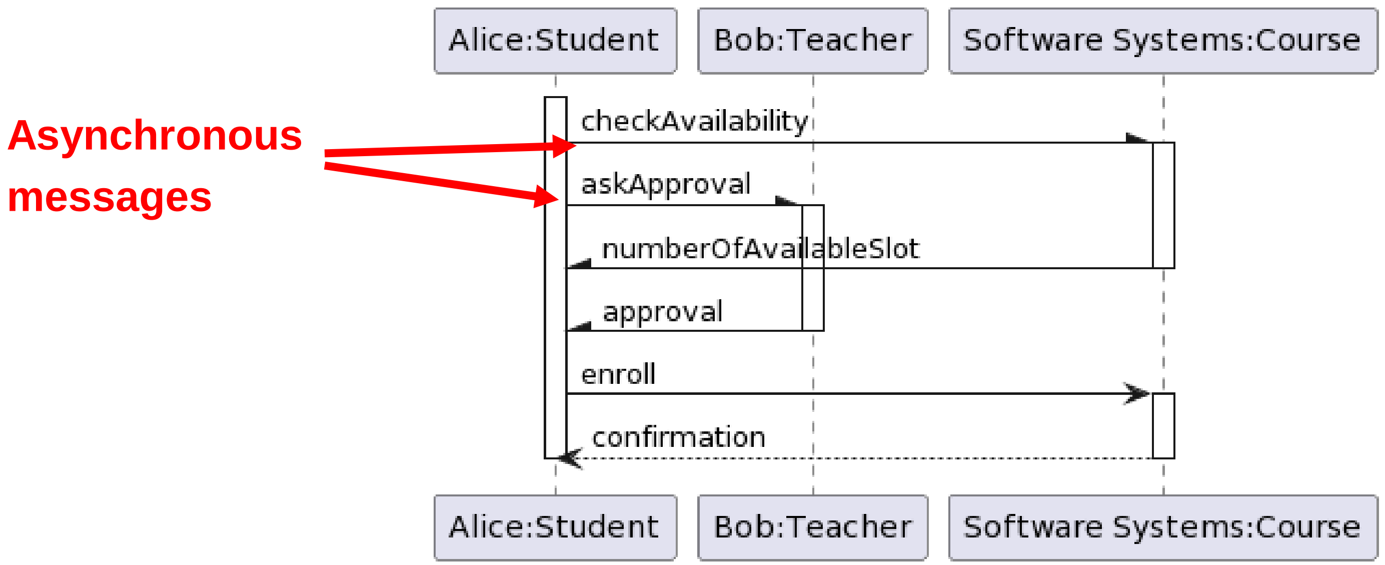
Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Message (method call)**: communication between participants.
    - Synchronous message and return.
      - If the caller sends a synchronous message, it **must wait** until it receives a response (message return) from the target.



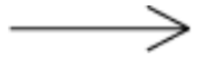
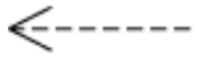
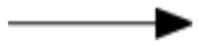
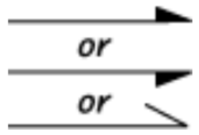
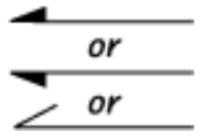
Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Message (method call)**: communication between participants.
    - Asynchronous message: allows the sender to send additional messages while the original one is being processed.



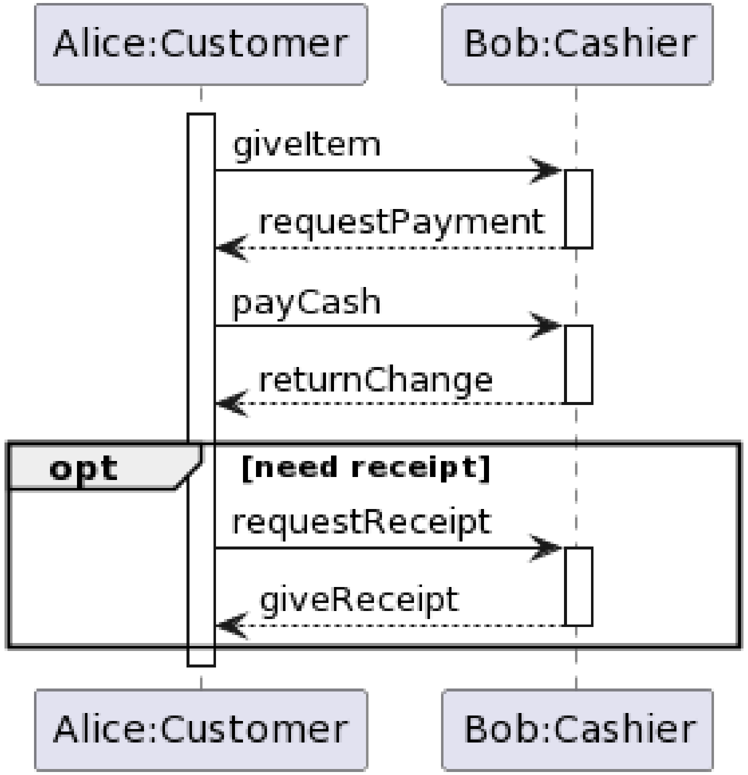
Sequence Diagram (cont.)

- Common elements in a sequence diagram:
    - Summary of different message conventions in UML:

Symbol	Meaning
	simple message which may be synchronous or asynchronous
	simple message return (optional)
	a synchronous message
	an asynchronous message
	an asynchronous message return

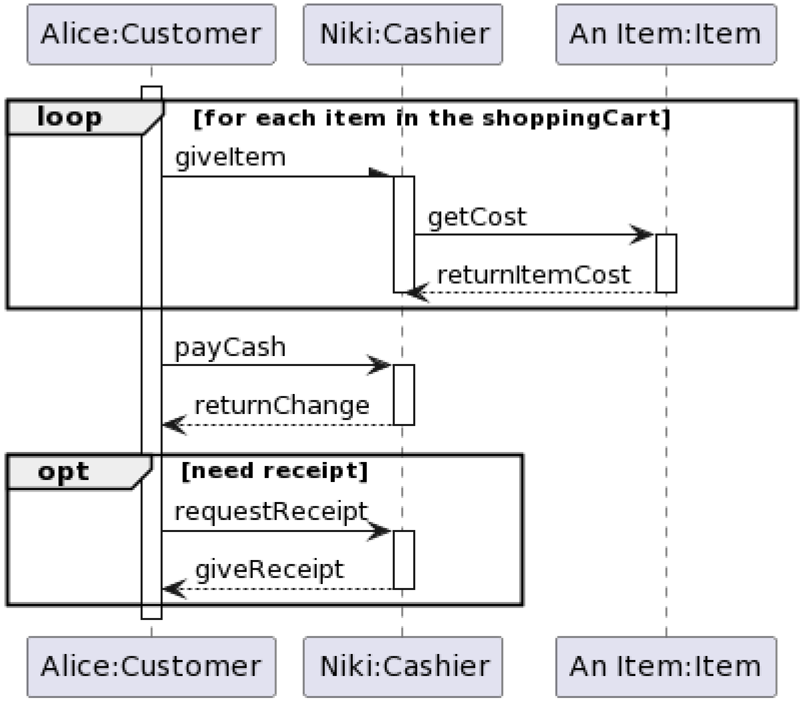
Sequence Diagram (cont.)

- Selection and loop:
  - **(opt) [condition]:** the fragment executes only if the supplied condition is true;



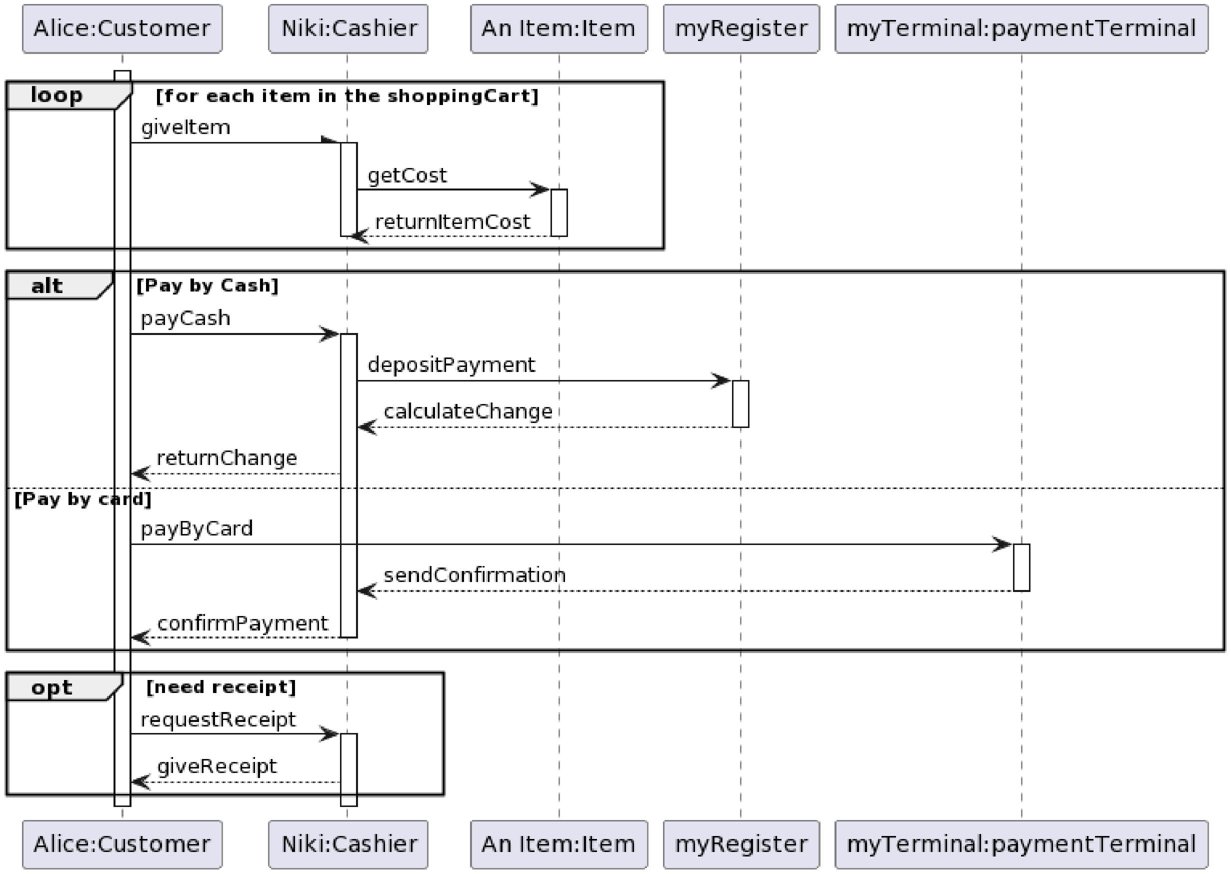
Sequence Diagram (cont.)

- Selection and loop:
  - **(loop) [condition or items to loop over]:** the fragment may execute multiple times if the supplied condition is true;



Sequence Diagram (cont.)

- Selection and loop:
  - **(alt) [condition]:** alternative multiple fragments = if / elseif/ else;



Sequence Diagram (cont.)

- When to use the Sequence Diagram?
  - To show the interaction between several objects within a single use case (usage scenario).
  - To explore the logic of a use case.

Closing remarks

- In the Lab session:
  - Go over the tutorial for Class and Sequence diagrams.
  - Work on the Class and Sequence diagrams in the modeling assignment.